Теория на Графите

работна версия

Минко Марков

Съдържание

Списък на фигурите

Списък на допълненията

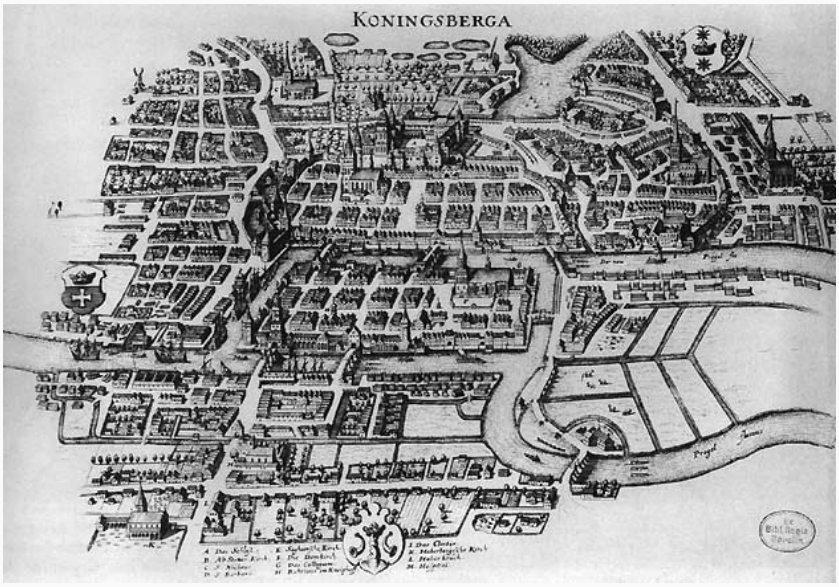
Списък на теоремите

Глава 1

История на теорията на графите

Терминът “graph” се появява за първи път в статия в *Nature* на известния математик James Sylvester през 1878 г.[†], но началото на теорията на графите е поставено век и половина преди това от гениалния математик Leonard Euler. През 1735 г. Euler решава една нерешена дотогава задача, известна като *задачата за седемте моста на Königsberg*. Преди да формулираме задачата ето малко пояснения. В град Königsberg[‡] в Източна Прусия тече река Pregel. В града насред реката има два големи застроени острова един до друг – Kneiphof (K) и Lomse (L). Кварталът на северния бряг е Altstadt (A), а на южния, Vorstadt (V). Kneiphof е свързан с два моста с Altstadt, с други два моста с Vorstadt и с един мост с Lomse. Lomse освен това е свързан с един мост с Altstadt и с един мост с Vorstadt. Карта на града с реката и двата острова е показана на Фигура 1.1. Картата е взета от *уикипедия*.

Фигура 1.1 : Карта на Königsberg.

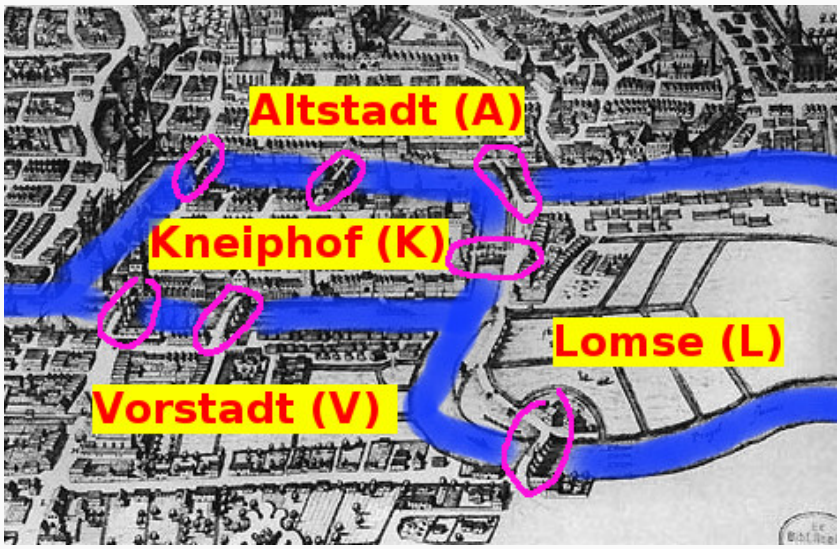


[†]Става дума за “graph” в смисъла на дискретната математика; на английски “graph” означава също и графика на функция, което е съвършено различно нещо. James Joseph Sylvester е създал или въвел в днешния смисъл много математически термини, например “matrix” и “discriminant”, които ползваме и днес.

[‡]Град Königsberg вече не съществува. Градът е основан през 1255 г. от тевтонските рицари край брега на Балтийско море и до края на Втората Световна Война е главният град на германската територия Източна Прусия. През войната градът е почти изцяло разрушен. Победителите във войната отнемат Източна Прусия от Германия завинаги. Königsberg заедно с територията около него стават съветски, а построеният наново град е наречен Калининград. След разпадането на Съветския Съюз, въпросната територия заедно с Калининград се оказва руски анклав между Полша и Литва на брега на Балтийско море.

Фигура 1.2 показва схема на града с четирите квартала K, L, A и V, реката и седемте моста.

Фигура 1.2 : Карта на Königsberg с означения.



Показани са островите Kneiphof (K) и Lomse (L), кварталите Altstadt (A) и Vorstadt (V), река Pregel в синьо и седемте моста.

Самата задача е:

> *Да се намери разходка в града, която започва в някой от четирите квартала K, L, A или V, минава през всеки от седемте моста точно по веднъж, и завършва в същия квартал, в който е започнала.*

В оригиналната статия на Euler няма изискване разходката да завършва там, където е започнала. Дали жителите на града са искали да разходката да е затворен маршрут, а Euler е пропуснал това изискване, или не, няма особено значение. В момента можем да решим с лекота задачата и в двата варианта (вж. Секция 2.10). Тук допускаме, че разходката трябва да завършва там, където е започнала.

Хората от Königsberg дълго се опитвали да открият такава разходка в града и не успявали. Euler, който по това време е работил в Академията в близкия Санкт Петербург, решил задачата при посещение в Königsberg, доказвайки, че такава разходка не съществува. Решението използва математически обект, който днес бихме нарекли *неориентиран мултиграф*. Оригиналното решение на Euler [24][†] не съдържа термина *граф* никъде, нито съдържа рисунка на граф. Но по същество неговото решение се основава на това, което днес наричаме граф. Той използва по една буква за всеки от четирите квартала, по една буква за всеки от седемте моста, и разглежда възможните решения като стрингове от тези букви – нещо, което днес бихме нарекли *цикъл в граф*. Още в самото начало на статията си Euler отбелязва, че тази геометрична задача е коренно различна от нормалните геометрични задачи, които включват някакви големини. Euler говори за нов тип геометрия[‡], въведена от Лайбниц, в която няма големини и пресмятания на големини и която се занимава само с взаимните позиции; Ойлер я нарича "геометрия на позициите".

[†]Сканирано копие на оригиналната статия на Euler е свободно достъпно онлайн. Оригиналната статия обаче е на латински, което я прави недостъпна за масовия съвременен читател. Пълен превод на английски на тази статия има, например, в книгата на Biggs, Lloyd и Wilson [12].

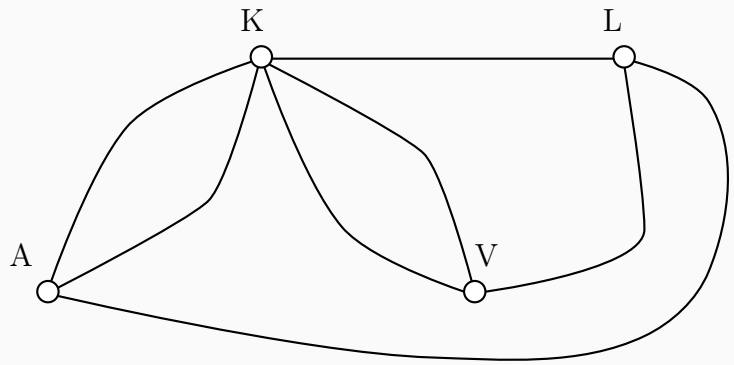
[‡]Която днес бихме нарекли *топология*.

Следното популярно решение на задачата е основано на решението на Euler. Ключовото наблюдение е, че за целите на тази задача, фóрмата и размерите на частите от реката, островите, кварталите и мостовете нямат никакво значение. Единственото, което има значение, е измежду четирите обекта K, L, A и V, кой с кой е свързан, и при това с колко моста. Напълно достатъчно е да се мисли за *абстракция* на града, в която всеки от K, L, A и V е просто една точка, като някои двойки точки са свързани с някакви връзки, а други, не. Тези връзки, естествено, отговарят на мостовете. Може да се мисли за схема, в която:

- четирите точки K, L, A и V са нарисувани в някакво подходящо разположение в равнината, което може да няма нищо общо с това, как са разположени съответните квартали в реалния град, и
- на всеки от седемте моста отговаря една връзка между съответните точки, чиито първообрази свързва въпросният мост, и
- всяка връзка между две точки е отбелязана с линия между тях, като геометрията на тези линии може да няма нищо общо с геометрията на реалните мостове.

Пример за такава схема има на Фигура 1.3. Забелязваме, че взаимното разположение на точките на Фигура 1.3 няма нищо общо с разположението на съответните квартали в града. Важното е, че точка K е свързана с точка A посредством две различни връзки-линии, които отговарят на двата моста между Kneiphof и Altstadt, и така нататък.

Фигура 1.3 : Абстракция на Königsberg.



За да решим задачата, достатъчно е да разгледаме този обект.

Всяка разходка в града, която минава през седемте моста точно веднъж и завършва в квартала, в който е започнала, може да се представи като разходка в обекта, нарисуван на Фигура 1.3, която минава през всяка линия точно веднъж и завършва в тази точка, в която е започнала. Euler забелязал, че такава разходка има само ако:

- за всяка точка има **четен брой** линии, които излизат от нея. Защо? – защото при такава разходка, на всяко “излизане” от точката трябва да съответства точно едно “влизане” след това, и
- има възможност да идем от всяка точка до всяка друга точка.

В съвременната терминология казваме, че всеки *връх* на графа трябва да има четна *степен* и графът трябва да е свързан, за да има *Ойлеров цикъл* (вж. Секция 2.10). Задачата за мостовете на Königsberg няма решение, защото има върхове от нечетна степен. Прочее, всички върхове са от нечетна степен, но е достатъчно да има поне един връх от нечетна степен[†], за да няма решение.

Повече информация за историята на Königsberg и мостовете на Pregel има в [30].

[†]Както става ясно от Следствие 1, броят на върховете от нечетна степен трябва да е четно число, така че не може да има точно един връх от нечетна степен.

Глава 2

Неориентирани графи и неориентирани мултиграфи

2.1 Обикновени графи.

2.1.1 Определение

Определенията в тази секция са близки до определенията в [27]. Подчертаваме, че всички множества, които разглеждаме, са крайни.

Съвременното понятие "граф" е чисто теоретико-множествено. Определение 1 няма нищо геометрично или топологично в себе си. За да разберем какво е "граф", достатъчно е да знаем какво е "множество", "подмножество", "наредена двойка" и "мощност на множество".

Определение 1: Граф, върхове и ребра

Граф е наредена двойка $G = (V, E)$, където V е непразно множество, чиито елементи се наричат *върхове*, E е множество, чиито елементи се наричат *ребра*, като

$$E \subseteq \{X \subseteq V : |X| = 2\}$$

Обикновено върховете се записват с малки латински букви като u, v и т. н. Имената на ребрата обикновено се записват като e_1, e_2 и т. н. Тъй като всяко ребро е двуелементно множество от върхове, естествено е да определяме ребрата като двуелементни множества, записвайки например "$e_1 = \{u, v\}$". Такъв запис обаче **не се използва**, а се използва "$e_1 = (u, v)$". Това противоречи на нашата конвенция, че фигурните скоби се използват при липса на наредба, а кръглите, при наличие на наредба. Но ние сме длъжни да се съобразим със световно приетата конвенция за запис на ребра на граф. А тя е, ребрата да се записват с кръгли скоби, независимо от това, дали върховете са наредени или не[†] [‡]. И така, записвайки дадено ребро като (u, v), ние **нямаме предвид** наредена двойка, а имаме предвид множеството с елементи u и v.

Конвенция 1

Когато става дума за графи, буквата n означава броя на върховете, освен ако не е дефинирана иначе, и буквата m означава броя на ребрата, освен ако не е дефинирана иначе. Типичен запис е $V = \{v_1, v_2, \ldots, v_n\}$ и $E = \{e_1, e_2, \ldots, e_m\}$.

Конвенция 2

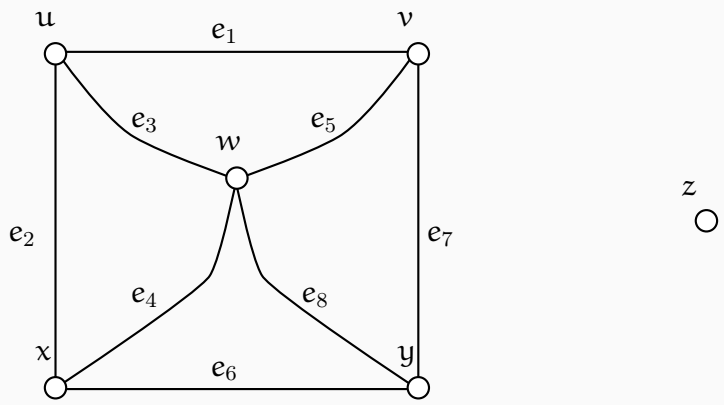
Друга полезна конвенция е, ако е дадено името на графа, да кажем G, но имената на множеството от върховете и на множеството от ребрата са неизвестни, да ги означаваме съответно с "$V(G)$" и "$E(G)$".

Фигура 2.1 показва примерен граф $G_1 = (\{u, v, w, x, y, z\}, \{e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8\})$, където $e_1 = (u, v)$, $e_2 = (u, x)$, $e_3 = (u, w)$, $e_4 = (x, w)$, $e_5 = (v, w)$, $e_6 = (x, y)$, $e_7 = (v, y)$, $e_8 = (w, y)$. Както се вижда, възможно е графът да има върхове, които не са в нито едно ребро, като връх z в случая. Такива върхове се наричат *изолирани*.

[†] Както ще видим в Глава 3, при ориентираните графи, при които има наредба между върховете, ребрата също се записват с кръгли скоби.

[‡] Има още една конвенция: ребрата да се записват само чрез имената на върховете, долепени едно до друго, без никакви скоби, например "$e_1 = uv$". Такава конвенция се ползва например в книгата на Diestel [18]. В тези лекционни записки няма да ползваме тази конвенция.

Фигура 2.1 : Рисунка на графа G_1.



Възможно е ребрата да не бъдат именувани явно и тогава пишем $G_1 = (\{u, v, w, x, y, z\}, \{(u, v), (u, x), (u, w), (x, w), (v, w), (x, y), (v, y), (w, y)\})$.

Наблюдение 1

В Определение 1 настояваме, че $V \neq \varnothing$, а допускаме възможността $E = \varnothing$.
Няма съществена пречка множеството от върховете да е празно, но е по-удобно да допуснем, че V вигаги е непразно. От друга страна, в много ситуации е важно да може да няма нито едно ребро, така че трябва да разрешим E да е празно.

Определение 2: Празен и тривиален граф.

Нека $G = (V, E)$ е граф. Ако $E = \varnothing$, казваме, че G е *празен граф*. Ако $E \neq \varnothing$, G е *непразен*. Ако $|V| = 1$, казваме, че G е *тривиален граф*.

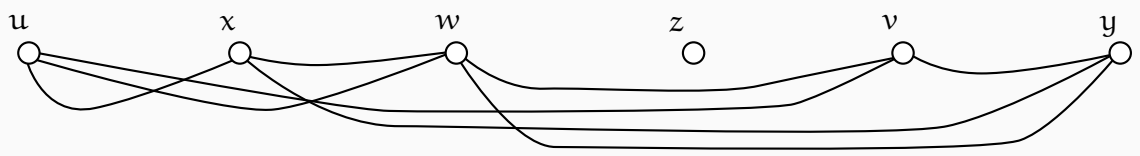
Забележете, че $|V| = 1$ влече $|E| = 0$, така че тривиален граф е частен случай на празен граф.

Много важно е да се разбере, че **рисунка на даден граф е принципно различно нещо от графа.** Аналогично, снимка на човек е нещо различно от човека, който е сниман. Графът е теоретико-множествен обект: наредена двойка от някакви множества. Дадена рисунка на този граф е само средство той да бъда визуализиран така, че да бъде удобно на читателя да си го представи. Един граф може да бъде нарисуван по безброй начини, но всички тези рисунки си остават рисунки на един и същи граф. Дори да формализираме понятието "рисунка" и да говорим за геометричен обект, състоящ се от точки и криви в равнината, този геометричен обект би бил принципно различно нещо от съответния граф – геометричният обект е по-сложен, неговите точки имат някакви координати и неговите криви имат някакви уравнения, докато при графа за координати на точките и за уравнения на кривите не става дума. Можем да кажем, че на един граф съответстват безброй много геометрични обекти в равнината, които са изоморфни[†] помежду си, а графът се явява тяхна **абстракция**. Например, друга

[†]Вижте Секция 2.8 за определение на "изоморфизъм на графи". Можем да дефинираме и изоморфизъм между геометрични графи.

възможна рисунка на графа от Фигура 2.1 е показана на Фигура 2.2[†]. Имената на ребрата са изпуснати на фигурата.

Фигура 2.2 : Друга рисунка на графа G_1 от Фигура 2.1.



2.1.2 Съседство и инцидентност.

Определение 3

Нека $G = (V, E)$ е граф. За всеки два върха u и v, такива че $(u, v) \in E$ казваме, че u *и* v *са съседи*. За всеки две ребра e_1 и e_2, такива че $e_1 \cap e_2 \neq \varnothing$ казваме, че e_1 *и* e_2 *са инцидентни*. За всеки връх u и всяко ребро e, такова че $u \in e$ казваме, че *реброто* e *е инцидентно с връх* u. Ако $e \in E$ и $e = (u, v)$, то казваме, че u *и* v *са краищата на* e.

Като пример да разгледаме графа G_1 на Фигура 2.1. Върховете u и v са съседи, u и y не са съседи, z не е съсед на никой връх, u и e_1 са инцидентни, u и e_7 не са инцидентни, краищата на e_7 са v и y, и така нататък.

Съседството определя релация $R_G \subseteq V \times V$, такава че $\forall u \in V : uR_Gv \leftrightarrow u$ и v са съседи. Ще наричаме тази релация, *релацията на съседство върху* G[‡] . Очевидно, R_G е антирефлексивна[§] (защото никой връх не е съсед на себе си), симетрична и не е транзитивна. Аналогично, инцидентността между ребрата определя релация със същите свойства.

2.1.3 Степени на върхове.

Определение 4: Множества $N(u)$, $N[u]$ и $\mathcal{I}(u)$.

Нека $G = (V, E)$ е граф. За всеки връх $u \in V$, $N(u) \stackrel{\text{def}}{=} \{v \in V \mid u \text{ и } v \text{ са съседи}\}$, $N[u] \stackrel{\text{def}}{=} N(u) \cup \{u\}$ и $\mathcal{I}(u) \stackrel{\text{def}}{=} \{e \in E \mid e \text{ е инцидентно с } u\}$.
Тези определения естествено се пренасят и за множества от върхове: ако $U \subseteq V$, то $N(U) = \bigcup_{x \in U} N(x) \backslash U$, $N[U] = \bigcup_{x \in U} N[x]$ и $\mathcal{I}(U) = \bigcup_{x \in U} \mathcal{I}(x)$.

[†]На Фигура 2.2 някои криви, които представляват ребра, се пресичат. Това не означава, че в точките на пресичане има върхове. Както ще видим в Секция 2.13, не винаги е възможно да нарисуваме граф в равнината по такъв начин, че кривите, които отговарят на ребрата, да не се пресичат във вътрешни точки.

[‡]В някакъв смисъл, R_G и G са едно и също нещо, защото ребрата на графа задават еднозначно въпросната релация и обратното. Незначителен технически детайл е това, че релацията се състои от наредени двойки, а ребрата са ненаредени; освен това, ако има изолирани върхове, те не участват в нито една наредена двойка на релацията и, от нейна гледна точка, тях просто ги няма, докато в графа ги има.

[§]Както ще видим в Секция 2.2, ако позволим в графа да има така наречените *примки*, които са ребра, всяко от които има за два края един и същи връх, релацията не е непременно антирефлексивна, защото тогава може връх да е съсед на себе си.

Като пример да разгледаме графа G_1 на Фигура 2.1. $N(u) = \{v, w, x\}$, $N(w) = \{u, v, x, y\}$, $N(z) = \varnothing$, $N[w] = \{u, v, w, x, y\}$, $N[z] = \{z\}$, $N(\{w, y\}) = \{u, v, x\}$, $N[\{w, y\}] = \{u, v, w, x, y\}$, $\mathcal{I}(y) = \{e_6, e_7, e_8\}$, $\mathcal{I}(z) = \varnothing$, и така нататък.

Определение 5: Степен на връх.

Нека $G = (V, E)$ е граф. За всеки връх $u \in V$, *степента на* u е $|\mathcal{I}(u)|$.

Нотация 1: $d(u)$, $\Delta(G)$ и $\delta(G)$.

Нека $G = (V, E)$ е граф. С $d(u)$ бележим степента на u, за всеки връх $u \in V$. Освен това, $\Delta(G) = \max\{d(u) \mid u \in V\}$ и $\delta(G) = \min\{d(u) \mid u \in V\}$.

Като пример да разгледаме графа G_1 на Фигура 2.1. По отношение на него, $d(u) = d(v) = d(x) = d(y) = 3$, $d(w) = 4$ и $d(z) = 0$. Освен това, $\Delta(G_1) = 4$ и $\delta(G_1) = 0$.

Допълнение 1: Друго възможно дефиниране на "степен на връх"

На пръв поглед, възможно е да дефинираме "степен на връх" и така: степента на u е броят на съседите на u, тоест $d(u) = |N(u)|$. Ако става дума за **обикновени графи**, каквито разглеждаме в момента, това определение е **еквивалентно** на Определение 5. Но, както ще видим в Глава 2.2, Определение 5 се пренася директно върху неориентирани мултиграфи, стига те да нямат примки. Да разгледаме пак мултиграфа на Фигура 1.3, който се появи в историческата справка. Ако приложим Определение 5 директно, ще получим, че връх K има степен 5, защото има 5 ребра, инцидентни с него. От друга страна, ако дефинираме степента на връх като броят на съседите му, би трябвало K да има степен 3. От изложението в Секция 2.2 става ясно, че има смисъл степента на K да е 5, а не 3.

Очевидно е, че изолираните върхове са точно върховете от степен нула. За върховете от степен едно също има термин – те се наричат *висящи* върхове, на английски *pendant vertices*. Графът на Фигура 2.1 няма висящи върхове. Пример за граф, който има висящ връх, е показан на Фигура 2.13 – там w е висящ връх.

Наблюдение 2: Максимална степен на връх.

За всеки граф G, $\Delta(G) \leqslant n - 1$, защото връх може да е съсед най-много на всички останали върхове.

Лема 1

За всеки граф $G = (V, E)$ е изпълнено:

$$\sum_{u \in V} d(u) = 2m$$

Доказателство: Сумата в лявата страна на равенството брои всяко ребро точно два пъти. □

Следствие 1

За всеки граф $G = (V, E)$, броят на върховете от нечетна степен е четно число.

Доказателство: Припомняме си, че $\sum_{u \in V} d(u) = 2m$. Очевидно V се разбива[†] на V_e, върховете от четна степен, и V_o, върховете от нечетна степен. Тогава $\sum_{u \in V} d(u) = \sum_{u \in V_e} d(u) + \sum_{u \in V_o} d(u)$. Очевидно $\sum_{u \in V_e} d(u) + \sum_{u \in V_o} d(u) = 2m$. Тъй като $\sum_{u \in V_e} d(u)$ винаги е четно число, то $\sum_{u \in V_o} d(u)$ също така винаги е четно число, понеже $\sum_{u \in V_e} d(u)$ и $\sum_{u \in V_o} d(u)$ се сумират до четно число, а именно $2m$.

Щом $\sum_{u \in V_o} d(u)$ е четно число, задължително е вярно, че $|V_o|$ е четно число. □

Определение 6: Редица от степените.

Нека $G = (V, E)$ е граф. *Редицата от степените на* G е редицата от числата-степени на върхове в G, подредени в ненамаляващ ред.

Например, редицата от степените на графа G_1 от Фигура 2.1 е $0, 3, 3, 3, 3, 4$.

Следната лема е известна като "the hand-shaking lemma". Неформално, тя казва, че в произволна група хора, които са се здрависали по произволен начин помежду си, има поне двама души, които са се здрависали един и същи брой пъти.

Лема 2: The hand-shaking lemma.

Нека $G = (V, E)$ е граф с поне два върха. Съществуват поне два различни върха $u, v \in V$, такива че $d(u) = d(v)$.

Доказателство: Твърдението е еквивалентно на твърдението, че в редицата от степените има поне едно повтаряне на елементи. От Наблюдение 2 заключаваме, че за всеки връх u, $d(u) \in \{0, 1, \ldots, n-1\}$. Това са n различни стойности. Със следните съображения ограничаваме възможните стойности до само $n - 1$.

Да допуснем, че графът има поне един изолиран връх. Това означава, че редицата от степените започва с 0. Лесно се вижда, че тогава няма връх от степен $n-1$: ако има връх от степен $n-1$, той трябва да е съсед на всички останали върхове, включително и на върха от степен 0, което противоречи на факта, че връх от степен 0 няма съседи. Показахме, че няма връх от степен $n-1$. Тогава, за всеки връх u, $d(u) \in \{0, 1, \ldots, n-2\}$. Но това са само $n-1$ различни стойности. А върховете са n. Съгласно принципа на Дирихле, трябва да има поне два върха u и v, такива че $d(u) = d(v)$.

Сега да допуснем, че няма изолирани върхове. Тогава за всеки връх u, $d(u) \in \{1, 2, \ldots, n-1\}$. Но това са само $n-1$ различни стойности. А върховете са n. Съгласно принципа на Дирихле, трябва да има поне два върха u и v, такива че $d(u) = d(v)$. □

[†]Незначителна подробност е, че това "разбиване" може да не е разбиване съгласно формалната дефиниция, защото всяко от V_e и V_o може да е празно. Това е без значение за валидността на доказателството.

Допълнение 2: Задачата със здрависванията

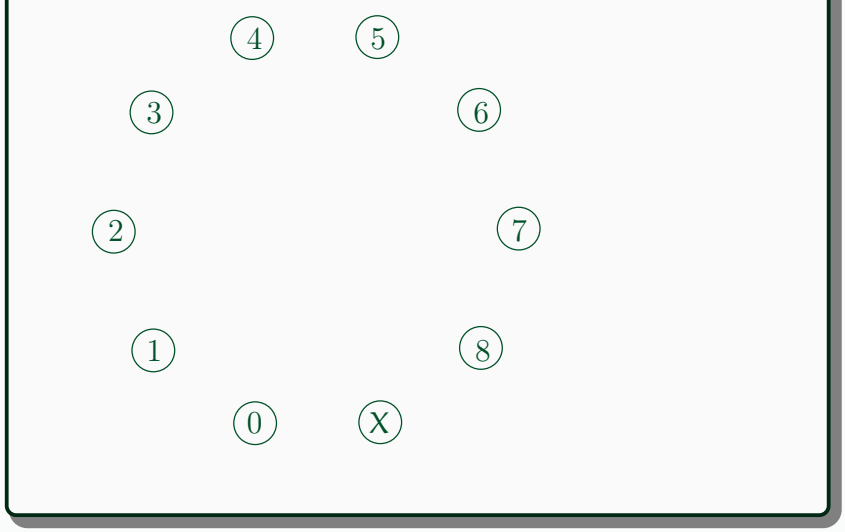
Задача 1

Семейство Стефанови е поканило четири семейства на гости. След пристигането си, някои от гостите се здрависват с други гости, някои, не. Никой не се здрависва със своя съпруг или съпруга, не се здрависва повече от веднъж с един и същи човек, и естествено не се здрависва със себе си. Когато всички гости са дошли и са настанени, г-н Стефанов пита всеки от останалите хора (това включва и неговата съпруга), с колко души се е здрависала или здрависал. Той получава само различни отговори. В колко здрависвания е участвал самият г-н Стефанов?

Тази задача е от книгата на Michalewicz и Fogel [43, стр. 26], като е предложена на авторите от Peter Ross от университета на Edinburgh.

Решение: Става дума за пет семейства общо, което означава 10 човека. Ще моделираме задачата с граф. Върховете са десетте човека, а ребро между два върха се поставя тогава и само тогава, когато съответните хора са се здрависали. Казано е, че г-н Стефанов е получил само различни отговори. Не е казано експлицитно, но се подразбира, че г-н Стефанов е питал (и получил отговори от) 9 човека. Очевидно всеки човек може да участва в най-малко 0 и най-много 8 ръкостискания, понеже не може да се здрависва със себе си и със съпругата си или съпруга си. Това са 9 различни стойности. Тъй като г-н Стефанов е чул 9 различни числа от 9 човека, то той е чул именно числата $0, 1, \ldots,$ 8. Забележете, че редицата $(0, 1, \ldots, 8)$ не е редицата от степените на графа, защото това са само степените на върховете на другите хора—не знаем самият г-н Стефанов в колко здрависвания е участвал, затова и степента на неговия връх засега не участва. Да означим върха, съответстващ на г-н Стефанов, с X, а върховете, съответстващи на останалите хора, със съответните числа, които те са казали на г-н Стефанов. Да си представим само върховете на графа без ребрата (вж. Фигура 2.3).

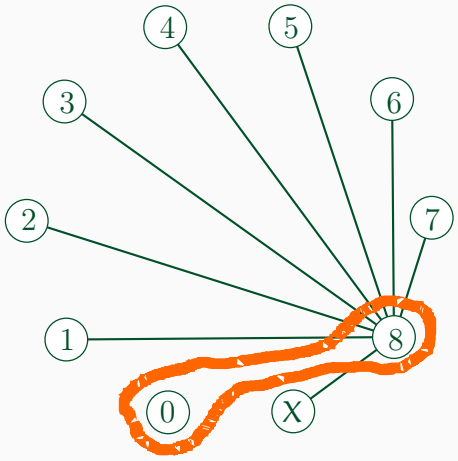
Фигура 2.3 : Графът на хората: само върховете.



Да разсъждаваме така. Връх 8 не може да е съсед на връх 0, следователно връх 8 трябва да е съсед на върхове 1, ..., 7, както и на връх X; ако не е съсед на тези

върхове, няма как да е от степен осем. Следователно, ребрата, показани на Фигура 2.4, са в графа. Нещо повече—заключаваме, че хората, съответстващи на върхове 8 и 0, са съпружеска двойка, понеже няма кой друг да е съпруг или съпруга на 8, освен 0 (помним, че съпружеските двойки не се ръкуват помежду си, а 8 се е ръкувал или ръкувала с всички останали).

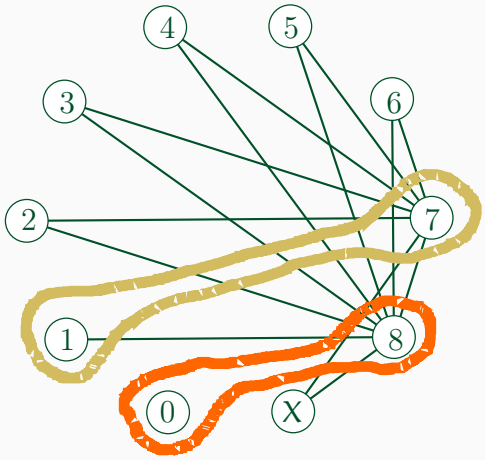
Фигура 2.4 : Тези ребра със сигурност са в графа.



Върхове 0 и 8 отговарят на семейна двойка.

Аналогично, връх 7 не може да е съсед на връх 0, защото връх 0 няма съседи, и не може да е съсед на връх 1, защото единственият съсед на връх е връх 8. Следователно, връх 7 трябва да е съсед—освен на 8—на върхове 2, . . . , 6, както и на връх X; ако не е съсед на тези върхове, няма как да е от степен седем. Следователно, ребрата, показани на Фигура 2.5, също са в графа. Нещо повече—хората, отговарящи на върхове 1 и 7, също са семейна двойка, защото няма кой друг да е съпруг или съпруга на 7, освен 1 (вече установихме, че 0 е в съпружеска двойка с 8).

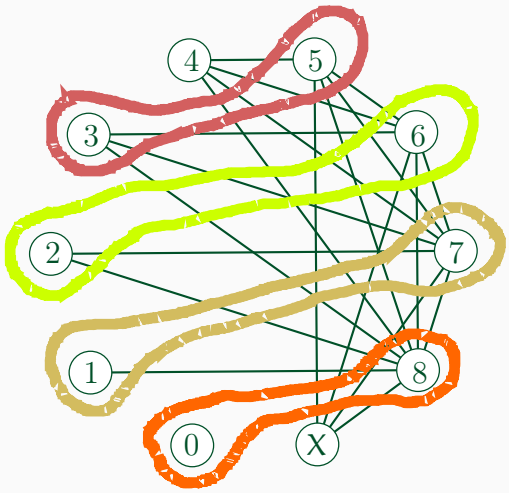
Фигура 2.5 : Тези ребра също със сигурност са в графа.



Върхове 1 и 7 отговарят на друга семейна двойка.

Ако продължим да разсъждаваме така, ще заключим, че 6 е съсед—освен на 8 и 7—на върхове 3, ..., 6, както и на връх X; и че 6 и 2 са съпружеска двойка. Също така, 5 е съсед—освен на 8, 7 и 6—на върхове 4, ..., 6, както и на връх X; и че 5 и 3 са съпружеска двойка. От разсъжденията дотук заключаваме, че графът съдържа поне ребрата, показани на Фигура 2.6.

Фигура 2.6 : Графът на хората. Оказва се, че това са всички ребра.



Разсъждавайки за върхове 8, 7, 6 и 5, заключаваме, че изобразените ребра са в графа. Четирите семейните двойки, които сме установили, са оградени.

Но графът не може да съдържа други ребра—за всеки връх u освен X вече установихме точно кои са ребрата, инцидентни с u, така че е невъзможно да има други ребра, защото освен X те трябва да имат още един край. И така, графът на ръкостисканията е точно

графът, показан на Фигура 2.6. Петата семейна двойка може да бъде единствено X и връх 4.
И така, г-н Стефанов се е здрависал точно 4 пъти, и неговата съпруга е четворката. □

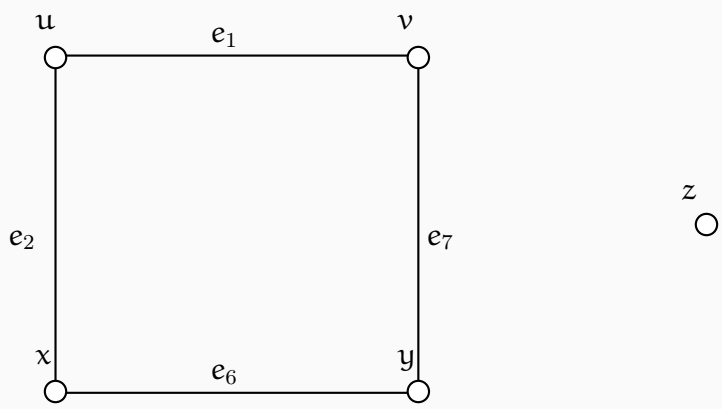
2.1.4 Изтриване на върхове и ребра.

Нека $G = (V, E)$ е граф с поне два върха. Нека u е връх в G. Да *изтрием* u *от* G означава да преобразуваме G в G', където:

$$G' = (V \backslash \{u\}, E \backslash \mathcal{I}(u))$$

Накратко пишем $G' = G - u$[†]. Като пример: ако изтрием връх w от графа G_1 на Фигура 2.1, ще получим графа на Фигура 2.7.

Фигура 2.7 : Графът $G_1 - w$, където G_1 е графът от Фигура 2.1.



Изтриването на върхове в някакъв смисъл е асоциативно: ако ще трием няколко върха, в каквото и ред да го сторим, резултатът ще е един и същи. И така, ако изтриваме върховете $u_1, u_2, \ldots, u_k$, записваме:

$$G - u_1 - u_2 - \cdots - u_k$$

Ако $\{u_1, u_2, \ldots, u_k\} = U$, може да запишем накратко

$$G - U$$

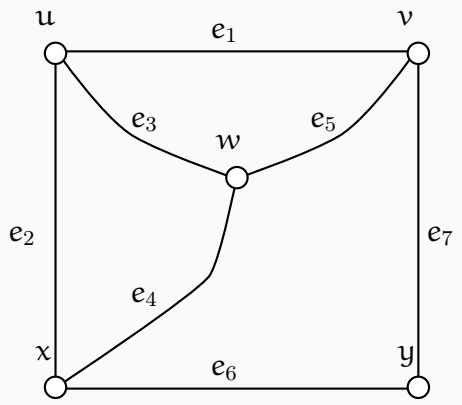
Нека $G = (V, E)$ е граф. Нека $e = (x, y)$ е ребро в него. Да *изтрием* e *от* G означава да преобразуваме G в

$$G' = (V, E \backslash \{e\})$$

Накратко пишем $G' = G - e$. Забележете, че при изтриване на ребро, неговите краища си остават в графа. Като пример: ако изтрием реброто e_8 от графа G_1 на Фигура 2.1, ще получим графа на Фигура 2.8.

[†]Подчертаваме, че ако u "изчезне" от графа, то задължително ребрата, индидентни с u, също трябва да "изчезнат". Недопустимо е да остане ребро, единият край на който не е "стъпил" върху връх и "виси".

Фигура 2.8 : Графът $G_1 - e_8$, където G_1 е графът от Фигура 2.1.



Изтриването на ребра също е асоциативно в някакъв смисъл: ако ще трием няколко ребра, в какъвто и ред да го сторим, резултатът ще е един и същи. И така, ако изтриваме ребрата e_1, e_2, $\ldots$, e_k, записваме:

$$G - e_1 - e_2 - \cdots - e_k$$

Ако $\{e_1, e_2, \ldots, e_k\} = F$, може да запишем накратко

$$G - F$$

2.1.5 Обединение и сечение на графи

Нека $G_1 = (V_1, E_1)$ и $G_2 = (V_2, E_2)$ са графи. *Обединението на* G_1 *и* G_2 е графът $G_1 \cup G_2 = (V_1 \cup V_2, E_1 \cup E_2)$. *Сечението на* G_1 *и* G_2 е графът $G_1 \cap G_2 = (V_1 \cap V_2, E_1 \cap E_2)$. Има смисъл да дефинираме сечението само ако $V_1 \cap V_2 \neq \varnothing$. От друга страна, обединението е дефинирано дори когато $V_1 \cap V_2 \neq \varnothing$.

Обединение и сечение на графи се обобщават по очевидния начин за повече от два графа, като нотацията е съответно $G = G_1 \cup \cdots \cup G_k$ и $G = G_1 \cap \cdots \cap G_k$.

2.1.6 Регулярни графи. Пълни и празни графи.

Оттук до края на Секция 2.1 правим определенията спрямо някакъв граф $G = (V, E)$.

G е k-*регулярен*, ако $\forall u \in V : d(u) = k$. G е *регулярен*, ако е k-регулярен за някое k.

Не особено формално казано, G е *пълен граф*, ако има всички възможни ребра при даденото множество върхове. Формално, G е пълен, ако $E(G) = \{(u, v) \,|\, u \in V, v \in V, u \neq v\}$.

Пълен граф на n върха се бележи с K_n. Ако игнорираме идентичностите на върховете, има само един пълен граф на n върха и има смисъл да говорим за **пълния** граф на n върха. На практика обикновено се говори именно за "пълния граф на n върха", а не за "пълен граф на n върха" по точно тази причина – игнорирали сме идентичностите на върховете. Очевидно K_n е $(n-1)$-регулярен. Подробно разискване на игнорирането на идентичностите на върховете има в Подсекция 2.8.2.

Наблюдение 3

K_n има точно $\binom{n}{2}$ ребра. □

Да си припомним Определение 2 на стр. 7. Всеки празен граф очевидно е 0-регулярен и всеки тривиален граф също е 0-регулярен.

2.1.7 Подграфи. Индуцирани подграфи. Покриващи подграфи.

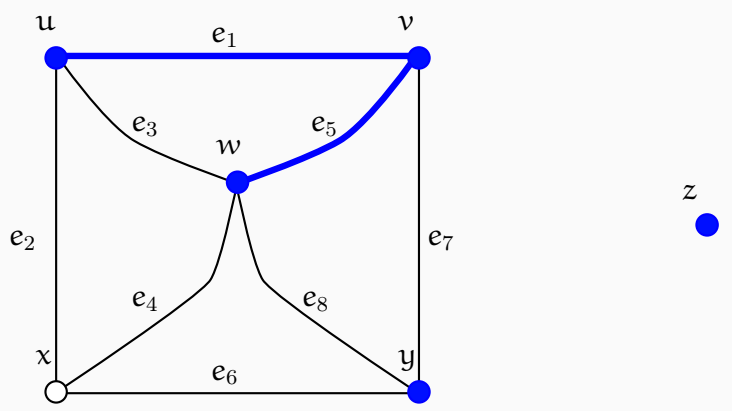
Определение 7: Подграф

Нека $G = (V, E)$ е граф. *Подграф на* G е всеки граф $G' = (V', E')$, такъв че $V' \subseteq V$ и $E' \subseteq E$.

Подчертаваме, че за да бъде G' подграф на G, **не е достатъчно** да бъде изпълнено $V' \subseteq V$ и $E' \subseteq E$. Всяко ребро в E' трябва да има за краища върхове, които са във V'. **Това** прави G' граф.

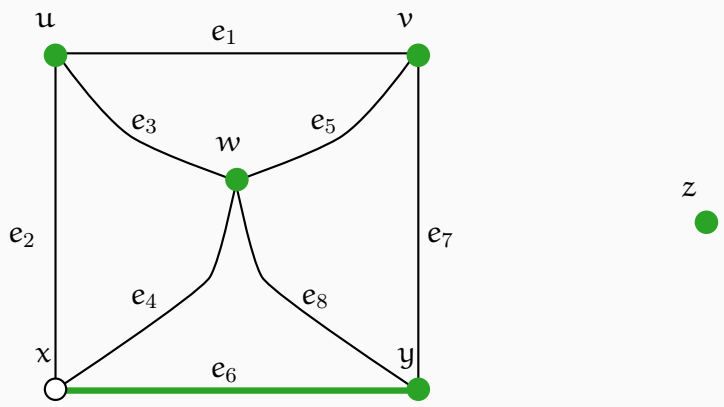
Като пример, да разгледаме графа G_1 на Фигура 2.1. Нека $H' = (\{u, v, w, y, z\}, \{(e_1, e_5)\})$. Тогава H' е подграф на G_1—вижте Фигура 2.9.

Фигура 2.9 : $H' = (\{u, v, w, y, z\}, \{e_1, e_5\})$ (в синьо) е подграф на G_1 от Фиг. 2.1.



Нека сега $H'' = (\{u, v, w, y, z\}, \{e_6\})$. Забелязваме, че H'' не е подграф на G_1 от Фигура 2.1, понеже единият край на реброто e_6, а именно връх x, не е в множеството $\{u, v, w, y, z\}$; с други думи, H'' изобщо не е граф. Да подчертаем отново: **в граф не може да има "хвърчащи" ребра, които не са "стъпили" с двата си края върху върхове от този граф**. Фигура 2.10 илюстрира не-графа H''.

Фигура 2.10 : $H'' = (\{u, v, w, y, z\}, \{e_6\})$ (в зелено) не е подграф на G_1 от Фиг. 2.1. H'' щеше да е подграф, ако и x беше негов връх.



Определение 8: Индуциран от подмножество върхове подграф

Нека $G = (V, E)$ е граф, $U \subseteq V$ и $U \neq \varnothing$. *Подграфът на* G*, индуциран от* U, е графът $G' = (U, E')$, където $E' = \{(u, v) \in E \mid u \in U \wedge v \in U\}$.

С други думи, при дадено подмножество от върхове U, индуцираният подграф е графът, чието множество върхове е U и чието множество ребра са точно тези ребра на G, на които и двата края са в U. "Индуциран" означава "доведен"[†] и, наистина, индуцираният подграф е "доведен" от избора на U, в смисъл че ребрата на индуцирания подграф са еднозначно определени от върховете на U.

Можем да дефинираме и подграф, индуциран от подмножество на ребрата.

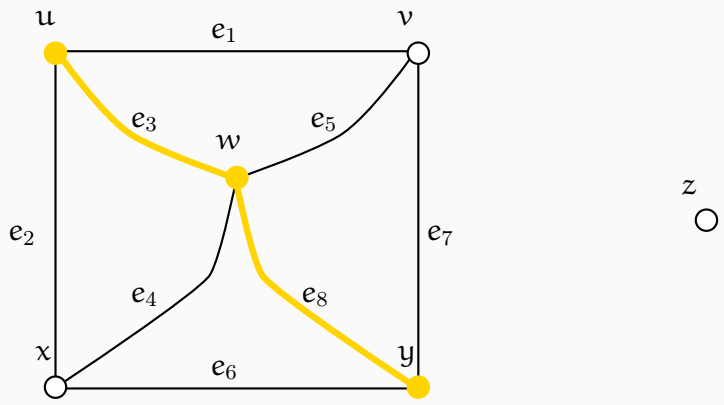
Определение 9: Индуциран от подмножество ребра подграф

Нека $G = (V, E)$ е граф и $E' \subseteq E$. *Подграфът на* G*, индуциран от* E', е графът $G' = (U, E')$, където $U = \{u \in V \mid \exists e \in E' \, \exists x \in V : e = (u, x)\}$.

Нека пак разгледаме графа G_1 от Фигура 2.1. Нека изберем едно подмножество от върхове, да кажем $\{u, w, y\}$. Подграфът на G_1, индуциран от $\{u, w, y\}$, е $(\{u, w, y\}, \{e_3, e_8\})$. Фигура 2.11 илюстрира този индуциран подграф.

[†]На латински "duco" означава "водя".

Фигура 2.11 : Графът G_1 от Фигура 2.1 и неговият подграф, индуциран от $\{u, w, y\}$, оцветен в оранжево.



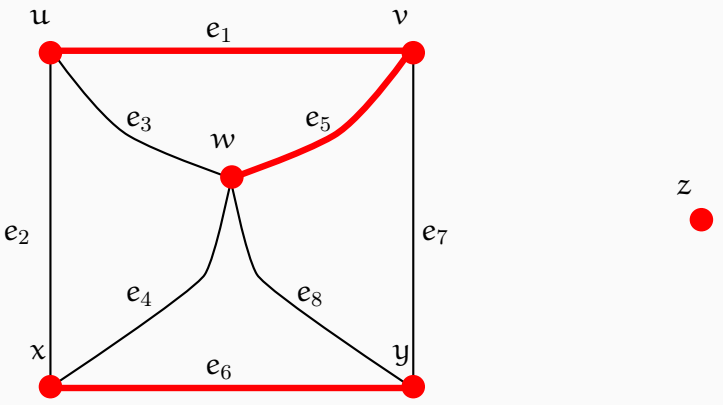
Определение 10: Покриващ подграф

Нека $G = (V, E)$ е граф. *Покриващ подграф на* G е всеки подграф $G' = (V', E')$ на G, такъв че $V' = V$.

На английски терминът е *spanning subgraph*.

В примерите за подграфи, които видяхме дотук, няма покриващ подграф. Покриващ подграф е показан на Фигура 2.12. Подграфът $J = (\{u, v, w, x, y, z\}, \{e_1, e_5, e_6\})$, нарисуван в червено, е покриващ за графа от Фигура 2.1.

Фигура 2.12 : Покриващ подграф (в червено) на графа от Фиг. 2.1.



Очевидно празният граф $(V, \varnothing)$ е покриващ граф за $G = (V, E)$, както и самият G е покриващ граф на себе си. Нещо повече, броят на покриващите графи е точно 2^m, защото множеството от върхове е фиксирано, а всяко ребро от E може да присъства, или не, независимо от другите.

2.1.8 Клики и антиклики.

Всяко $U \subseteq V$, такова че $U \neq \varnothing$ и $|U| = k$, е k-*клика* в G, ако между всеки два върха на U има ребро. U е k-*антиклика*, ако между никои два върха на U няма ребро. Ако кажем само *клика* или *антиклика*, става дума за k-клика или k-антиклика за някое k. Използваме понятието *независимо множество* като синоним на антиклика.

Като алгоритмична задача, задачата за кликите е максимизационна: трудно е да се намерят големи клики. Намирането на малки клики е тривиално—например, всеки връх е 1-клика сам по себе си—но безинтересно. Аналогично, задачата за антикликите също е максимизационна—ако графът не е пълен, лесно може да намерим двуелементна антиклика.

Като пример, да разгледаме графа на Фигура 2.1. Множеството върхове $\{u, v, w\}$ е 3-клика. Също така, $\{x, w, y\}$ е 3-клика. Краищата на всяко ребро са 2-клика, например $\{v, y\}$. Всеки връх сам по себе си е тривиална 1-клика, например $\{x\}$ или $\{z\}$. В този граф 4-клика няма. Примери за антиклики са: $\{z, v\}$ е 2-антиклика, $\{u, y, z\}$ е 3-антиклика и така нататък. 4-антиклика в графа няма.

Определение 11: Кликово число

Кликово число на G е мощността на максимална[а] клика в G. То се бележи с $\omega(G)$. *Числото на независимост на* G е мощността на максимална антиклика[б] в G. То се бележи с $\alpha(G)$.

[а] Не максимална по включване, а максимална по мощност.
[б] Да си припомним, че "независимо множество" е синоним за "антиклика".

Например, $\omega(G_1) = 3$ и $\alpha(G_1) = 3$, където G_1 е графът от Фигура 2.1.

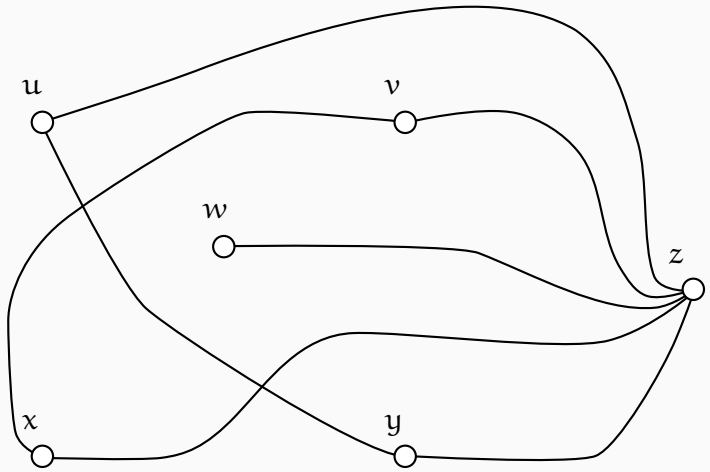
2.1.9 Допълнение на граф.

Определение 12: Допълнение на граф

Допълнението на G, което бележим с $\overline{G}$, е графът $\overline{G} = (V, E')$, където $E' = \{(u, v) \mid u \in V, v \in V, u \neq v\} \backslash E$.

Неформално казано, E' се състои от точно тези ребра, които "ги няма" в G. Да разгледаме графа от Фигура 2.1. Неговият граф-допълнение е изобразен на Фигура 2.13.

Фигура 2.13 : Графът-допълнение на графа G_1 от Фигура 2.1.



Наблюдение 4

$\overline{\overline{G}} = G$, за всеки граф G.

Наблюдение 5

Нека $G = (V, E)$ е граф. За всяко $U \subseteq V$ е изпълнено следното: U е клика в G тогава и само тогава, когато U е антиклика в $\overline{G}$.

Следствие 2

$\omega(G) = \alpha(\overline{G})$.

В Теорема 1 имаме предвид включващо **или**.

Теорема 1: $n \geqslant 6$ влече 3-клика или 3-антиклика

Нека $G = (V, E)$ е граф с поне 6 върха. В G има 3-клика или 3-антиклика.

Доказателство: Без ограничение на общността, ще докажем твърдението за $n = 6$. Нека E' е множеството от ребрата на $\overline{G}$. Конструираме графа $\widetilde{G} = (V, \widetilde{E})$, където $\widetilde{E} = E \cup E'$. Очевидно $\widetilde{G}$ е пълен граф на 6 върха, така че $|\widetilde{E}| = \binom{6}{2} = 15$. Да дефинираме, че *червените ребра*[†] в $\widetilde{G}$ са ребрата от E, а *сините ребра* са ребрата от E'. Тогава $\widetilde{E}$ се разбива на множеството от червените ребра и множеството от сините ребра. Нещо повече. Твърдението, което искаме да докажем, е еквивалентно на твърдението, че в $\widetilde{G}$ има 3-клика, която индуцира подграф само с червени ребра или има 3-клика, която индуцира подграф само със сини ребра.

Да разгледаме произволен връх $u \in V$. В $\widetilde{G}$, този връх е инцидентен с точно 5 ребра. Всяко от тях е или червено, или синьо. Съгласно обобщения принцип на Дирихле, поне 3 от

[†]Това слагане на цветове на ребрата е много различно от оцветяването на ребра съгласно Определение 37 на стр. 69. В Определение 37 се иска всички двойки инцидентни ребра да са в различни цветове. Тук няма такова изискване.

тях са в един и същи цвят. Без ограничение на общността, нека този цвят е червен. И така, три ребра, инцидентни с u, да ги наречем (u, x), (u, y), (u, z), са червени.

Сега разглеждаме ребрата (x, y), (x, z) и (y, z). Да допуснем, че поне едно от тях е червено. Без ограничение на общността, нека това е (x, y). Тогава в $\widetilde{G}$ има 3-клика, а именно $\{u, x, y\}$, такава че индуцираният от нея подграф има само червени ребра, а именно (u, x), (u, y) и (x, y), които са червени съгласно текущите допускания.

Сега да допуснем, че нито едно от ребрата (x, y), (x, z) и (y, z) не е червено. Тогава и трите са сини. Тогава в $\widetilde{G}$ има 3-клика, а именно $\{x, y, z\}$, такава че индуцираният от нея подграф има само сини ребра, а именно (x, y), (x, z) и (y, z), които са сини съгласно текущите допускания. □

2.2 Неориентирани мултиграфи.

2.2.1 Определение. Паралелни ребра. Основен граф. Примки.

Неформално казано, "неориентиран мултиграф" е обобщение на "неориентиран граф", в което позволяваме да има много ребра с краища едни и същи два върха. Това усложнява дефиницията на понятието – вече не можем да идентифицираме ребро с двата върха, които са негови краища. Сега всяко ребро трябва да има собствена идентичност, а множеството от краищата на дадено ребро е просто един от неговите атрибути. Казвайки само "мултиграф", имаме предвид неориентиран мултиграф. Ориентирани мултиграфи ще разгледаме в Глава 3.

Определение 13: Мултиграф

Мултиграф е наредена тройка $G = (V, E, f_G)$, където V е непразно множество, чиито елементи се наричат *върхове*, E е множество, чиито елементи се наричат *ребра*, $V \cap E = \varnothing$ и

$$f_G : E \to \{X \subseteq V : |X| = 2\}$$

е *свързващата функция*.

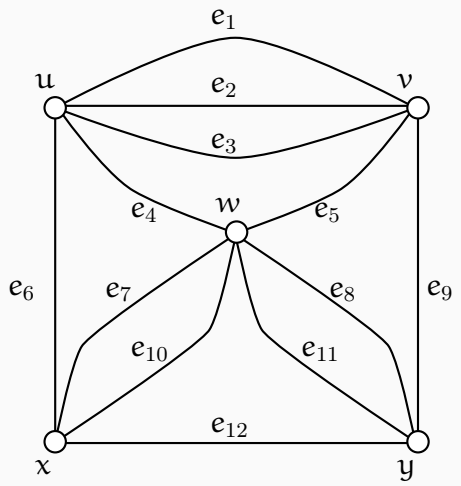
Забележете разликата между Определение 13 и Определение 1. Според Определение 1 върховете са протоелементи (атоми) и множеството от върховете е опорното множество, а ребрата са двуелементни подмножества на опорното множество. В Определение 13 ребрата се третират като друг вид протоелементи, така че множеството от ребрата се явява като второ опорно множество (без общи елементи с първото). Свързващата функция определя връзката между двете опорни множества.

Също като при обикновените графи, върховете по правило се записват с малки латински букви като u, v и т. н., имената на ребрата по правило се записват като e_1, e_2 и т. н., но сега пишем $f_G(e_1) = \{u, v\}$, в случай, че u и v са краищата на реброто e_1. Записи като $e_1 = \{u, v\}$ или $e_1 = (u, v)$ в контекста на мултиграфи **не са разрешени**, понеже в този контекст не идентифицираме ребро с неговите краища.

Също като при обикновените графи, буквата n означава броя на върховете, освен ако не е дефинирана иначе, и буквата m означава броя на ребрата, освен ако не е дефинирана иначе. Но за разлика от обикновените графи, при които $\binom{n}{2}$ е точна горна граница за броя на ребрата, така че неравенството $m \leqslant \binom{n}{2}$ винаги е в сила, сега броят на ребрата може да е **произволно** по-голям от броя на върховете. Иначе казано, ако знаем броя на върховете на мултиграф, не можем да кажем нищо за броя на неговите ребра.

Фигура 2.14 показва пример за мултиграф.

Фигура 2.14 : Пример за неориентиран мултиграф G.

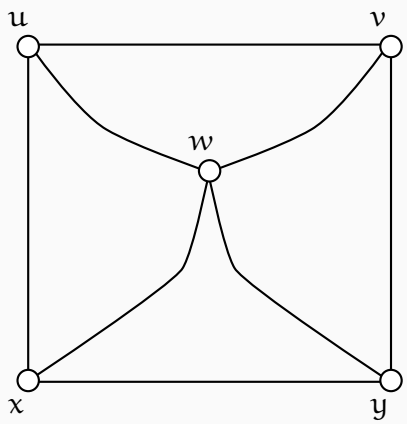


Ако трябва да запишем мултиграфа G от Фигура 2.14 съгласно формалната дефиниция, то $G = (\{u, v, w, x, y\}, \{e_1, \ldots, e_{12}\}, f_G)$, където $f_G(e_1) = f_G(e_2) = f_G(e_3) = \{u, v\}$, $f_G(e_4) = \{u, w\}$, $f_G(e_5) = \{v, w\}$, $f_G(e_6) = \{u, x\}$, $f_G(e_7) = f_G(e_{10}) = \{w, x\}$, $f_G(e_8) = f_G(e_{11}) = \{w, y\}$, $f_G(e_9) = \{v, y\}$, $f_G(e_{12}) = \{x, y\}$.

За всяко $E' \subseteq E$, такова че $\forall e', e'' \in E : f_G(e') = f_G(e'')$, казваме, че ребрата в E' са *паралелни ребра*. Всяко максимално по включване множество от паралелни ребра ще наречем *сноп*. Очевидно множеството от ребрата се разбива на сноповете. В примера на Фигура 2.14, сноповете са $\{e_1, e_2, e_3\}$, $\{e_7, e_{10}\}$, $\{e_8, e_{11}\}$, $\{e_4\}$, $\{e_5\}$, $\{e_6\}$, $\{e_9\}$ и $\{e_{12}\}$. Неформално говорейки, мултиграфът е "истински мултиграф", ако има поне един сноп с повече от едно ребро; в противен случай е обикновен граф. Говорейки формално, мултиграфът е обикновен граф тогава и само тогава, когато свързващата функция е инекция. *Основният граф*[†] на даден мултиграф G е графът, който се получава от заменянето на всеки сноп с едно единствено ребро (така че след замените няма паралелни ребра). Фигура 2.15 показва основния граф на мултиграфа от Фигура 2.14. Имената на ребрата не са показани.

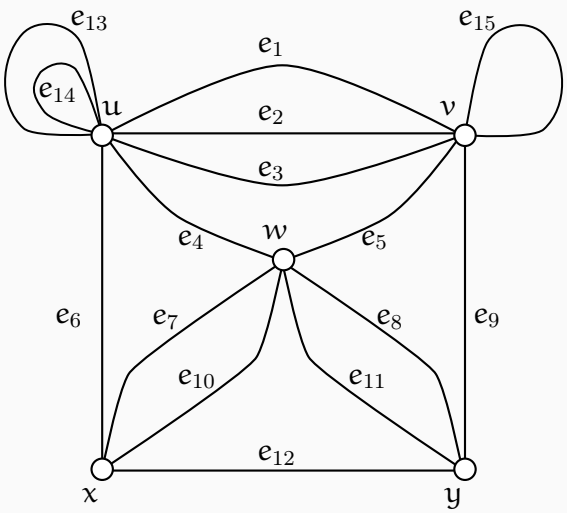
[†]На английски терминът е *the underlying graph of a multigraph*.

Фигура 2.15 : Основният граф на мултиграфа от Фигура 2.14.



Може да разширим разбирането за мултиграф, като позволим наличието на *примки*. Примка е ребро, краищата на което съвпадат. Името очевидно идва оттам, че рисуваме такива ребра като примки. Фигура 2.16 изобразява мултиграфа от Фигура 2.14, към който са добавени три примки: две примки e_{13} и e_{14} към връх u и една примка e_{15} към връх v.

Фигура 2.16 : Мултиграф с примки H.



Формалното определение е почти същото като Определение 13 с единствената разлика, че кодомейнът на свързващата функция включва и едноелементните подмножества на множеството от върховете.

Определение 14: Мултиграф с възможни примки

Мултиграф с възможни примки е наредена тройка $G = (V, E, f_G)$, където V е непразно множество, чиито елементи се наричат *върхове*, E е множество, чиито елементи се наричат *ребра*, $V \cap E = \varnothing$ и

$$f_G : E \to \{X \subseteq V : |X| = 2 \vee |X| = 1\}$$

е *свързващата функция*.

В примера с H от Фигура 2.16, $f_H(e_{13}) = f_H(e_{14}) = \{u\}$, $f_H(e_{15}) = \{v\}$, $f_H(e_1) = f_H(e_2) = f_H(e_3) = \{u, v\}$, и така нататък. По аналогия с казаното преди, примките към един и същи връх са паралелни ребра, така че $\{e_{13}, e_{14}\}$ сноп, а $\{e_{15}\}$ е друг сноп. Основният граф на мултиграф с възможни примки H е същият като основния граф на мултиграфа G, който се получава от G чрез изтриване на примките. И така, основният граф на мултиграфа с примки от Фигура 2.16 е пак графът от Фигура 2.15.

За удобство приемаме, че "мултиграф" означава "мултиграф с възможни примки", така че, ако искаме да забраним примки в разглеждания мултиграф, трябва да кажем това експлицитно.

2.2.2 Пренасяне на определения от графи върху мултиграфи

Някои от определенията и резултатите за обикновени графи може да бъдат пренесени върху мултиграфите, други – не. Някои графови задачи, които ще разгледаме нататък, нямат смисъл върху мултиграфи, тъй като наличието на паралелни ребра не променя нищо; като пример, задачата за намиране на Хамилтонов цикъл (вж. Секция 2.9) е практически същата върху мултиграф и неговия основен граф. От друга страна, задачата за намиране на Ойлеров цикъл (вж. Секция 2.10) се дефинира именно върху мултиграфи, тъй като при тях добавянето на ново ребро, паралелно на едно или повече вече налични ребра, може да промени това дали Ойлеров цикъл съществува, или не съществува.

Съседство и инцидентност при мултиграфите се дефинират по същия начин, както при обикновените графи, с малката разлика, че сега връх може да е съсед на себе си, ако има примки. Всяка примка е инцидентна с върха, към който е "вързана".

Степен на връх в мултиграф се дефинира по начин, подобен на Определение 5, с тази разлика, че, ако има примки към този връх, всяка примка се брои **два пъти**. Примерно, степента на връх u от мултиграфа на Фигура 2.16 е 9, понеже петте ребра e_1, e_2, e_3, e_4 и e_6 се броят по един път, а двете примки e_{13} и e_{14} се броят по два пъти.

Определение 15: Степен на връх в мултиграф с възможни примки.

Нека $G = (V, E, f_G)$ е мултиграф с примки. За всеки връх $u \in V$, *степента на* u е сумата от броя на ребрата, инцидентни с u, които не са примки, и два пъти броя на примките на u.

Лема 3 е тривиално обобщение на Лема 1.

Лема 3

За всеки мултиграф с възможни примки G е изпълнено:

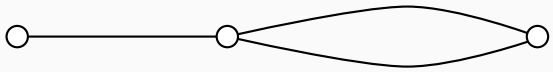
$$\sum_{u \in V(G)} d(u) = 2m$$

Доказателство: Сумата в лявата страна на равенството брои всяко ребро—примка или не—точно два пъти. □

При мултиграфите не говорим за "пълен мултиграф", защото потенциално може да има колкото искаме много ребра с краища дадени два върха. Празен мултиграф се дефинира точно както се дефинира празен граф. По тези причини, лесно можем да въведем понятието "антиклика" при мултиграфите, но не можем да въведем "клика".

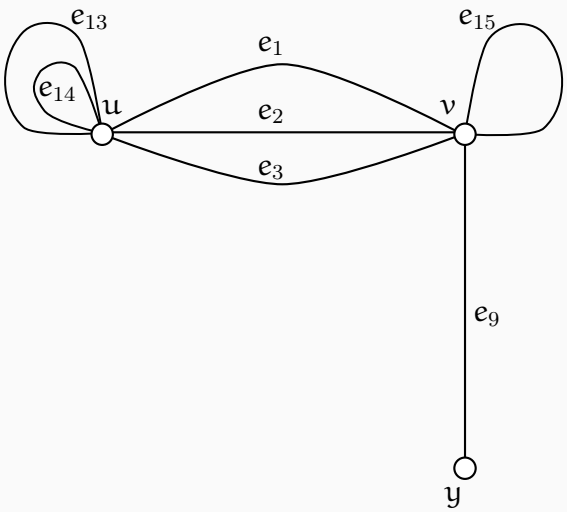
Изолирани и висящи върхове в мултиграф се дефинират точно както при графите. Наблюдение 2 обаче вече не е в сила, понеже максималната степен на връх не е функция на n. Лема 1 остава в сила и за мултиграфи, тъй като сумата вляво продължава да брои всяко ребро два пъти, независимо от това дали става дума за примка, или не. Следствие 1 остава в сила също. Редица от степени на мултиграф се дефинира също както в Определение 6. Примерно, редицата от степени на мултиграфа от Фигура 2.16 е $4, 4, 6, 7, 9$. Лема 2 обаче не е в сила за мултиграфи: тривиално е да се конструира мултиграф, в който няма върхове с една и съща степен, какъвто е мултиграфът на Фигура 2.17.

Фигура 2.17 : Мултиграф с редица от степените $1, 2, 3$.



Определенията "подграф" и "индуциран подграф" се обобщават за мултиграфи по естествения начин. Не е прието да се казва "подмултиграф", така че се въздържаме от използването на тази дума и говорим за "подграфи" дори в контекста на мултиграфи. Ако говорим за индуциран подграф на мултиграф, отново имаме предвид индуциран от множество върхове подграф; в подграфа "влизат" точно тези ребра, краищата на които са във въпросното множество. Това означава, че всяка примка, чийто връх е в във въпросното множество, трябва да се намира и в индуцирания подграф. Примерно, Фигура 2.18 показва индуцирания от $\{u, v, y\}$ подграф на мултиграфа от Фигура 2.16.

Фигура 2.18 : Подграфът, индуциран от $\{u, v, y\}$.



Допълнение на мултиграф няма, тъй *a priori* не е ясно кой би бил универсумът, спрямо който да бъде това допълнение; да си припомним, че пълен мултиграф върху дадено множество върхове не е дефиниран[†].

Пътища и цикли при мултиграфи се дефинират по същия начин, както при обикновени графи (вж. Секция 2.3). Прости пътища и прости цикли също се дефинират по същия начин, като обаче има една особеност.

Наблюдение 6

В мултиграф може да има прост цикъл с дължина 1 или 2, което е невъзможно при обикновените графи. Прост цикъл с дължина 0 обаче не може да има – никой отделен връх не е прост цикъл.

Наистина, всяка примка задава прост цикъл с дължина 1, а всяка двойка паралелни ребра задава прост цикъл с дължина 2. Примерно, на Фигура 2.16, u, e_{13}, u е прост цикъл с дължина 1, а u, e_1, v, e_2, u е прост цикъл с дължина 2. Наблюдение 8 остава в сила и при прости цикли с дължина 2—всеки от тях има 4 различни описания—но не и при простите цикли с дължина 1 (примките), всяка от които има само 1 описание.

Има и друга особеност. Конвенция 3 не е в сила при мултиграфите в общия случай. В общия случай, редицата от имената на върховете на пътя не го идентифицира уникално, защото може между два върха, съседи в него, да има повече от едно ребро. Поради това е смислено пътищата в мултиграфите да се описват пълно: и с върховете, и с ребрата.

Свързаност при мултиграфи се дефинира и се използва, но свързаността на мултиграф е същата като свързаността на основния му граф.

Върхово покриване и доминиращо множество на мултиграф може да се дефинират, но и те биха били същите като тези на основния граф.

[†]Пълен мултиграф може да се дефинира, ако се въведе максимален брой ребра с едни и същи краища, като това може да е един и същи максимум за всеки два върха, или отделно дефиниран максимум за всяка двойка върхове. Но това е усложнение, което ние не правим.

Нищо не пречи на мултиграф да бъде двуделен в смисъла на понятието от Секция 2.6, стига да няма примки.

Изоморфизмът между мултиграфи се третира в Подсекция 2.8.1.

Върхово оцветяване на мултиграф е очевидно същото като върховото оцветяване на основния му граф. Реброво оцветяване на мултиграф може да се дефинира по очевидния начин и, за даден мултиграф, то не е същото като за основния му граф, като обаче оптимално реброво оцветяване на основния граф очевидно влече оптимално реброво оцветяване на мултиграфа, и обратно.

По отношение на планарността е от решаващо значение да се разглеждат мултиграфи с примки. Наистина, мултиграф е планарен тогава и само тогава, когато основният му граф е планарен, но въвеждането и дефинирането на дуален граф (на планарен) изисква, в общия случай, да се разглеждат мултиграфи, и то с примки. Ако не разглеждаме мултиграфи, понятието “дуален граф” би било твърде ограничено.

2.3 Пътища и цикли

Неформално казано, нека си представим, че е даден граф и някакво същество живее в графа и по-точно във върховете на графа, но може да ползва ребрата за придвижване. Бивайки в кой да е връх, то може да се придвижи до произволен негов съсед. Нека това същество тръгнало от някакъв връх и стигнало до друг връх, или се е върнало в началния връх, при това минавайки през някакви междинни върхове. "Път в граф" е описание на придвижването на това същество из графа.

Определение 16: Път.

Нека $G = (V, E)$ е граф. *Път в* G наричаме всяка алтерниращa редица от върхове и ребра, за някое $t \geqslant 0$:

$$p = (u_{i_0}, e_{k_0}, u_{i_1}, e_{k_1}, u_{i_2}, \ldots, u_{i_{t-1}}, e_{k_{t-1}}, u_{i_t})$$

където $u_{i_p} \in V$ за $0 \leqslant p \leqslant t$, $e_{k_p} \in E$ за $0 \leqslant p \leqslant t-1$, и освен това е изпълнено $e_{k_p} = (u_{i_p}, u_{i_{p+1}})$ за $0 \leqslant p \leqslant t-1$. Върховете u_{i_0} и u_{i_t} се наричат *краищата на пътя*. Останалите върхове са *вътрешните върхове на пътя*. Още казваме, че p е път *между* u_{i_0} и u_{i_t} и в общия случай нямаме право да разменим u_{i_0} с u_{i_t}.
Дължината на пътя е броят на ребрата в него. Ще бележим дължината на пътя с $|p|$. В случая, $|p| = t$.
Ако всички елементи на пътя—върхове и ребра—са уникални, казваме, че p е *прост път*.

Забележете, че определението допуска път без нито едно ребро (при $t = 0$), така че всеки отделен връх е тривиален път с дължина нула. Път без върхове обаче не може да има съгласно това определение. Също така забележете, че пътищата без ограничения са безброй много в общия случай—тоест, ако графът има поне едно ребро—заради възможността да се "връщаме" по ребра назад, във върхове, в които вече "сме били". От друга страна, простите пътища са само краен брой, щом графът е краен.

На английски традиционно се използва термините *path* за това, което ние сега нарекохме "път", и *simple path* съответно за "прост път". В англоезичната литература има и друга конвенция за именуване: *walk* е алтерниращa редица като дефинираната горе без ограничения, *trail* не допуска повтаряне на ребра, но допуска повтаряне на върхове, а само *path* не допуска нито повтаряне на ребра, нито на върхове. Тъй като на български не е прието да се прави разграничение между тези три възможности с три различни термина, ние ще си останем с "път" и "прост път".

Следната нотация се среща в [18, стр. 7].

Нотация 2: u-v път.

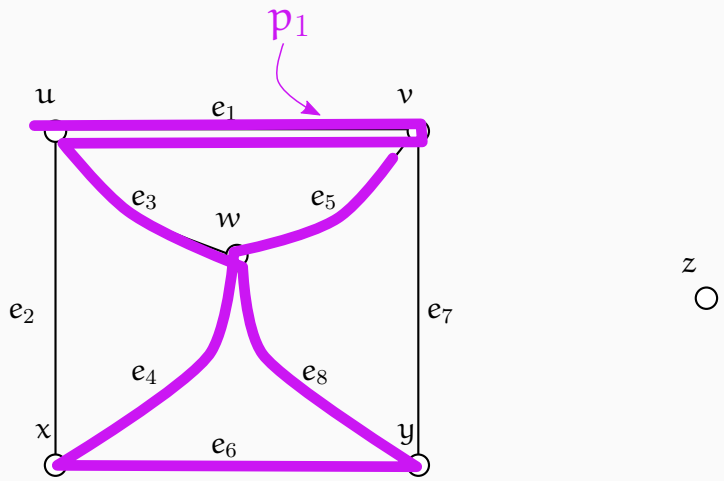
Нека е даден граф G и $u, v \in V(G)$. "u-v път" е кратък запис за "път с краища u и v".

Нека пак разгледаме графа G_1 от Фигура 2.1. Един възможен път в него е

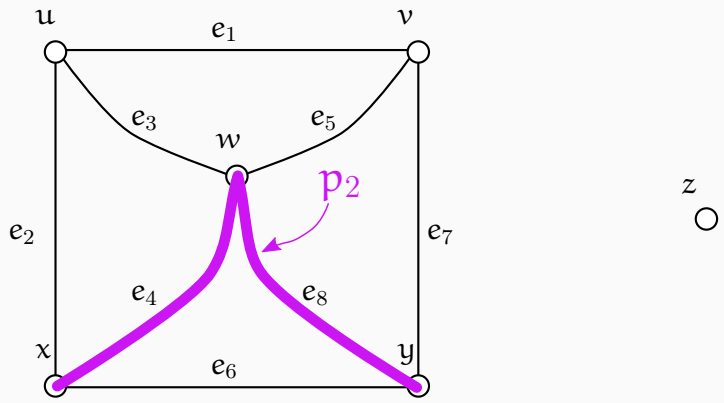
$$p_1 = (u, e_1, v, e_1, u, e_3, w, e_8, y, e_6, x, e_4, w, e_5, v)$$

Фигура 2.19 илюстрира p_1. Забележете, че p_1 не е прост път. Примери за прости пътища са $p_2 = (x, e_4, w, e_8, y)$ и $p_3 = (z)$, като p_2 е показан на Фигура 2.20.

Фигура 2.19 : Път p_1 в графа G_1 от Фигура 2.1.



Фигура 2.20 : Прост път p_2 в графа G_1 от Фигура 2.1.



Конвенция 3

Когато описваме път е допустимо да изпускаме имената на ребрата и кръглите скоби в краищата. Например, може да запишем $p_1 = u, v, u, w, y, x, w, v$, $p_2 = x, w, y$ и $p_3 = z$. Този запис е недвусмислен, защото между два върха не може да има повече от едно ребро и е напълно ясно кои са изпуснаните имена на ребра.

Но дори да използваме икономичния запис без имената на ребрата, помним, че път е алтернираща редица от върхове и ребра, а не редица от върхове.

Конвенция 4

Ако p е път и не сме дали имена на множествата от върховете и ребрата му, то с $V(p)$ ознаваме множеството от неговите върхове и с $E(p)$ означаваме множеството от неговите ребра.

Конвенция 5

Простите пътища, в които няма повтаряне на елементи, се ползват по-често от другия вид (който има повтаряне на елементи). Затова, отсега нататък, като кажем "път", разбираме "прост път". Ако имаме предвид път с повтаряне на елементи, трябва изрично да споменем, че не е непременно прост.

Заслужава си да разсъждаваме над следното. Понятието "път" се използва в теорията на графите в два смисъла, които имат нещо общо, но все пак са различни. От една страна, съгласно Определение 16, "път" е редица от върхове и ребра с определени свойства. От друга страна, често понятието "път" означава подграф на G със своите върхове и ребра. Прочее, Фигура 2.20 показва именно това: подграф на G_1. За да осмислим разликата, да разгледаме следната редица от върхове и ребра в G_1: $p_4 = (y, e_8, w, e_4, x)$. Задаваме си въпроса, дали p_4 е същият обект като p_2, или не. Ако ги гледаме като редици от върхове и ребра, както определението изисква, те са различни обекти (взаимно инверсни пермутации). Ако ги гледаме като подграфи на G_1, както често става на практика, те са един и същи обект (подграф). Поради това възприемаме следната конвенция.

Конвенция 6

Терминът "път в граф" означава, от една страна, редица от върхове и ребра съгласно Определение 16, но от друга страна означава подграф. Второто значение се ползва по-често на практика, а от контекста трябва да е ясно кое значение имаме предвид.

Подчертаваме, че има смисъл да гледаме на път p като на подграф **само ако p е прост път**; ако има повтаряне на върхове или ребра, то p и подграфът $(V(p), E(p))$ са принципно различни неща.

Наблюдение 7

На всеки път-подграф с дължина поне единица съответстват точно два пътя-редици, всеки от които е инверсната пермутация на другата. Можем да кажем, че всеки път-подграф, който има поне едно ребро, има две различни описания, които са редици от върхове и ребра.

Например, на пътя-подграф на G_1, отбелязан на Фигура 2.20 в лилаво, съответстват два различни пътя-редици, а именно p_2 и p_4, които са негови описания.

И накрая, *подпът* на даден път p ще наричаме всяка непрекъсната подредица на p, която отговаря на изискванията за път, тоест е алтерниращa редица от върхове и ребра, започваща и завършваща на връх. Подчертаваме, че става дума за непрекъсната подредица. Например, ако $p = (u, e_1, v, e_2, w, e_3, x, e_4, y)$ е път, то подпътища са, например, (u), (u, e_1, v, e_2, w) и (v, e_2, w), но (u, e_2, w) не е подпът.

Понякога се налага да разглеждаме два или повече пътя, които нямат общи върхове с изключение на това, че може да имат общи краища. Следното определение се среща например в [18, стр. 7].

Определение 17: Независими пътища.

Нека $p_1, \ldots, p_k$ са пътища в един и същи граф G. Казваме, че $p_1, \ldots, p_k$ *са независими*, ако нито един от тях не съдържа вътрешни върхове на някой от другите.

В частност, ако става дума само за два пътя, те да са независими означава или да нямат никакви общи върхове, или имат един общ край и нямат други общи върхове, или и двата им края са общи и нямат други общи върхове.

Определение 18: Цикъл.

Нека $G = (V, E)$ е граф и p е път в него, където:

$$p = (u_{i_0}, e_{k_0}, u_{i_1}, e_{k_1}, u_{i_2}, \ldots, u_{i_{t-1}}, e_{k_{t-1}}, u_{i_t})$$

Казваме, че p е *цикъл*, ако $u_{i_0} = u_{i_t}$. Казваме, че p е *прост цикъл*, ако p е цикъл с поне едно ребро и освен това, всички елементи освен $u_{i_0} = u_{i_t}$ са уникални.

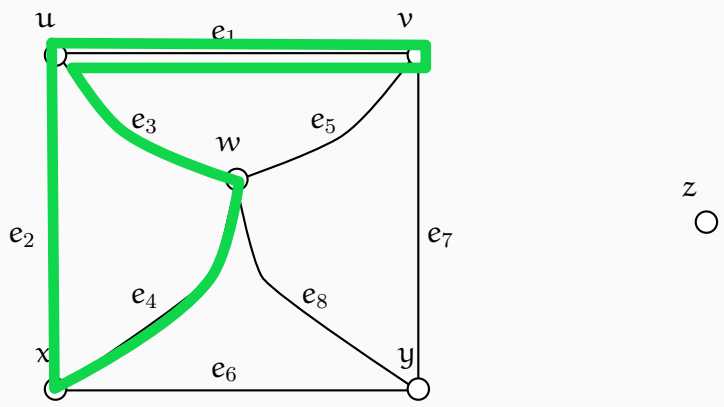
Иначе казано, цикъл е път, на който краищата съвпадат. Определението допуска цикъл без нито едно ребро (при $t = 0$), така че всеки отделен връх е тривиален цикъл с дължина нула. Обаче **не може да има прост цикъл без ребра**.

Тъй като "цикъл" е частен случай на път, дефиницията на "дължина" е същата: дължината на цикъла е броят на ребрата в него. Отново използваме нотацията $|p|$ за дължината.

Нека пак разгледаме графа G_1 от Фигура 2.1. Един цикъл в него е

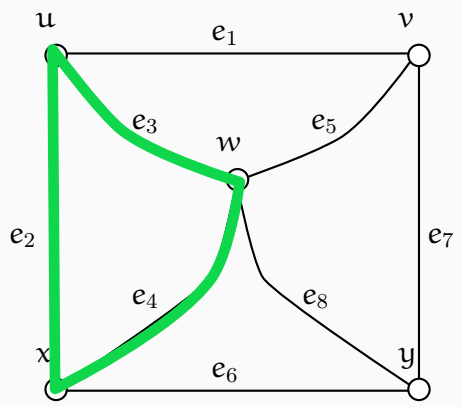
$$c_1 = (u, e_1, v, e_1, u, e_3, w, e_4, x, e_2, u)$$

Фигура 2.21 : Цикъл c_1, който не е прост, в графа G_1 от Фигура 2.1.



Цикълът c_1 е показан на Фигура 2.21. c_1 не е прост цикъл. Прост цикъл е $c_2 = (u, e_3, w, e_4, x, e_2, u)$, който е изобразен на Фигура 2.22.

Фигура 2.22 : Прост цикъл c_2 в графа G_1 от Фигура 2.1.



Подобно на пътищата, циклите може да се описват като пропускаме имената на ребрата и записваме само имената на върховете. В случая, $c_1 = u, v, u, w, x, u$ и $c_2 = u, w, x, u$.

Лесно се забелязва, че минималната дължина на прост цикъл в обикновен граф е 3. Индивидуален връх като u или z в последния пример представлява цикъл (с дължина 0), но не е прост цикъл. Едно ребро, например $e_1 = (u, v)$ е цикъл с дължина 2, ако го разгледаме като u, e_1, v, e_1, u, но това не е прост цикъл, защото e_1 също се повтаря. И така, е точната долна граница за дължината на простите цикли в обикновените графи е 3. Забележете разликата между обикновените графи и мултиграфите, при които може да има прости цикли с дължина 1 или 2 (вижте Наблюдение 6).

Конвенция 12 е аналог на Конвенция 7 от неориентираните графи.

Конвенция 7

Простите пътища и простите цикли се ползват по-често от другия вид. Затова, отсега нататък, като кажем "цикъл", разбираме "прост цикъл". Ако имаме предвид цикъл, който не е прост, трябва изрично да споменем, че не е прост.

Граф, в който няма цикли, се нарича *ациклични*. Тъй като тук имаме предвид прости цикли, това определение е смислено; ако имахме предвид цикли, които не са непременно прости, ациклични графи нямаше да има, защото всеки отделен връх е цикъл съгласно Определение 18.

Четен цикъл е всеки цикъл с четна дължина. *Нечетен цикъл* е всеки цикъл с нечетна дължина.

Конвенция 8 и Наблюдение 8 са аналогични съответно на Конвенция 6 и Наблюдение 7.

Конвенция 8

Терминът "цикъл в граф" означава, от една страна, редица от върхове и ребра съгласно Определение 18, но от друга страна означава подграф. Второто значение се ползва по-често на практика, а от контекста трябва да е ясно кое значение имаме предвид.

Подчертаваме, че има смисъл да гледаме на цикъл c като на подграф **само ако p е прост цикъл**; ако има повтаряне на върхове или ребра, то c и подграфът $(V(c), E(c))$ са принципно различни неща.

Наблюдение 8

На всеки цикъл-подграф с дължина k съответстват точно $2k$ на брой цикли-редици, които се получават една от друга с ротация и/или рефлексия. Можем да кажем, че всеки цикъл-подграф c има $2|c|$ различни описания, които са редици от върхове и ребра.

Например, цикълът c_2 от Фигура 2.22 има следните различни описания:

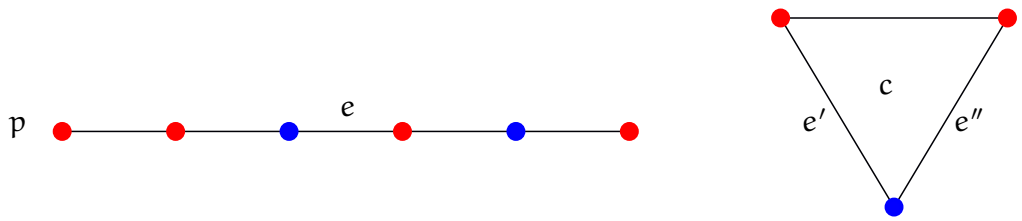
$u, e_3, w, e_4, x, e_2, u$
$x, e_2, u, e_3, w, e_4, x$
$w, e_4, x, e_2, u, e_3, w$
$u, e_2, x, e_4, w, e_3, u$
$x, e_4, w, e_3, u, e_2, x$
$w, e_3, u, e_2, x, e_4, w$

Следното твърдение е прекалено очевидно, за да бъде наречено "лема" или "теорема", но е добре да бъде изразено в явен вид, а не да се ползва тихомълком нататък.

Наблюдение 9

Нека p е път и $V(p)$ има разбиване $\{V_1, V_2\}$. Тогава в $E(p)$ има ребро e, такова че единият край на e е във V_1, а другият му край е във V_2.
Нека c е цикъл и $V(c)$ има разбиване $\{V_1, V_2\}$. Тогава в $E(c)$ има две различни ребра e' и e'', които може да имат или да нямат общ край, такива че за всяко от тях, единият край е във V_1, а другият е във V_2.

Ето пример за път p и цикъл c, като върховете и на пътя, и на цикъла са разбити на два дяла, които са означени със син и червен цвят[†]. Както се вижда, в пътя има ребро e с краища в различни цветове, а в цикъла има две ребра e' и e'', краищата на всяко от които са в различни цветове.



[†]Нарочно не казваме "върховете са оцветени в два цвята", за да не става объркване с върхово оцветяване на граф (Определение 34). При върховото оцветяване се иска краищата на всяко ребро да са в различни цветове, докато в Наблюдение 9 няма такова изискване.

2.4 Свързаност в графи.

2.4.1 Основни определения и свойства.

Определение 19: Свързаност в граф. Свързан граф.

Нека $G = (V, E)$ е граф. За всеки два върха $u, v \in V$ казваме, че u и v са *свързани*, ако съществува u-v път. G е *свързан*, ако всеки два върха в него са свързани.

Не всеки граф е свързан. Например, графът на Фигура 2.1 на стр. 7 не е свързан, защото връх z не е свързан с никой друг връх.

Нека $Q_G \subseteq V \times V$ е следната релация: $\forall u, v \in V : uQv \leftrightarrow u$ и v са свързани. Да наречем тази релация, *релацията на достижимост върху* G.

Лема 4

Релацията на достижимост е транзитивна.

Доказателство: Ще покажем, че за всеки три върха u, v, w е изпълнено следното: ако има u-v път p и има v-w път q, то има u-w път r. Читателят може да се изкуши да каже, че това е очевидно, ако вземем $r = p \cup q$. Но, ако p и q имат други общи върхове освен v, то $p \cup q$ не е път. Налага се да направим конструкция, която задължително изгражда път.

Без ограничение на общността, нека p е най-къс u-v път и q е най-къс v-w път[†]. Нека връх b е най-близкият до u връх в p от $V(p) \cap V(q)$. Тогава b е най-близкият до w връх в q от $V(p) \cap V(q)$. Нека p' е подпътят на p между u и b включително и q' е подпътят на q между b и w включително. Тогава $r = p' \cup q'$ е прост път между u и w, тъй като $V(p') \cap V(q') = \{b\}$. □

Лема 5

Нека R_G и Q_G са релациите, дефинирани съответно на стр. 8 и на тази страница. Q_G е рефлексивно и транзитивно затваряне на R_G. □

Щом R_G е симетрична и Q_G е рефлексивно и транзитивно затваряне на R_G, то Q_G е рефлексивна, симетрична и транзитивна. Тогава Q_G е релация на еквивалентност. Нейните класове на еквивалентност са множества върхове, във всяко от които всеки два върха са свързани, но никои два върха от различни класове не са свързани. Като пример да разгледаме графа на Фигура 2.1 на стр. 7. Релацията на достижимост върху него има два класа на еквивалентност: $\{u, v, w, x, y\}$ и $\{z\}$.

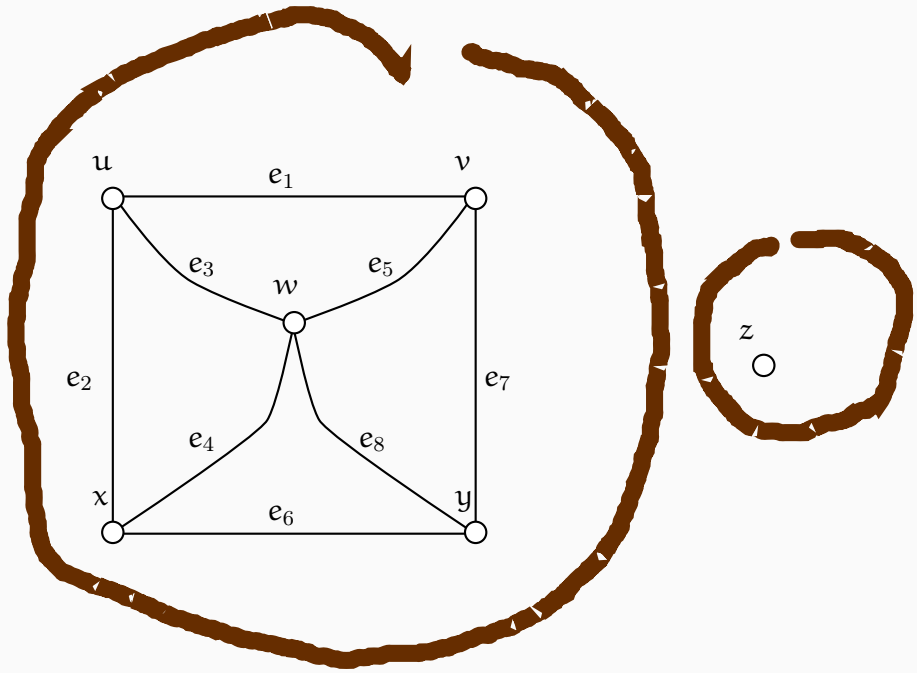
Определение 20: Свързани компоненти.

Нека $G = (V, E)$ е граф и Q_G е релацията на достижимост върху G. Подграфите на G, индуцирани от класовете на еквивалентност на Q_G, се наричат *свързаните компоненти на* G.

Като пример да разгледаме пак графа на Фигура 2.1 на стр. 7. Той има точно две свързани компоненти, които са очертани на Фигура 2.23.

[†]Какво е дължина на път в граф е описано в Подсекция 2.4.2

Фигура 2.23 : Двете свързани компоненти на графа G_1 от Фигура 2.1.



Възможно е следното алтернативно определение на "свързани компоненти", която използва Определение 19.

Определение 21: Свързани компоненти, алтернативно определение.

Свързаните компоненти на граф са максималните по включване свързани подграфи.

Наблюдение 10

Определения 20 и 21 са еквивалентни.

Ясно е, че пълният граф е свързан за всяко n, а празният не е свързан при $n \geqslant 2$. Неформално казано, при фиксирано множество от върхове, наличието на повече ребра благоприятства свързаността, а на по-малко ребра, несвързаността. Теорема 2 и Теорема 3 дават количествени резултати в този смисъл.

Теорема 2: Точна долна граница за $\delta(G)$ в гарантирано свързан граф

Нека $G = (V, E)$ е граф. Ако $\delta(G) \geqslant \left\lceil \frac{n-1}{2} \right\rceil$, то G е свързан. Ако $\delta(G) < \left\lceil \frac{n-1}{2} \right\rceil$, то G може да не е свързан.

Доказателство: Да допуснем, че $\delta(G) \geqslant \left\lceil \frac{n-1}{2} \right\rceil$ и G не е свързан. Тогава G има поне две свързани компоненти G_1 и G_2. Нека те имат съответно n_1 и n_2 върха. Без ограничение на общността, нека $n_1 \geqslant n_2$. Тогава $n_2 \leqslant \left\lfloor \frac{n}{2} \right\rfloor$; ако допуснем, че $n_2 \geqslant \left\lfloor \frac{n}{2} \right\rfloor + 1$, то графът трябва да има поне $2\left(\left\lfloor \frac{n}{2} \right\rfloor + 1\right) = 2\left\lfloor \frac{n}{2} \right\rfloor + 2$ върха, а това е невъзможно, тъй като $2\left\lfloor \frac{n}{2} \right\rfloor + 2 > n$. И така, $n_2 \leqslant \left\lfloor \frac{n}{2} \right\rfloor$.

Съгласно Наблюдение 2, $\Delta(G_2) \leqslant n_2 - 1$, от което следва, че:

$$\Delta(G_2) \leqslant \left\lfloor \frac{n}{2} \right\rfloor - 1 \tag{2.1}$$

От друга страна, $\delta(G) \geqslant \left\lceil \frac{n-1}{2} \right\rceil$ по допускане и, очевидно, $\delta(G_2) \geqslant \delta(G)$, така че:

$$\delta(G_2) \geqslant \left\lceil \frac{n-1}{2} \right\rceil \tag{2.2}$$

Но:

$$\left\lceil \frac{n-1}{2} \right\rceil > \left\lfloor \frac{n}{2} \right\rfloor - 1 \tag{2.3}$$

което се доказва тривиално, ако разгледаме случаите n четно и n нечетно. От (2.1), (2.2) и (2.3) следва веднага, че $\delta(G_2) > \Delta(G_2)$, което е невъзможно. Следователно G е свързан.

Сега да допуснем, че $\delta(G) < \left\lceil \frac{n-1}{2} \right\rceil$. Първо да допуснем, че n е четно. Да речем, $n = 2s$. Тогава $\delta(G) < \left\lceil \frac{2s-1}{2} \right\rceil$, а $\left\lceil \frac{2s-1}{2} \right\rceil = s$. Тогава $\delta(G) \leqslant s - 1$. Очевидно съществува граф с $2s$ върха, в който минималната степен на връх е $s - 1$ и който не е свързан – това е граф, състоящ се от и само от две копия на K_s, които нямат общ връх. Сега да допуснем, че n е нечетно. Да речем, $n = 2s+1$. Тогава $\delta(G) < \left\lceil \frac{2s+1-1}{2} \right\rceil$, а $\left\lceil \frac{2s+1-1}{2} \right\rceil = \left\lceil \frac{2s}{2} \right\rceil = s$. Тогава $\delta(G) \leqslant s-1$. Очевидно съществува граф с $2s + 1$ върха, в който минималната степен на връх е $s - 1$ и който не е свързан – това е граф, състоящ се от и само от едно копие на K_s и едно копие на K_{s+1}, които нямат общ връх. □

Доказателството на Теорема 3 се основава на Лема 6.

Лема 6: Максимален брой ребра при даден брой свързани компоненти.

Нека $G = (V, E)$ е граф с k свързани компоненти, където $1 \leqslant k \leqslant n$. Тогава $m \leqslant \binom{n-k+1}{2}$.

Доказателство: Нека свързаните компоненти са $G_1, \ldots, G_k$, и G_i има n_i върха за $1 \leqslant i \leqslant k$. Тъй като компонентите имат поне по един връх, очевидно $1 \leqslant n_i \leqslant n-k+1$ за всяко i. Освен това, $\sum_{i=1}^{k} n_i = n$. При дадени $n_1, \ldots, n_k$, броят на ребрата се максимизира, когато всеки G_i е пълен граф – това е очевидно.

Сега ще докажем, че броят на ребрата в G е максимален, когато една свързана компонента съдържа $n - k + 1$ върха, което означава, че тя съдържа $\binom{n-k+1}{2}$ ребра, а останалите компоненти имат по един връх, което означава, че имат по нула ребра. Това следва лесно от факта, че функцията-брой на ребрата е квадратична в броя на върховете, но ще извършим доказателството подробно. Да си представим алгоритъм, който получава свързаните компоненти $G_1, \ldots, G_k$ със съответно $n_1, \ldots, n_k$ върха и прави следното: докато не е вярно, че всички свързани компоненти без една имат по точно един връх, разглежда две компоненти G_i и G_j със съответно n_i и n_j върха (където $n_i > 1$ и $n_j > 1$) и прехвърля по един връх от едната от тях в другата съгласно следното правило:

- ако G_i и G_j имат еднакъв брой върхове, прехвърляме връх от коя да е от тях в другата,
- в противен случай прехвърляме от тази с по-малко върхове в другата.

докато една от тях не остане само с един връх. След всяко прехвърляне на връх, компонентата, която получава връх, става пак пълен граф (с добавяне на всички необходими за целта

ребра), а тази, от която се вади връх, остава пълен граф, но на върхове с един по-малко от преди.

Без ограничение на общността, нека $n_i \geqslant n_j$ и нека прехвърлим един връх от G_j в G_i. Да видим как това се отразява на броя на ребрата след добавяне и махане на необходимите бройки ребра. Преди прехвърлянето на връх, в тези компоненти е имало $\binom{n_i}{2} + \binom{n_j}{2}$ ребра. След прехвърлянето и добавянето и махането на ребра, така че G_i и G_j пак да са пълни графи е вярно, че G_i вече има $\binom{n_i+1}{2}$ ребра, а G_j, само $\binom{n_j-i}{2}$ ребра. Тогава общият брой ребра на G нараства с

$$\left(\underbrace{\binom{n_i+1}{2} + \binom{n_i-1}{2}}_{\text{брой ребра в } G_i \text{ и } G_j \text{ след прехвърлянето}} \right) - \left(\underbrace{\binom{n_i}{2} + \binom{n_j}{2}}_{\text{брой ребра в } G_i \text{ и } G_j \text{ преди прехвърлянето}} \right)$$

тъй като в останалите свързани компоненти не се мени нищо. Но

$$\begin{aligned}
&\binom{n_i+1}{2} + \binom{n_j-1}{2} - \binom{n_i}{2} - \binom{n_j}{2} = \\
&\frac{1}{2}((n_i+1)n_i + (n_j-1)(n_j-2) - n_i(n_i-1) - n_j(n_j-1)) = \\
&\frac{1}{2}(n_i^2 + n_i + n_j^2 - 3n_j + 2 - n_i^2 + n_i - n_j^2 + n_j) = \frac{1}{2}(2n_i - 2n_j + 2) = n_i - n_j + 1
\end{aligned}$$

Излиза, че дори да започнем да прехвърляме при равен брой върхове в двете компоненти пак "печелим" едно ребро, а ако прехвърляме от по-малка като брой върхове компонента в по-голяма, "печалбата" е дори по-голяма.

Лесно се вижда, че всяка итерация на този алгоритъм увеличава с поне единица броя на ребрата в графа, и крайният брой ребра, независимо от поредицата избори от коя в коя компонента да прехвърляме, е $\binom{n-k+1}{2}$. Това е абсолютният максимум за броя на ребрата при k свързани компоненти. Както стана ясно от доказателството, тази горна граница е достижима. □

Теорема 3: Точна долна граница за m в гарантирано свързан граф

Нека $G = (V, E)$ е граф. Ако $m \geqslant \binom{n-1}{2} + 1$, то G е свързан. Ако $m \leqslant \binom{n-1}{2}$, то G може да не е свързан.

Доказателство: От Лема 6 знаем, че при две свързани компоненти максималният брой ребра е $\binom{n-2+1}{2} = \binom{n-1}{2}$, а при повече от две свързани компоненти е дори по-малък. Следователно, ако ребрата са повече от $\binom{n-1}{2}$, няма как да има повече от една свързана компонента; с други думи, графът е свързан.

От друга страна, ако ребрата са точно $\binom{n-1}{2}$, може да има две свързани компоненти, едната от които е пълен граф на $n-1$ върха, а другата е изолиран връх. □

Теорема 4: Изтриване на ребро от цикъл на свързан граф

Нека $G = (V, E)$ е свързан граф. Нека c е цикъл в G и нека e е ребро от c. Тогава $G - e$ е свързан.

Доказателство: Ще докажем, че за всеки $x, y \in V$ има x-y път в $G - e$. Тъй като G е свързан, то в G съществува x-y път p. Ако e не е ребро от p, то пътят p съществува и в $G - e$.

Да разгледаме възможността e да бъде ребро от p. Тогава p не е път в $G - e$ и трябва да посочим друг път между x и y. Нека краищата на e са върховете u и v – очевидно това са върхове от цикъла c. Без ограничение на общността, нека пътят p в G съдържа върховете u и v в този ред:

$$p = x \ldots\ldots u, v \ldots\ldots y$$

С други думи, в посока от x към y, връх u се появява преди връх v в p. Забележете, че u и v са съседи в p, понеже реброто $e = (u, v)$ е в p, а p е прост път – щом p е прост път, то тази поява на u в него е единствена и тази поява на v в него е единствена. Възможно е x да съвпада с u или y да съвпада с v. Нека a е първият връх от c, който се появява в p, ако гледаме на p в посока от x към y. Разбира се, a може да съвпада с x или u, но без ограничение на общността да разгледаме общия случай, в който x, a и u са различни върхове. Аналогично, b е последният връх от c, който се появява в p, ако гледаме на p в посока от x към y. b може да съвпада с v или y, но без ограничение на общността да разгледаме общия случай, в който v, b и y са различни върхове. Тогава p има вида

$$p = x \ldots\ldots a \ldots\ldots u, v \ldots\ldots b \ldots\ldots y$$

Нека p' е подпътят на p от x включително до a включително. Нека p'' е подпътят на p от b включително до y включително:

$$p = \underbrace{x \ldots\ldots a}_{p'} \ldots\ldots u, v \ldots\ldots \underbrace{b \ldots\ldots y}_{p''}$$

Но a и b са различни върхове, тъй като p е прост път. Тогава c се получава от обединението на два пътя, да ги наречем q_1 и q_2, и двата от които имат крайни върхове a и b:

$$c = q_1 \cup q_2$$

Точно единият от q_1 и q_2 съдържа реброто e (което означава, че съдържа и u и v). Без ограничение на общността, нека това е q_1. Тогава подпътят на p между a включително и b включително е q_1:

$$p = \underbrace{x \ldots\ldots \overbrace{a \ldots\ldots u, v \ldots\ldots b}^{q_1}}_{p'} \underbrace{\ldots\ldots y}_{p''}$$

С други думи, $p = p' \cup q_1 \cup p''$. Но изтриването на e не касае q_2. След изтриването на e, q_1 вече не е път (в $G - e$), но q_2 остава път (в $G - e$). Нещо повече: $p' \cup q_2 \cup p''$ е път в $G - e$ с краища x и y. □

2.4.2 Разстояния в графи. Диаметър, радиус и център на граф.

Определение 22: Разстояние в свързан граф.

Нека $G = (V, E)$ е свързан граф. За всеки два не непременно различни върха u и v, *разстоянието между* u *и* v е дължината на най-къс път между u и v. Разстоянието между u и v се бележи с $\mathrm{dist}(u, v)$.

Не е добра идея да се каже "най-къси**ят** път", защото може да има много най-къси пътища. Но може алтернативно да се каже "минимална**та** дължина на u-v път".

Определение 23: Метрика.

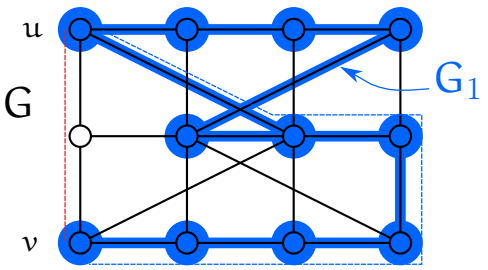
Нека A е произволно множество и $f : A \times A \to \mathbb{R}^+ \cup \{0\}$. Казваме, че f *е метрика*, ако:

$$\forall x, y \in A : f(x, y) = 0 \leftrightarrow x = y$$
$$\forall x, y \in A : f(x, y) = f(y, x)$$
$$\forall x, y, z \in A : f(x, y) + f(y, z) \geqslant f(x, z)$$

Наблюдение 11

Функцията dist от Определение 22 е метрика.

За пример да вземем свързания граф, показан на Фигура 2.13. В него $\text{dist}(u, u) = 0$, $\text{dist}(v, z) = 1$, $\text{dist}(u, v) = 2$ и така нататък.



Фигура 2.24: Най-късият път между u и v в G е очертан с червена прекъсната линия. Най-късият път между u и v в G_1 е очертан със синя прекъсната линия.

Понякога разглеждаме едновременно свързан граф G, негов свързан подграф G_1 и някакви върхове u и v, които са върхове както в G, така и в G_1. Лесно се вижда, че разстоянието между u и v в оригиналния граф може да е различно от разстоянието между u и v в подграфа. Като пример да разгледаме графа G на Фигура 2.24 и някакъв негов подграф G_1, очертан в синьо. Разстоянието между u и v в G е 2, защото такава е дължината на най-късия път между u и v в G. Каква е дължината на най-къс път между u и v в G_1? Този път трябва да се състои изцяло от сини ребра и сини върхове—защото е в подграфа. Лесно се вижда, че ако използваме само сини ребра и върхове, най-късият път между u и v има дължината 6. Ако вече сме въвели и G, и G_1, то използването на нотацията "$\text{dist}(u, v)$" е двусмислено; за конкретния пример, $\text{dist}(u, v)$ може да е 2, ако имаме предвид целия граф, но може и да е 6, ако имаме предвид подграфа. За да различаваме разстоянието между u и v в G от разстоянието между u и v в G_1, слагаме индекс G или G_1 така:

$$\text{dist}_G(u, v) = 2$$
$$\text{dist}_{G_1}(u, v) = 6$$

Наблюдение 12: Разстоянието в подграф не надхвърля разстоянието в целия граф

За произволен свързан граф G, за произволен свързан подграф G' на G и за произволни върхове $u, v \in V(G')$ е изпълнено:

$$\text{dist}_G(u, v) \leqslant \text{dist}_{G'}(u, v)$$

□

Определение 24

Нека $G = (V, E)$ е свързан граф и $u \in V$ е произволен. *Ексцентрицитетът на* u е

$$\epsilon(u) = \max\{\mathrm{dist}(u, v) \mid v \in V\}$$

Радиусът на G е

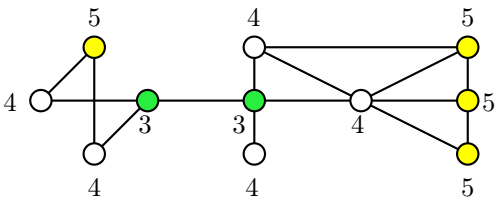
$$\mathrm{rad}(G) = \min\{\epsilon(u) \mid u \in V\}$$

Диаметърът на G е

$$\mathrm{diam}(G) = \max\{\epsilon(u) \mid u \in V\}$$

Център на графа е всеки връх x, такъв че $\epsilon(x) = \mathrm{rad}(G)$. *Периферен връх* на графа е всеки връх x, такъв че $\epsilon(x) = \mathrm{diam}(G)$.

С думи, ексцентрицитетът е максималното разстояние между u и кой да е друг връх на графа; радиусът на графа минималното, по всички върхове, максимално разстояние в графа; диаметърът е максималното, по всички върхове, максимално разстояние в графа. Диаметър и радиус на графи са важни понятия: ако графът моделира комуникационна мрежа, в която има закъснение единица между съседни върхове, то максималното закъснение, което може да се получи между два върха на мрежата, е равно на диаметъра на графа. Също така, върховете-центрове (те може да са повече от един) в графа отговарят на тези върхове на мрежата, които са най-централно разположени в смисъл, че закъснението между тях и кой да е друг връх е минимално. Съответно периферните върхове са най-отдалечено разположените върхове.



Фигура 2.25: Граф с диаметър 5 и радиус 3.

Като пример, да разгледаме графа на Фигура 2.25. До всеки връх на графа е написан неговият ексцентрицитет. Да се убедим, че ексцентрицитетите на върховете са написани вярно. Първо забелязваме, че няма двойки върхове на разстояние повече от 5. После, за всеки от четирите жълти върха, максималното разстояние до друг връх е 5; а именно, за всеки от трите жълти върха вдясно, жълтият връх вляво е на разстояние 5, и съответно левият жълт връх е на разстояние 5 от кой да е от жълтите върхове вдясно. След това забелязваме, че за двата бели върха вляво, върховете на максимално разстояние са жълтите върхове вдясно (на разстояние 4); аналогично, за трите бели върха вдясно, максимално отдалечен връх (на разстояние 4) е жълтият връх вляво. И накрая, за левия зелен връх, максимално отдалечен (на разстояние 3) е всеки от жълтите върхове вдясно; аналогично за десния зелен връх максимално отдалечен (на разстояние 3) е жълтият връх вляво. Тогава графът има диаметър 5 и радиус 3, като в жълто са оцветени периферните върхове, а в зелено, центровете.

От примера на Фигура 2.25 може да помислим, че винаги радиусът е горе-долу половината от диаметъра и по-точно, че $\mathrm{rad}(G) = \left\lceil \frac{\mathrm{diam}(G)}{2} \right\rceil$. Това обаче не винаги е така! Най-общо, следното отношение е в сила

$$\mathrm{rad}(G) \leqslant \mathrm{diam}(G) \leqslant 2 \cdot \mathrm{rad}(G)$$

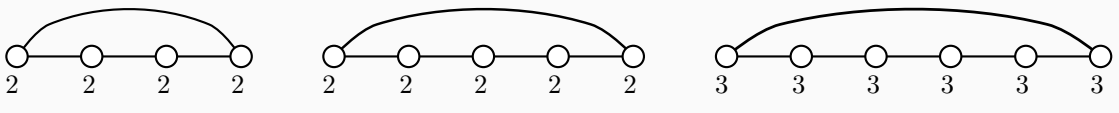
Клас от графи, за които радиусът е равен на диаметъра, са циклите. Във всеки граф-

цикъл[†] с дължина k, за всеки връх u:

$$\epsilon(u) = \left\lfloor \frac{k}{2} \right\rfloor$$

Примери за това има на Фигура 2.26:

Фигура 2.26 : Графи-цикли и ексцентрицитетите на върховете им.



Да си припомним дефиницията на пълния граф K_n от на стр. 15. Лесно се вижда, че диаметърът на K_n е 1 за $n > 1$. Радиусът също е 1 при $n > 1$. Освен това, всеки връх в K_n е едновременно център и периферен връх. Всичко това не е изненада—при пълната свързаност "всеки с всеки" в K_n, понятията диаметър, радиус, център и периферен връх се обезсмислят и формалното прилагане на определенията води до такива на пръв поглед парадоксални изводи.

В Подсекция 2.4.2 досега разглеждахме само свързани графи. Можем ли да дефираме смислено разстояния в графи, които не са свързани? Отговорът е, че можем, само че разстоянието може да не е число. Ако G е не-свързан граф и u и v са върхове от различни свързани компоненти, има смисъл да дефинираме разстоянието между u и v като безкрайност. Формално, това е нещо, което записваме с "∞" и което може да участва в сравнения с числа, като $\infty > x$ за всяко реално x. Резултатът от сравнението $\infty < \infty$ обаче е недефиниран. Сумата от число и ∞ е ∞, а $\infty + \infty$ също е ∞.

Определение 25: Разстояние в не непременно свързан граф.

Нека $G = (V, E)$ е граф. За всеки два не непременно различни върха u и v, *разстоянието между* u *и* v е

- дължината на най-къс път между u и v, ако u и v са свързани,
- ∞, в противен случай.

И при не непременно свързаните графи бележим това разстояние с "$\mathrm{dist}(u, v)$", само че сега е възможно $\mathrm{dist}(u, v) = \infty$.

Разстоянието в не непременно свързаните графи също е метрика (припомнете си Определение 23):

- разстоянието между връх и себе си е нула,
- разстоянието е симетрично в смисъл, че или $\mathrm{dist}(u, v)$ е число и $\mathrm{dist}(u, v) = \mathrm{dist}(v, u)$, или $\mathrm{dist}(u, v) = \infty$, но тогава и $\mathrm{dist}(v, u) = \infty$ и пак $\mathrm{dist}(u, v) = \mathrm{dist}(v, u)$,
- неравенството на триъгълника е изпълнено:

[†]Не казваме просто "цикъл", за да не се бърка понятието с цикъл в граф. Тук имаме предвид, че самият граф е цикъл.

- ако $\mathrm{dist}(u, v)$ и $\mathrm{dist}(v, w)$ са числа, то очевидно u и v са свързани, както и v и w са свързани и $\mathrm{dist}(u, v) + \mathrm{dist}(v, w) \geqslant \mathrm{dist}(u, w)$,
- ако $\mathrm{dist}(u, v)$ е число, а $\mathrm{dist}(v, w)$ е ∞, то u и v са свързани, а v и w не са свързани, което влече, че и u и w не са свързани, ерго, $\mathrm{dist}(u, w) = \infty$; тогава е вярно, че $\mathrm{dist}(u, v) + \mathrm{dist}(v, w) \geqslant \mathrm{dist}(u, w)$, защото и лявата, и дясната страна са ∞,
- ако $\mathrm{dist}(u, v) = \mathrm{dist}(v, w) = \infty$, по пак $\mathrm{dist}(u, v) + \mathrm{dist}(v, w) \geqslant \mathrm{dist}(u, w)$, защото лявата страна е $\infty + \infty = \infty$, а u и w очевидно не са свързани, така че и дясната страна е ∞.

2.4.3 Срязващи върхове и мостове. Разцепване на срязващи върхове. Блокове в графи.

Определение 26: Срязващ връх и срязващо ребро

Нека $G = (V, E)$ е граф. *Срязващ връх* в G е всеки $u \in V$, такъв че $G - u$ има повече свързани компонентни от G. *Срязващо ребро*, още се нарича *мост*, е всяко $e \in E$, такова че $G - e$ има повече свързани компоненти от G.

Определение 27: Срязващ връх и срязващо ребро, алтернативно определение

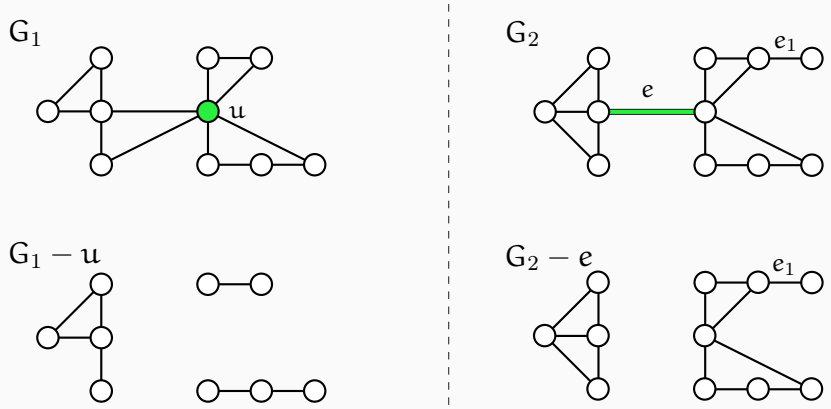
Нека $G = (V, E)$ е граф. Срязващ връх в G е всеки $u \in V$, такъв че за някои различни върхове $v, w \in V$ е вярно, че всеки път между v и w съдържа u като вътрешен връх. Срязващо ребро е всяко $e \in E$, такова че за някои различни върхове $v, w \in V$ е вярно, че всеки път между v и w съдържа e.

Лема 7

Определения 26 и 27 са еквивалентни. □

Лесно се вижда, че на всеки висящ връх в свързана компонента с повече от два върха съответства точно един срязващ връх, а именно единственият му съсед. Самият висящ връх не е срязващ. Ако свързаната компонента има точно два върха, или с други думи, ако е едно единствено ребро, то в нея срязващ връх няма. На всеки висящ връх съответства точно един мост, а именно единственото ребро, инцидентно с него; това остава в сила дори свързаната компонента да се състои само от това ребро. Фигура 2.27 илюстрира Определения 26 и 27. В графа G_1, връх u е срязващ. В графа G_2, и e, и e_1 са мостове, но е показано само изтриването на e.

Фигура 2.27 : Срязващ връх и срязващо ребро.



u е срязващ връх.

e е срязващо ребро (e_1 също).

Допълнение 3: Приложение на срязващи върхове и ребра

Срязващите върхове и мостовете са важни понятия, ако графът моделира някаква свързаност. Например, ако моделира комуникационна мрежа, или пътна мрежа, или електрическа мрежа.

Нека графът моделира комуникационна мрежа от компютри и жични връзки между тях. Нека мрежата е свързана в смисъл, че всеки два компютъра могат да си "говорят"; може би не директно, а през други компютри, но имат връзка един с друг. Тогава срязващ връх в графа съответства на компютър, чиято повреда би довела до това, че мрежата би се разпадала на две или повече подмрежи, които не си "говорят". В разпаднатата мрежа има компютри, които не могат да комуникират взаимно. А срязващо ребро отговаря на жична връзка, чиято повреда би довела до разпадане на мрежата на точно две подмрежи, които не си "говорят".

И така, срязващите върхове и мостовете отговарят на някакъв критично важни компоненти на мрежата. Мрежа, чийто съответен граф има срязващи върхове или мостове е потенциално по-ненадеждна от мрежа, чийто съответен граф няма такива елементи. От гледна точка на надеждността, очевидно предпочитаме мрежи, чиито съответни графи нямат срязващи върхове или мостове.

Определение 28: Разцепване на срязващи върхове

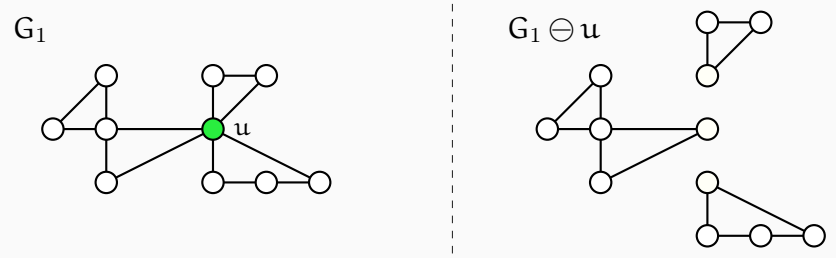
Нека $G = (V, E)$ е граф и нека $u \in V$ е срязващ връх. Нека свързаните компоненти на G са $C_1, \ldots, C_\ell$ и нека $u \in V(C_1)$. Нека $U = N(u)$. Нека $C_1 - u$ има k свързани компоненти, наречени $G_1, \ldots, G_k$. Нека $U_i = V(G_i) \cap U$, за $1 \leqslant i \leqslant k$. Нека $z_1, \ldots, z_k$ са k нови върхове; те не са от V. Нека, за $1 \leqslant i \leqslant k$,

$$E_i = \{(z_i, x) \mid x \in U_i\}$$
$$H_i = (V(G_i) \cup \{z_i\}, E(G_i) \cup E_i)$$

Очевидно H_i са свързани графи. Да *разцепим* u означава да преобразуваме G, заменяйки C_1 със свързаните компоненти $H_1, \ldots, H_k$, а свързаните компоненти $C_2, \ldots, C_\ell$ останат непокътнати. Графът–резултат означаваме с $G \ominus u$.

Неформално казано, разцепването на срязващ връх u се състои в създаването на негови копия (самият u изчезва), на брой колкото са свързаните компоненти **спрямо него**, всяко копие бива свързано с точно една от тези компоненти, с точно тези върхове в нея, с които u е бил свързан. Подчертаваме, че разцепването на върхове е приложимо само към срязващи върхове, въпреки че има очевидно обобщение—разцепване на множество от върхове, което множество е обобщение на "срязващ връх" (Секция 2.12). Фигура 2.28 показва разцепване на връх – след разцепването на u се появяват три свързани компоненти.

Фигура 2.28 : Разцепване на връх.



u е срязващ връх. След разцепването на u.

Операцията "разцепване на срязващ връх" е асоциативна: можем да разцепваме различни върхове в каквато искаме последователност и резултатът е един и същи. Ако $u_1, \ldots, u_t$ са различни срязващи върхове в G, пишем $G \ominus u_1 \ominus \cdots \ominus u_t$. Резултатът от разцепването на всички срязващи върхове е предмет на следващата дефиниция.

Определение 29: Блокове в граф

Нека $G = (V, E)$ е граф. *Блок в* G е всеки максимален по включване свързан подграф, който няма срязващи върхове.

Определение 30: Блокове в граф, алтернативна дефиниция

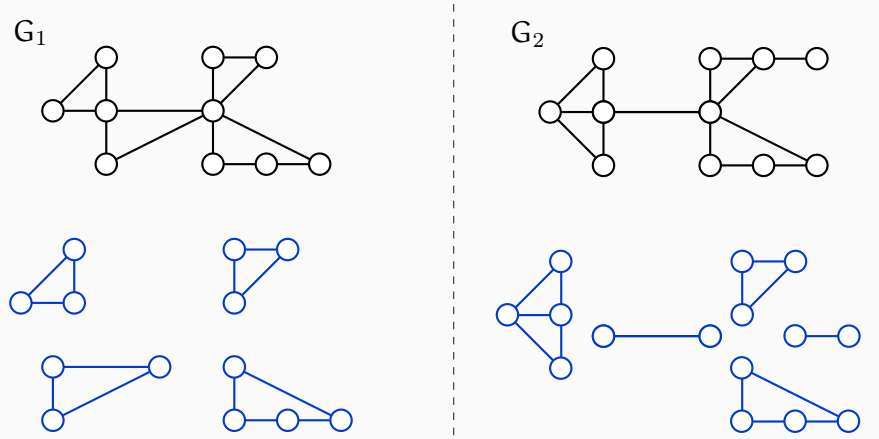
Нека $G = (V, E)$ е граф. *Блоковете на* G са тези свързани компоненти, които се получават от разцепването на всички срязващи върхове.

Наблюдение 13

Определения 29 и 30 са еквивалентни.

Фигура 2.29 илюстрира блоковете на два графа G_1 и G_2. Тя илюстрира и "разцепване на срязващи върхове":

Фигура 2.29 : Блокове в графи.



Блоковете на G_1 в синьо. Блоковете на G_2 в синьо.

2.4.4 Срез в граф

Да си припомним, че k-*разбиване на множество* A за някакво $k \in \mathbb{N}^+$ е всяко множество $X = \{S_1, S_2, \ldots, S_k\}$, такова че

- $S_i \subseteq A$, $1 \leqslant i \leqslant k$.
- $S_i \neq \varnothing$, $1 \leqslant i \leqslant k$.
- $S_i \cap S_j = \varnothing$, $1 \leqslant i < j \leqslant k$.
- $\bigcup_{i=1}^{k} S_i = A$.

Определение 31: Срез в граф.

Нека $G = (V, E)$ е граф. *Срез в* G наричаме всяко 2-разбиване X на V. Ако $X = \{V_1, V_2\}$, то *срез-множеството на* X е

$$E' = \{(u, v) \in E \mid u \in V_1 \wedge v \in V_2\}$$

За всяко ребро $e \in E'$ казваме, че e *прекосява среза* X. *Теглото на* X е $|E'|$. *Страните на среза* са V_1 и V_2.

Забележка: ако графът е тегловен (вж. Секция 3.2), то теглото на среза е сумата от теглата на ребрата, които прекосяват среза.

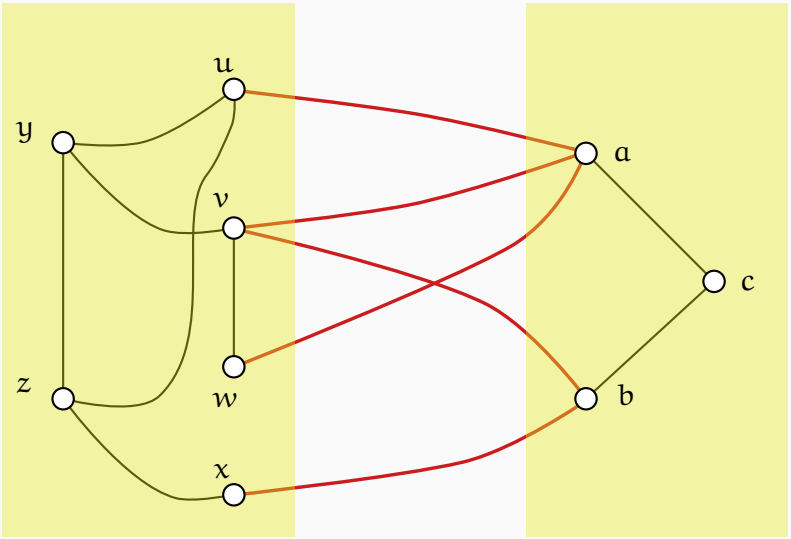
Понятието "срез" има голяма практическа важност. Задачата за намиране на *максимален поток в граф*, която е от огромна важност в оптимизацията, има бързо алгоритмично решение, което използва срезове в графи[†]. Повече за тази задача има в Секция 4.5. Задачата за максимален поток в граф се дефинира върху ориентирани графи, но "срез в ориентирани графи" е тривиално обобщение на току-що въведеното понятие "срез"

Друго важно приложение на срезовете е в областта на компютърното зрение [48, глава 5, стр. 79].

Понякога срезът се дефинира не като разбиването на V, а просто като множеството от ребрата, които го прекосяват, което множество тук нарекохме "срез-множеството" [48]. Често в практиката са интересни срезове, които се дефинират спрямо два предварително избрани върха: нека предварително са фиксирани различни върхове $s, t \in V$ и тогава всеки срез, такъв че s и t се намират в различни негови страни, се нарича s/t-срез.

По правило срезовете в графи се онагледяват чрез рисунки, в които едната страна на среза се рисува вляво, а другата, вдясно. Това много прилича на начина, по който се рисуват двуделните графи (вж. Секция 2.6). Естествено, не всеки граф е двуделен, докато срез може да се дефинира във всеки граф, така че теорията на срезовете в графи и теорията на двуделните графи са различни неща. Пример за рисунка на срез в граф е показана на Фигура 2.30.

Фигура 2.30 : Срез в граф.



Ребрата от срез-множеството са в червено. Двете страни на среза са маркирани с жълта основа.

[†]На английски: *graph cut*.

2.5 Върхово покриване и доминиращо множество на граф

Допълнение 4: За практическата важност на понятията

Да си представим n на брой експеримента, които се извършват в някаква лаборатория. Резултатите от някои двойки експерименти си противоречат; от други двойки, не. Иска се да се намери минимално подмножество от експерименти, чието игнориране води до изчезване на противоречията. Да моделираме задачата с граф. Върховете на на графа отговарят на експериментите, а ребрата, на противоречията. Иска се да се намери минимално подмножество от върхове, такива че тяхното изтриване от графа да не оставя нито едно ребро. Множество от върхове, чието изтриване води до това, че не остават ребра, се нарича "върхово покриване". С други думи, търсим минимално върхово покриване.

Сега да си представим, че са дадени някакви градове и пътища между тях. Иска се да се построят болници в градовете, но—за да се икономисат пари—не във всеки град. Изискването е, за всеки град, или в него да има болница, или в съседен град да има болница, и да бъдат построени колкото е възможно по-малко болници. Ако моделираме задачата с граф, иска се да се намери минимално подмножество от върхове, такива че всеки връх на графа да е един от тях, или да има съсед измежду тях. Множество от върхове, такова че всеки връх е в него или има съсед в него, се нарича "доминиращо множество". С други думи, в тази задача търсим минимално доминиращо множество.

Определение 32: Върхово покриване и доминиращо множество.

Нека $G = (V, E)$ е граф и $U \subseteq V$. Казваме, че U е *върхово покриване на* G, ако:

$$\forall (u, v) \in E : u \in U \vee v \in U$$

Казваме, че U е *доминиращо множество на* G, ако

$$\forall u \in V(u \in U \vee \exists v \in N(u)(v \in U))$$

Като алгоритмични задачи, и двете са минимизационни. Трудно е да се намерят малки върхови покривания и малки доминиращи множества. Най-малката мощност на върхово покриване на G се бележи с $\tau(G)$ и се нарича *число на покриване на* G. Най-малката мощност на доминиращо множество на G се бележи с $\gamma(G)$ и се нарича *доминиращо число на* G.

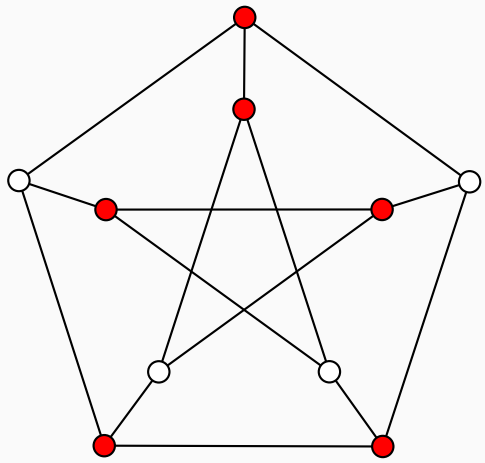
Примери за върхово покриване и доминиращо множество са показани на Фигура 2.31. Графът P, изобразен върху двете подфигури, се нарича граф на Petersen. Графът на Petersen не може да бъде покрит с по-малко от 6 върха: в лявата страна на Фигура 2.31 виждаме, че само външният петоъгълник[†] иска поне 3 върха, а независимо от това, вътрешните пет диагонала[‡] искат също поне 3 върха, така че 6 е долна граница за мощността на върховото покриване и върховото покриване на Фигура 2.31 е оптимално. И така, $\tau(P) = 6$.

От друга страна, графът на Petersen не може да бъде доминиран от по-малко от 3 върха. Защо? – защото графът на Petersen е 3-регулярен, а в 3-регулярен граф един връх може да доминира най-много 4 върха общо (включително и себе си). Тогава кои да е два върха могат да доминират най-много $2 \cdot 4 = 8$ върха общо (включително и себе си). Графът на Petersen

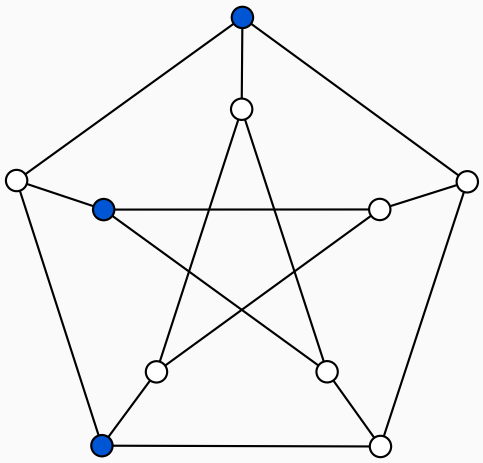
[†]Подграфът, индуциран от външните пет върха.
[‡]Подграфът, индуциран от вътрешните пет върха.

има 10 върха, следователно не може да бъде доминиран от 2 върха. И така, $\gamma(P) = 3$, и доминиращото множество, показано в дясната страна на Фигура 2.31, е опитмално.

Фигура 2.31 : Върхово покриване и доминиращо множество в графа на Petersen.



Върхово покриване с 6 върха.



Доминиращо множество с 3 върха.

Определенията на върхово покриване и доминиращо множество привидно си приличат, но всъщност са много различни. Разликата в мощностите на минимално върхово покриване и минимално доминиращо множество може да е драстична. Например, всяко минимално върхово покриване на K_n има точно $n - 1$ върха (налага се да изтрием $n - 1$ върха, за да не останат ребра), докато всяко минимално доминиращо множество на K_n има точно 1 връх (избираме кой да е връх и забелязваме, че всички останали са му съседи). От друга страна, празният граф на n върха има минимално върхово покриване с мощност 0 (става дума за празното множество; тъй като този граф няма ребра, няма нужда да изтриваме върхове изобщо, така че да не останат ребра), но минималното доминиращо множество е с мощност n (всички върхове трябва да са в него, понеже няма ребра и никой връх не можа да доминира друг връх).

Сравнете Наблюдение 14 с Наблюдение 5. Наблюдение 5 казва, че клика в даден граф влече антиклика в графа-допълнение, и обратното. А Наблюдение 14 казва, че върху един и същи граф, върхово покриване и антиклика са комплементарни в смисъл, че допълнението на върхово покриване до V е антиклика, и обратното.

Наблюдение 14

Нека $G = (V, E)$ е произволен граф. За всяко $U \subseteq V$ е изпълнено следното: U е върхово покриване тогава и само тогава, когато $V \setminus U$ е антиклика. □

Следствие 3

$\alpha(G) + \tau(G) = n$, за всеки граф G.

Наблюдение 15

Нека $G = (V, E)$ е граф без изолирани върхове. Тогава всяко върхово покриване е доминиращо множество. Конверсното не е вярно. □

Забележка: конверсното твърдение на импликацията $p \to q$ е импликацията $q \to p$. В случая, конверсното твърдение би било "всяко доминиращо множество е върхово покриване".

Следствие 4

Нека $G = (V, E)$ е граф без изолирани върхове. Тогава $\tau(G) \geqslant \gamma(G)$. □

Допълнение 5: За графа на Petersen

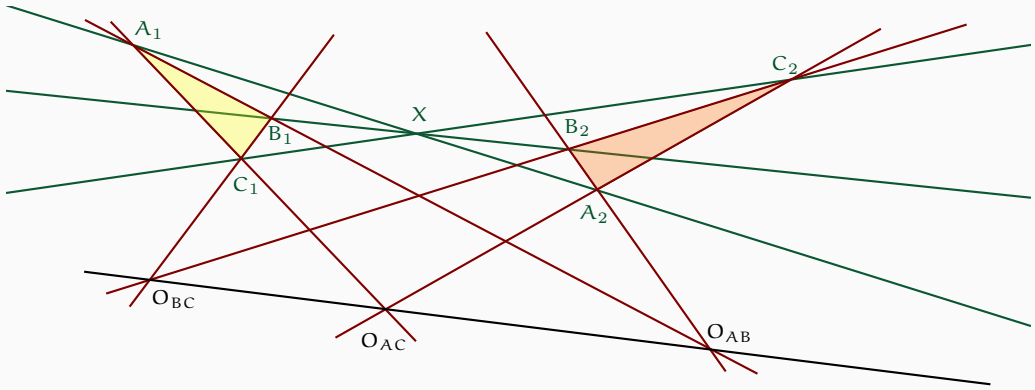
Графът на Petersen, който използвахме тук за илюстрация, е интересен математически обект, който е достатъчно малък, за да бъде възприеман визуално с лекота, и е достатъчно сложен, за да има нетривиални свойства. Графът на Petersen често се използва за илюстрация на различни свойства в теорията на графите. Той е 3-регулярен граф с 10 върха и 15 ребра. Практиката е показала, че много хипотези (недоказани твърдения) в теорията на графите намират лесно опровержение (стига да не са верни, разбира се), ако бъдат "опитани" върху графа на Petersen.
Името на графа идва от датския математик Julius Petersen. Въпреки името, този граф е изследван преди Petersen от английския математик Kempe в контекста на т. нар. конфигурации на Desargues. Desargues е френски математик от 16-17 век, който доказал следната теорема.

Теорема 5: Теорема на Desargues

Нека $A_1B_1C_1$ и $A_2B_2C_2$ са произволни триъгълници. Нека правите A_1B_1 и A_2B_2 се пресичат в точка O_{AB}, правите A_1C_1 и A_2C_2 се пресичат в точка O_{AC} и правите B_1C_1 и B_2C_2 се пресичат в точка O_{BC}. Правите A_1A_2, B_1B_2 и C_1C_2 се пресичат в една и съща точка X тогава и само тогава, когато O_{AB}, O_{AC} и O_{BC} лежат на една и съща права.

Фигура 2.32 илюстрира Теорема 5. На фигурата са показани общо 10 прави: трите зелени прави, върху които лежат върховете на двата триъгълника, шестте кафяви прави, определени от страните на триъгълниците, и черната права, определена от O_{AB}, O_{AC} и O_{BC}. Тези 10 прави се пресичат по тройки в 10 точки, които са именувани.

Фигура 2.32 : Илюстрация на теоремата на Desargues.



Да наречем такава съвкупност от 10 прави и техните 10 пресечни точки по тройки, *конфигурация на Desargues*. Кемре забелязал, че ако в конфигурация на Desargues съпоставим на всяка права един връх и свържем два върха с ребро тогава и само тогава, когато съответните им прави **не се пресичат** в една от десетте пресечни точки, ще получим това, което днес бихме нарекли граф, изоморфен на графа на Petersen[a]. Например, на Фигура 2.32 черната права не се пресича (в една от десетте пресечни точки) с точно три други прави, а именно с трите зелени прави; а в графа на Petersen, върхът, който съответства на синята права, е съсед точно на трите върха, съответни на зелените прави, и така нататък.
За повече подробности около графа на Petersen и връзката с конфигурациите на Desargues вижте книгата на Holton и Sheehan [34].

[a]Секция 2.8 обяснява понятието "изоморфизъм".

Допълнение 6: NP-трудност на $\tau(G)$ и $\gamma(G)$

Като алгоритмични задачи, намирането на минимално върхово покриване и на минимално доминиращо множество са практически нерешими **в общия случай**. Най-добрите известни алгоритми за тях, грубо казано, работят с разглеждане на възможности, чиито брой е експоненциален в броя на върховете, ако не са дадени някакви ограничения върху графите – оттам идва и "в общия случай".
Намирането на $\tau(G)$ и $\gamma(G)$ за произволен граф G са задачи от известния клас на ***NP**-трудните задачи*. Характерното за тези задачи е, че имат огромна практическа важност, за нито една от тях не е известен практично бърз алгоритъм, но и не е доказано, че такъв алгоритъм не съществува. Освен това, всички тези задачи се свеждат бързо една до друга в следния смисъл: ако за една от тях има бърз алгоритъм, то за всяка от тях има бърз алгоритъм, и, обратно, ако за една от тях се докаже, че няма бърз алгоритъм, от това ще следва, че за никоя от тях няма бърз алгоритъм. Най-важното предизвикателство пред съвременната теоретична компютърна наука е отговорът на въпроса дали наистина **NP**-трудните задачи са практически нерешими. Популярното име на това предизвикателство е задачата дали **P = NP**.
Всичко това е доста неформално казано, но този материал е далече извън обхвата на настоящите лекционни записки. Добро въведение в теорията на изчислителната

сложност, част от която е **NP**-трудността, са учебниците “Computers and Intractability” на Garey и Johnson [26] и “Computational Complexity” на Papadimitriou [47].

2.6 Двуделни графи

Определение 33

Нека $G = (V, E)$ е граф. Ако съществува разбиване на V на две подмножества V_1 и V_2, такива че $\forall (u, v) \in E : u \in V_1 \wedge v \in V_2$, казваме, че G е *двуделен граф*. Множествата V_1 и V_2 са *дяловете* на G.

Някои автори като Berge [10] и Golumbic [28] ползват следната нотация за двуделни графи.

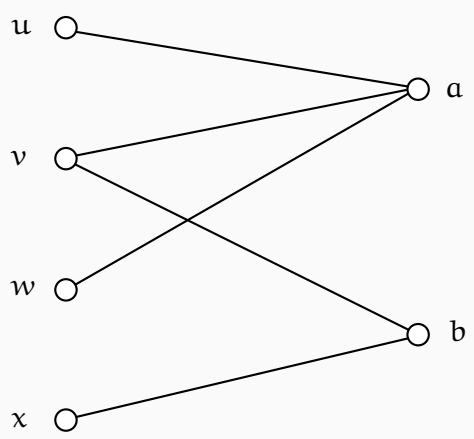
Нотация 3

Ако за граф G напишем "$G = (V_1, V_2, E)$" и е известно или ясно, че V_1 и V_2 са множества от върхове, то искаме да кажем, че G е двуделен граф с дялове V_1 и V_2.

Лесно се вижда, че всеки от дяловете в Определение 33 е антиклика (вижте Подсекция 2.1.8).

Фигура 2.33 показва рисунка на двуделен граф. Типично, двуделните графи се рисуват така: дяловете вляво и вдясно, върховете им разположени вертикално, и ребрата с отсечки между дяловете.

Фигура 2.33 : Двуделен граф.



Единият дял е $\{u, v, w, x\}$, а другият дял е $\{a, b\}$.

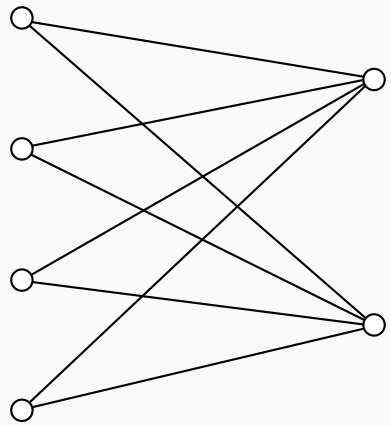
Двуделните графи имат голямо приложение при моделиране на житейски ситуации, в които има два вида обекти, като обектите от първия вид са в някаква релация с обектите от втория.

Определение 33 има малък формален недостатък. Според него, графът, състоящ се от един единствен връх (и без ребра) не е двуделен, защото по определение разбиването става на непразни множества, така че всеки двуделен граф—ако следваме буквата на Определение 33—има поне два върха. Това не е особен проблем, защото така или иначе, интересни са големите графи, но за всеки случай ще додефинираме, че графът с един единствен (изолиран) връх е двуделен, като единият му дял е празен. Но всеки граф с повече от един връх, който е двуделен, има непразни дялове.

Пълен двуделен граф се нарича двуделен граф, такъв че всеки връх от единия дял е съсед на всеки връх от другия дял. Ако дяловете имат мощности p и q, пълният двуделен граф

се бележи с $K_{p,q}$. Фигура 2.34 показва рисунка на $K_{4,2}$. Очевидно, $K_{p,q}$ е същият обект като $K_{q,p}$, тъй като дяловете не са подредени—точно кой от дяловете е нарисуван вляво и кой, вдясно, е въпрос на естетическо предпочитание.

Фигура 2.34 : $K_{4,2}$.

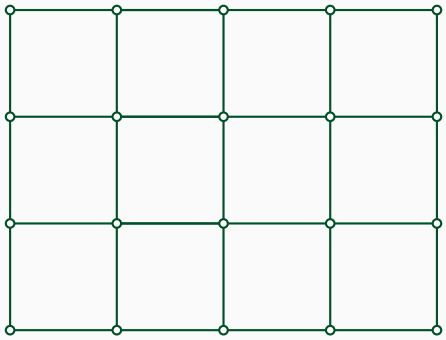


Допълнение 7: Недеформируеми структури и двуделни графи

Добре известно е, че следната задача от строителното инженерство се решава елегантно с използване на двуделни графи. Подробности по нея, както и доста нейни обобщения, може да се намерят в [6].

Преди да формулираме задачата ще обясним за какво става дума. Да дефинираме двумерна структура, която—за целите на тази задача—ще наричаме "структура от греди с размери $p \times q$". Без формална дефиниция: дадени са $(p-1)q + p(q-1)$ еднакви стоманени греди с дължина 1, тънки и дълги, неразтегливи, неподатливи на натиск и неогъващи се. Гредите са наредени в нещо като правоъгълна мрежа с размери p реда на q колони и са захванати с шарнирни[a] връзки в краищата си. Пример с $p = 4$ и $q = 5$ е показан на Фигура 2.35.

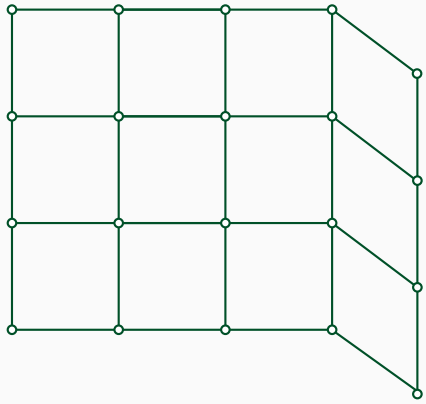
Фигура 2.35 : Структура от греди 4×5.



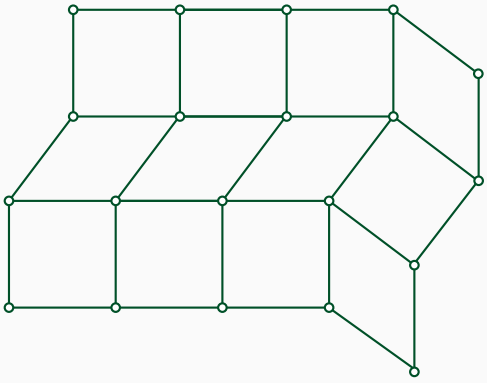
Структурата има 4 реда, 5 колони и общо $(4-1)5 + 4(5-1) = 31$ греди.

Тъй като връзките са шарнирни, структурата не е фиксирана, а е податлива на деформации, ако се приложи външна сила. Например, структурата от Фигура 2.35 може да се деформира до някоя от структурите, показани на Фигура 2.36.

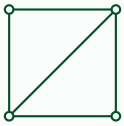
Фигура 2.36 : Деформации на структурата от Фигура 2.35.



Една възможна деформация.



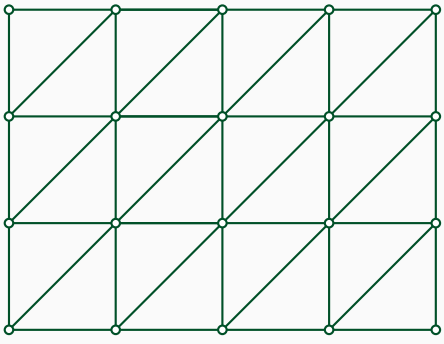
Друга възможна деформация.



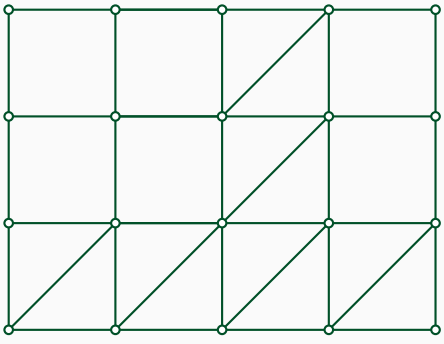
Сега да допуснем, че в допълнение на $\mathtt{p} \cdot \mathtt{q}$ еднаквите греди са дадени и други греди, по-дълги, които може да бъдат поставяни диагонално на "квадратите" на началната структура (преди деформациите). Всички диагонални греди са с една и съща дължина $\sqrt{2}$. Въпреки че диагоналните греди се захващат също с шарнирни връзки, те осигуряват *недеформируемост* на структурата. Най-прост пример за недеформируема структура от греди е един квадрат с поставен диагонал, показан тук. Тъй като триъгълникът е недеформируем дори при шарнирни връзки във върховете, а квадратът с диагонал в някакъв смисъл се състои от два "залепени" триъгълника, то показаната тук структура е недеформируема.

Лесно се вижда, че структурите на Фигура 2.37 са недеформируеми. За други структури е по-трудно да преценим на око дали са деформируеми или не (вж. Фигура 2.38).

Фигура 2.37 : Недеформируеми структури.

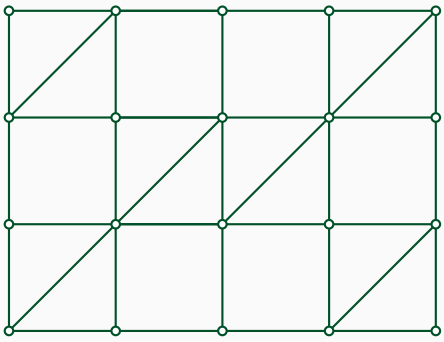


Ако всички във всички квадрати има диагонали, структурата е недеформируема.

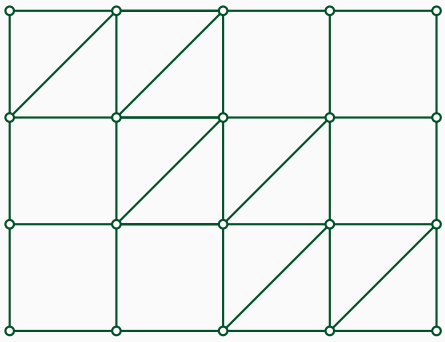


Дори в някои от квадратите да няма диагонали, пак може структурата да е недеформируема.

Фигура 2.38 : Не е очевидно дали тези структури са деформируеми.



Структурата е деформируема.



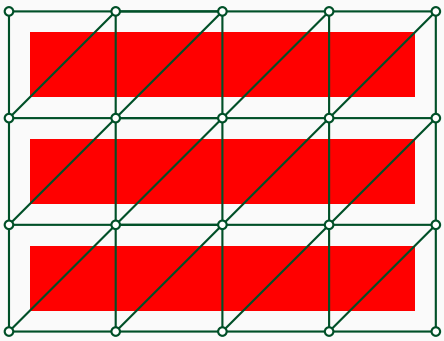
Структурата е недеформируема.

Задача 2

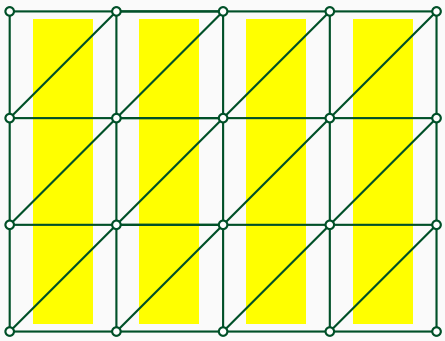
Дадена е структура от греди с диагонали. Да се намери просто необходимо и достатъчно условие, така че структурата да не е деформируема.

Решение: Да дефинираме, че структурата има $\mathsf{p}-1$ *хоризонтала* и $\mathsf{q}-1$ *вертикала*, където под "хоризонтал" се разбират редовете от клетките, които са между гредите (а не самите греди); и под "вертикал" се разбират колоните от клетките, които са между гредите (а не самите греди). Например, на Фигура 2.39 с червено са показани хоризонталите, а с жълто, вертикалите. Гредите на даден хоризонтал са вертикалните греди в него и гредите на всеки вертикал са хоризонталните греди в него.

Фигура 2.39 : Хоризонтали и вертикали на структура.



Структурата от Фиг. 2.35 има 3 хоризонтала, тук оцветени в червено.



Структурата от Фиг. 2.35 има 4 вертикала, тук оцветени в жълто.

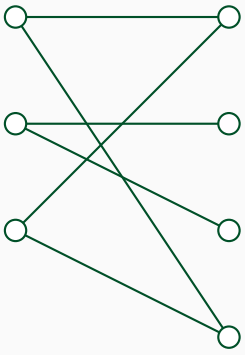
Следните съображения са очевидни:

- Гредите във всеки хоризонтал са успоредни. Това остава в сила дори хоризонталът да да прави "чупки" надолу и нагоре, както например всички хоризонтали на Фигура 2.36.

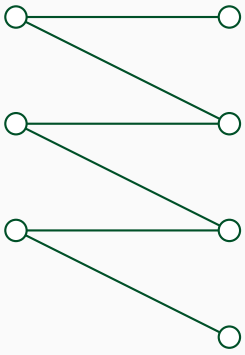
- Аналогично, гредите във всеки вертикал са успоредни. Това остава в сила дори когато вертикалът прави "чупки" наляво и надясно, както например най-десният вертикал на Фигура 2.36 вдясно.
- Поставянето на диагонална греда води до това, че съответният ромб е задължително квадрат.
- За всяка недеформируема структура:
  - гредите във всеки хоризонтал са успоредни на гредите във всеки друг хоризонтал;
  - гредите във всеки вертикал са успоредни на гредите във всеки друг вертикал;
  - гредите на всеки хоризонтал са перпендикулярни на гредите на всеки вертикал.

Да си представим двуделен граф $G = (V, E)$ с дялове V_1 и V_2, такъв че V_1 има $p-1$ върха, а V_2 има $q-1$ върха. Нещо повече, върховете от V_1 съответстват на хоризонталите, а тези от V_2, на вертикалите. За всеки два върха $u \in V_1$ и $v \in V_2$ има ребро (u, v) тогава и само тогава, когато в пресечната клетка на u-ия хоризонтал и v-ия вертикал в структурата има поне една диагонална греда. Примери за такива графи има на Фигура 2.40. Несвързаният граф на Фигура 2.40 вляво отговаря на деформируемата структура на Фигура 2.38 вляво. Свързаният граф на Фигура 2.40 вдясно отговаря на деформируемата структура на Фигура 2.38 вдясно.

Фигура 2.40 : Двуделни графи, съответстващи на структури от греди.



Несвързан граф съответства на деформируема структура.



Свързан граф съответства на недеформируема структура.

Решение на задачата, която разглеждаме, се дава от Теорема 6. Тя е взета от книгата на Baglivo и Graver [6, Theorem 10.5, pp. 81], но доказателството ѝ не е тривиално и тук ще го прескочим.

Теорема 6: Необходимо и достатъчно условие за недеформиреумост.

Дадена структура от греди е недеформируема тогава и само тогава, когато съответният двуделен граф е свързан. □

[a]От френската дума *charnière*, която означава панта или става.

2.7 Оцветявания на графи

Под "оцветяване на графи" може да се разбира едно от следните две неща: оцветяване на върхове или оцветяване на ребра. Тъй като първото е много по-често срещано, ако кажем само "оцветяване на графи" ще разбираме "оцветяване на върхове".

2.7.1 Оцветяване на върхове

Допълнение 8: История на задачата за оцветяването

Тази графова задача има интересна история. Широко разпространената версия—да я наречем "практичната версия"—свърза произхода на задачата с изработването на административни карти през 19 век. Административната карта, както подсказва името, се фокусира върху административното делене на някаква територия на някакви области, игнорирайки релефа и други подробности. За по-ясно възприемане, всяка област се оцветява в един цвят, а нейните съседи, в различни, достатъчно контрастиращи цветове.

Забележка 1

Определението на "съседни области" е: "такива, които имат обща граница, която не е просто точка". Очевидно, ако допускаме съседни области да са и такива, които имат само обща точка, то може да има произволно много области, които са взаимно съседни и това да налага използването на произволно много цветове. Пример за реални области, имащи само обща точка, са щатите Юта, Ню Мексико, Аризона и Колорадо в САЩ. По нашето определение на "съседство", нито два от тях не са съседни.

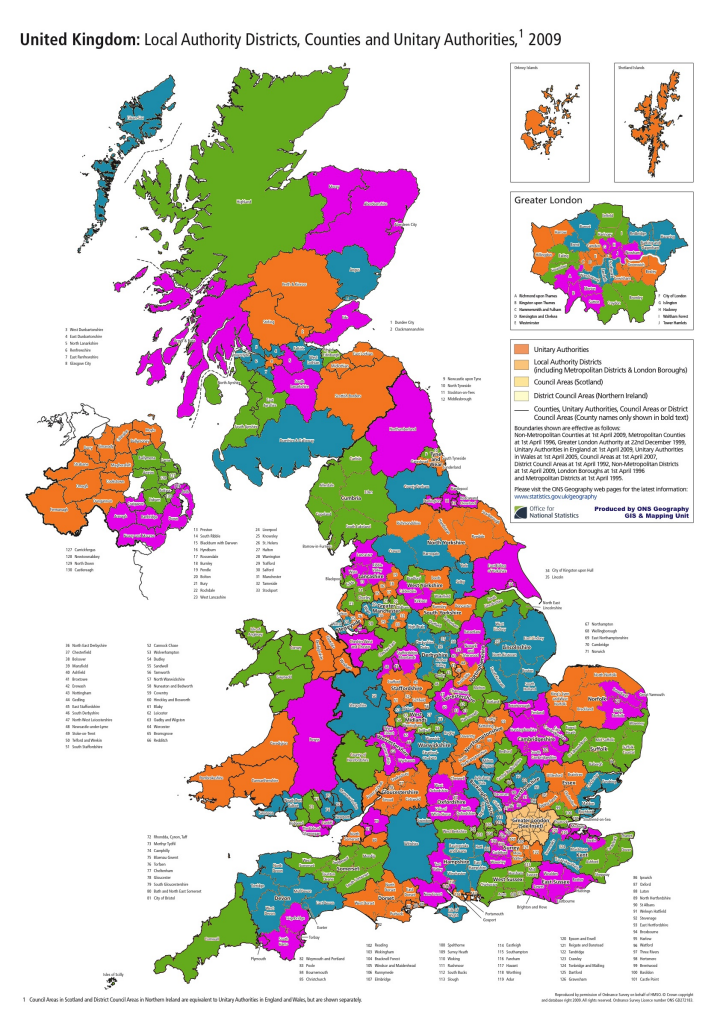
И така, през 19 век английските картографи, правейки административни карти, установили, че 4 цвята са необходими за оцветяване на някои карти и достатъчни за оцветяването на всяка известна карта.

Забележка 2

Лесно може да се даде пример за карта, която иска 4 цвята: представете си една област, обградена от "цикъл" от нечетен брой области, като началната област е съседна на всяка област от "цикъла", а областите от "цикъла" са съседни по двойки по протежение на цикъла.

Примерна, хипотетична административна карта на Обединеното Кралство е показан на Фигура 2.41. Картата илюстрира едно възможно делене на Обединеното Кралство на райони за гласуване.

Фигура 2.41 : Хипотетична административна карта на Обединеното Кралство.



Картата използва 4 цвята: магента, син, зелен и оранжев, като всички двойки съседни райони са в различни цветове. Територията на Лондон е в бежово, защото тамошните териториални единици са прекалено малки по площ. Те са показани уголемени вдясно, оцветени отново в основните 4 цвята. Картата показва възможно териториално разделяне на UK с цел по-справедливо разпределяне на парламентарните места при гласуване и е взета от *сайта на Heriot-Watt University*.

По онова време използването на цветове при печатането било лукс. Колкото повече цветове се използвали, толкова оскъпяването било по-голямо, което мотивирало използването на минимален брой цветове. Картографите започнали да се питат, дали има карта, която да иска повече от 4 цвята, или 4 цвята стигат за всяка карта. След малко ще обясним каква е връзката между оцветяването на карти и графите.

Другата версия—може да я наречем "академичната версия"—за възникването на задачата за оцветяване на карти е описана в книгата на Rousse Ball [7]. Според нея, задачата първоначално била спомената в лекция на Möbius през 1840 г., но истинският интерес към нея започнал през 1850 г. след запитване на един студент към професор

De Morgan дали е вярно, че 4 цвята винаги стигат; De Morgan не можел да отговори и се обърнал към прочутия Hamilton, който също не можал да отговори. С времето задачата достигнала до математици като Cayley и Peirce и започнала да интригува математическата общност. Чак през 1976 г. било доказано, че 4 цвята наистина са достатъчни. Доказателството станало известно като *Теоремата за четирите цвята* на Appel и Haken (вж. [4], [5]), които го редуцирали до систематично разглеждане на 1936 конфигурации. Разглеждането на всички тези случаи било направено с компютър и това бил първият важен резултат в математиката, в чието доказателство се използва резултат от работата на компютър.

Теорема 7: Теорема за четирите цвята, формулировка чрез карти

Всяка административна карта може да бъде оцветена с не повече от 4 цвята. □

Да се върнем на оцветяването на административни карти, с което започнахме. Задачата да бъде оцветена административна карта по указания начин може да бъде формулирана и чрез графи. Можем да съпоставим граф на картата, където върховете отговарят на районите, а ребро между два върха се слага тогава и само тогава, когато съответните райони са съседни.

Забележка 3

Полученият граф е *планарен граф*. Конверсно, на всеки планарен граф съответства поне една географска карта, чиито райони отговарят на върховете, а съседствата между районите, на ребрата. Не всеки граф е планарен. Съществуват не-планарни графи, които не може да бъдат оцветени с 4 цвята. Повече информация за планарните графи има в Секция 2.13.

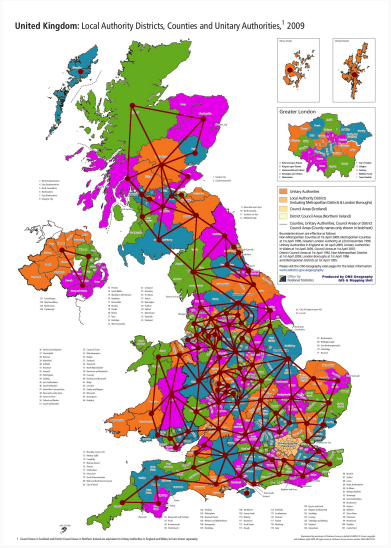
Да бъде оцветена картата с не повече от 4 цвята е същото като да бъде оцветен—съгласно Определение 34—съответният планарен граф с не повече от 4 цвята.

Теорема 8: Теорема за четирите цвята, формулировка чрез графи

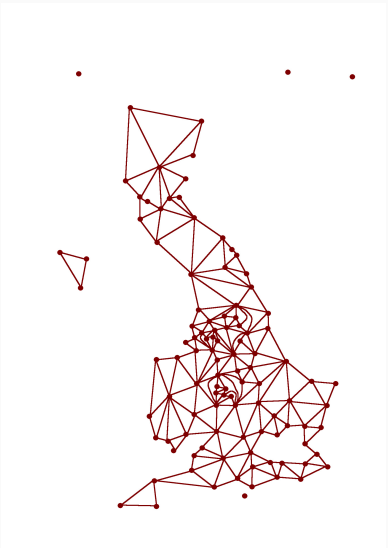
За всеки планарен граф G, $\chi(G) \leqslant 4$. □

На Фигура 2.42 вдясно е показан графът на съседствата, който се получава от картата на Фигура 2.41. На Фигура 2.42 вляво е показан междинен етап от получаването на този граф.

Фигура 2.42 : Получаването на графа, съответстващ на картата от Фигура 2.41.



Лондон с неговото териториално разделение е игнориран.



Това е самият граф.

Тривиално е да се съобрази, че графът на Фигура 2.42 е 4-оцветим: това е очевидно, щом съответната му карта е оцветена с 4 цвята.

Определение 34: Оцветяване на върхове

Нека $G = (V, E)$ е граф. *Оцветяване на върховете на* G, или просто *оцветяване на* G, е функция $f : V \to C$, където C е множество, чиито елементи се наричат *цветове*. Иска се функцията да е такава, че $\forall (u, v) \in E : f(u) \neq f(v)$.

Има смисъл оцветяването да е сюрекция, за да няма неизползвани цветове.

Като алгоритмична задача, задачата се дефинира за произволни графи, не само за планарни, и е минимизационна—трудно е графът да бъде оцветен с малко цветове. Оцветяване с n цвята би било тривиално, но безинтересно. Лесно се вижда, че всяко множество от върхове от един и същи цвят е антиклика. Така че задачата всъщност е, да се разбие V на минимален брой антиклики.

Определение 35: Хроматично число.

Минималният брой цветове, с които може да бъде оцветен даден граф G, се нарича *хроматичното число на* G. Хроматичното число на G се бележи с $\chi(G)$.

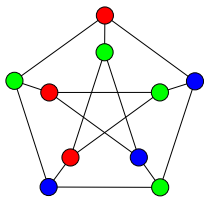
Оцветяването на върховете се появява в практически приложения, в които графът моделира несъвместимост между някакви обекти, като несъвместимостта е симетрична. Върховете отговарят на обектите, а ребро между два върха се поставя тогава и само тогава, когато съответните обекти са несъвместими. По отношение на реалната ситуация може да ни интересуват две неща:

- да намерим максимален брой обекти, които са съвместими по двойки; в термините на графите, това е задачата за **намиране на максимална антиклика**.

- да намерим разбиване на множеството от обектите на минимален брой подмножества, такива, че във всяко от тях обектите да са съвместими по двойки; в термините на графите, това е задачата за **разбиване на множеството от върховете на минимален брой антиклики**.

Втората задача се формулира с цветове: върховете от всяка антиклика са оцветени в един и същи цвят, който е различен от цветовете на върховете от останалите антиклики.

Ето пример за такава задача, при която графът моделира симетрична несъвместимост. Дадено е някакво помещение за работа. Да кажем, цех. Дадени са някакви дейности $A_1, \ldots, A_n$, които може да се вършат в този цех, като всяка дейност си има свое място, на което може да се върши, и там има машини, специфични за нея. Да кажем, че всяка дейност се върши за единица време. Ние искаме да извършим всички дейности за минимално общо време. Всички дейности биха могли да се вършат едновременно, тъй като има достатъчно място (което би значело да загубим само единица време за всички дейности), но има едно важно ограничение: някои двойки дейности са несъвместими и не могат да се вършат едновременно. Например, шлайфане, което отделя прах, и боядисване, което иска да няма прах във въздуха. Заради наличието на такива несъвместимости не може да извършим всички дейности наведнъж. Нека съответният граф, моделиращ несъвместимостта, е G. Ако се интересуваме какъв е максималният брой дейности, които може да извършим наведнъж, то ние всъщност питаме какво е числото на независимостта на G, което означаваме с $\alpha(G)$ (вижте Определение 11). Ако се интересуваме в колко единици време най-малко може да бъдат извършени всички дейности, то ние всъщност питаме какво е хроматичното число на G, което означаваме с $\chi(G)$. Заслужава да се отбележи, че и намирането на $\alpha(G)$, и намирането на $\chi(G)$, са **NP**-трудни задачи за графи без ограничения. Понятието "**NP**-трудност" е дефинирано неформално в Допълнение 6.



Фигура 2.43

Пример за оцветяване на граф е показан на Фигура 2.43 вляво. На фигурата е изобразен графът на Petersen, чиито върхове са оцветени в три цвята: син, червен и зелен. Това е оптимално оцветяване за графа на Petersen, следователно той има хроматично число 3. Ще докажем, че оцветяването е оптимално в смисъл, че графът на Petersen не може да бъде оцветен с по-малко цветове. Забелязваме, че графът на Petersen има нечетен цикъл: например, външният контур на нарисувания граф е цикъл с дължина 5. От друга страна, от Теорема 9 на тази страница следва, че ако граф има поне един нечетен цикъл, то хроматичното му число не е 2. Следователно, хроматичното число на графа на Petersen не е 2. Очевидно хроматичното му число не е 1. Следователно, хроматичното му число е поне 3 и показаното оцветяване е оптимално като брой цветове.

Веднага се вижда, че $\chi(G) = 1$ тогава и само тогава, когато G е празният граф—наличието на поне едно ребро влече $\chi(G) \geqslant 2$. В сила е следният любопитен резултат.

Теорема 9: Граф е 2-оцветим тстк няма нечетни цикли

Нека $G = (V, E)$ е непразен граф. $\chi(G) = 2$ тогава и само тогава, когато G няма нечетни цикли.

Без ограничение на общността, нека G е свързан.

Доказателство, I: Да допуснем, че $\chi(G) = 2$. Нека цветовете са бял и черен и функцията на оцветяването е f. С други думи, $f : V \to \{\text{бял}, \text{черен}\}$. Ще докажем, че G няма нечетни цикли. Да допуснем, че G има поне един нечетен цикъл c. Разглеждаме произволен връх u от c. Без ограничение на общността, нека $f(u) = \text{бял}$. Веднага се вижда, че съседите на u в цикъла са черни, техните съседи в цикъла (на разстояние 2 от u цикъла) са бели, и така нататък, като максимално отдалечените от u в цикъла върхове са точно два и са оцветени в един и същи цвят; и, естествено, са съседи, бидейки върхове от цикъла, което противоречи на 2-оцветимостта.

Това не е прецизно доказателство. Може да го формализираме по следния начин. За всеки нечетен цикъл s, за всеки връх x в s, е вярно, че има точно два максимално отдалечени от x върха (в s), да ги наречем y и z, като $\text{dist}_s(x, y) = \text{dist}_s(x, z) = \left\lfloor \frac{|s|}{2} \right\rfloor$. Без ограничение на общността, нека $f(x) = \text{бял}$. За всеки връх b от цикъла ще докажем по индукция по $\text{dist}_s(x, b)$ следното твърдение:

- $f(b) = \text{бял}$, ако $\text{dist}_s(x, b)$ е четно, и
- $f(b) = \text{черен}$, ако $\text{dist}_s(x, b)$ е нечетно.

където $\text{dist}_s(x, b)$ взема стойности от крайното множество $\left\{0, 1, \ldots, \left\lfloor \frac{|s|}{2} \right\rfloor\right\}$. Базата на индукцията е за $\text{dist}_s(x, b) = 0$, тоест за $b = x$: твърдението е вярно. Допускаме, че твърдението е вярно за всеки връх b, такъв че $0 \leqslant \text{dist}(x, b) < \left\lfloor \frac{|s|}{2} \right\rfloor$ (има точно два такива върха) и забелязваме, че твърдението остава вярно за b', такъв че $\text{dist}(x, b') = \text{dist}(x, b) + 1$. След като сме доказали твърдението, веднага виждаме, че $f(y) = f(z)$, независимо дали разстоянието в цикъла между кой да е от тях и x е четно или нечетно. Но y и z са съседи и не може да бъдат оцветени в един и същи цвят. Изведеното противоречие опровергава последното направено допускане, а именно, че в G има поне един нечетен цикъл.

Доказателство, II: Да допуснем, че няма нечетни цикли. Ще докажем конструктивно, че $\chi(G) = 2$. Нека x е произволен връх от G. Дефинираме функцията на оцветяването по следния начин:

$$\forall v \in V : f(v) = \begin{cases} \text{бял}, & \text{ако } \text{dist}(x, v) \text{ е четно} \\ \text{черен}, & \text{ако } \text{dist}(x, v) \text{ е нечетно} \end{cases}$$

Разбира се, дефинирането на функцията не е достатъчно. Трябва да докажем, че тя реализира именно оцветяване на графа. С други думи, трябва да докажем, че няма ребро, двата края на което са в един и същи цвят.

Да допуснем, че има ребро $e \in E$, такова че $e = (y, z)$ и $f(y) = f(z)$. Без ограничение на общността, нека $f(y) = f(z) = \text{бял}$, тоест $\text{dist}(x, y)$ е четно и $\text{dist}(x, z)$ е четно. Нека $\text{dist}(x, y) = 2s$ за някое $s \in \mathbb{N}^+$ и $\text{dist}(x, z) = 2t$ за някое $t \in \mathbb{N}^+$. Веднага се вижда, че $2s \leqslant 2t + 1$, защото от факта, че има път с дължина $2t$ между x и z и факта, че има ребро между z и y, следва, че има път с дължина $\leqslant 2t + 1$ между x и y. Аналогично, $2t \leqslant 2s + 1$. И така:

$$2s \leqslant 2t + 1$$
$$2t \leqslant 2s + 1$$

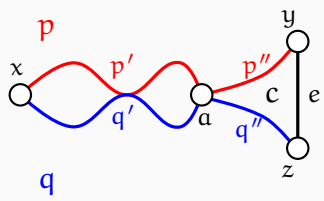
Тогава $2t - 1 \leqslant 2s \leqslant 2t + 1 \leftrightarrow t - \frac{1}{2} \leqslant s \leqslant t + \frac{1}{2}$. Но s и t са цели числа, така че $s = t$. Излиза, че y и z са равноотдалечени от x.

Нека p е произволен x-y път с мининална дължина и q е произволен x-z път с мининална дължина. Както видяхме, $|p| = |q|$. Единият край на тези два пътя съвпада (връх x), но те може да имат и други общи върхове освен x. Нека връх a е най-отдалеченият от x връх, който е общ за p и q. Нека подпътят на p между x и a включително е p' и подпътят на p между a и y включително е p''. Нека подпътят на q между x и a включително е q' и подпътят на q между a и z включително е q''. Това означава, че $p = p' \cup p''$ и $q = q' \cup q''$. Фигура 2.44(а) илюстрира пътищата p и q заедно с реброто $e = (y, z)$.

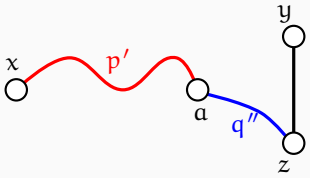
Твърдим, че $|p'| = |q'|$. Да допуснем, че $|p'| \neq |q'|$. Без ограничение на общността, нека $|p'| < |q'|$. Тогава пътят $p' \cup q''$ свързва x със z и е по-къс от q; това противоречи на направеното по-рано допускане, че q е най-къс път между x и z. Фигура 2.44(б) дава нагледно доказателство, че $|p'| = |q'|$: ако не беше така, тоест ако $|p'| < |q'|$, заместването на q' с p' в q би дало по-къс между x и z от q.

Доказахме, че $|p'| = |q'|$. Освен това знаем, че $|p| = |q|$ и че $|p| = |p'|+|p''|$ и $|q| = |q'|+|q''|$. Следователно, $|p''| = |q''|$. Нека $|p''| = k$. Тогава $|q''| = k$. Нека r означава пътя y, e, z. Веднага виждаме, че $|c| = 2k + 1$, където $c = p'' \cup q'' \cup r$ (вж. Фигура 2.44(в)). Но това означава, че c е нечетен цикъл, което противоречи на първоначалното допускане, че в G няма нечетни цикли.

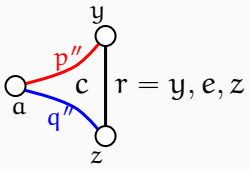
Фигура 2.44 : Илюстрация на доказателството на Теорема 9.



Фигура 2.44(а)



Фигура 2.44(б)



Фигура 2.44(в)

□

Абсолютно очевидно е, че наличието на k-клика в графа G влече $\chi(G) \geqslant k$, защото самото клика иска k цвята за своето оцветяване. Затова следното твърдение е наречено "наблюдение".

Наблюдение 16

$\omega(G) \leqslant \chi(G)$, за всеки граф G.

Лесно може да измислим примери за графи, чието кликово число е строго по-малко от хроматичното число. Например, нека G е граф-цикъл с 5 върха. Тогава $\omega(G) = 2$, защото 3-клика няма, но $\chi(G) = 3$, защото има нечетен цикъл (припомняме си Теорема 9).

Изследователите на графи са нарекли свойството, кликовото число да е равно на хроматичното, *перфектност*. Но определението на "перфектен граф" е по-сложно от изискването $\omega(G) = \chi(G)$. Всеки граф може да бъде модифициран чрез добавяне на достатъчно голяма клика, така че кликовото и хроматичното число да се изравнят. Перфектността на графите изисква всяка част от графа да има определена структура. Това налага следното определение.

Определение 36: Перфектен граф.

Граф G е *перфектен*, ако за всеки индуциран подграф G' на G е вярно, че $\omega(G') = \chi(G')$.

Перфектните графи са интересни както от теоретична, така и от практическа гледна точка. Примерно, задачите за намиране на $\alpha(G)$, $\chi(G)$ и $\omega(G)$ са бързо решими, ако G е перфектен, въпреки че са **NP**-трудни (вижте Допълнение 6) в общия случай. Перфектните графи са въведени от великия изследовател на теорията на графите Claude Berge [11]. Примери за перфектни графи ще видим следващи секции.

Теорема 10: $\chi(G) \cdot \alpha(G) \geqslant n$

За всеки граф G, $\chi(G) \cdot \alpha(G) \geqslant n$.

Доказателство: Както вече отбелязахме, оцветяването на граф G с k цвята е намиране на разбиване $\{V_1, V_2, \ldots, V_k\}$ на $V(G)$, такова че всяко от $V_1, \ldots, V_k$ е антиклика. Тогава всяко оптимално оцветяване на G, което става с $\chi(G)$ цвята, е разбиване $\{V_1, V_2, \ldots, V_{\chi(G)}\}$ на $V(G)$, като всяко от тези множества е антиклика. Но тогава $|V_i| \leqslant \alpha(G)$ за $1 \leqslant i \leqslant \chi(G)$, тъй като по определение не може да има антиклика, по-голяма от $\alpha(G)$. Тогава $\sum_{i=1}^{\chi(G)} |V_i| \leqslant \chi(G) \cdot \alpha(G)$. Но по принципа на разбиването, $\sum_{i=1}^{\chi(G)} |V_i| = n$. Желаният резултат следва веднага. □

Теорема 11 е лесно следствие от Теорема 10, Наблюдение 16 и Следствие 2:

$$\omega(G) = \alpha(\overline{G}) \quad // \text{ от Следствие 2} \tag{2.4}$$
$$\omega(\overline{G}) = \alpha(G) \quad // \text{ Наблюдение 4 и (2.4)} \tag{2.5}$$
$$\omega(\overline{G}) \leqslant \chi(\overline{G}) \quad // \text{ Наблюдение 16} \tag{2.6}$$
$$\alpha(G) \leqslant \chi(\overline{G}) \quad // \text{ (2.5) и (2.6)} \tag{2.7}$$
$$\chi(G) \cdot \alpha(G) \geqslant n \quad // \text{ Теорема 10} \tag{2.8}$$
$$\chi(G) \cdot \chi(\overline{G}) \geqslant n \quad // \text{ (2.7) и (2.8)} \tag{2.9}$$

Но Теорема 11 има и самостоятелно интересно доказателство, което ще направим подробно.

Теорема 11: $\chi(G) \cdot \chi(\overline{G}) \geqslant n$

За всеки граф $G = (V, E)$, $\chi(G) \cdot \chi(\overline{G}) \geqslant n$.

Доказателство: Нека $k = \chi(G)$ и $\ell = \chi(\overline{G})$. Ще покажем, че $k\ell \geqslant n$. Щом $k = \chi(G)$ и $\ell = \chi(\overline{G})$, то съществуват множества C и D, такива че $|C| = k$ и $|D| = \ell$ и оцветявания $f : V \to C$ и $g : V \to D$ съответно на G и $\overline{G}$.

Нека E' е множеството от ребрата на $\overline{G}$. Също както в доказателството на Теорема 1, конструираме графа $\widetilde{G} = (V, \widetilde{E})$, където $\widetilde{E} = E \cup E'$. Очевидно $\widetilde{G}$ е пълен граф на n върха, но е именуван граф (вижте Подсекция 2.8.2). Да разгледаме $h : V \to C \times D$, дефинирана така:

$$\forall u \in V : h(u) = (f(u), g(u))$$

Твърдим, че h е оцветяване на $\widetilde{G}$ съгласно Определение 34. Да разгледаме произволно ребро $e = (u, v)$ на $\widetilde{G}$ (което е същото като да разгледаме произволно двуелементно подмножество $\{u, v\}$ на V).

- Ако $e \in E$, то $h(u) \neq h(v)$, понеже $f(u) \neq f(v)$, а $h(u) = (f(u), g(u))$ и $h(v) = (f(v), g(v))$.
- Ако $e \in E'$, то $h(u) \neq h(v)$, понеже $g(u) \neq g(v)$, а $h(u) = (f(u), g(u))$ и $h(v) = (f(v), g(v))$.

Щом краищата на всяко ребро на $\widetilde{G}$ получават различни цветове от h, то h е оцветяване на $\widetilde{G}$. Нещо повече, h е оцветяване на $\widetilde{G}$ в $k\ell$ цвята, защото $|C \times D| = |C| \cdot |D|$ съгласно комбинаторния принцип на умножението.

Но $\widetilde{G}$ е пълен граф на n върха, откъдето веднага следва, че $\chi(\widetilde{G}) = n$. Това влече, че $k\ell \geqslant n$, понеже, ако допуснем, че $k\ell < n$, то чрез h ние сме конструирали оцветяване на пълен граф на n върха с по-малко от n цвята, което е очевидно невъзможно. □

Лесно се забелязва, че съществува следната връзка между "двуделността" на графите и оцветяването на графите.

Наблюдение 17

За всеки граф G, G е двуделен тогава и само тогава, когато $\chi(G) \leqslant 2$.

Теорема 12: $\chi(G) \leqslant \Delta(G) + 1$

За всеки граф G, $\chi(G) \leqslant \Delta(G) + 1$.

Доказателство: Ще покажем алгоритъм, който оцветява G с не повече от $\Delta(G) + 1$ цвята.

Алгоритъм 1: Алчно Оцветяване

Вход: граф $G = (\{v_1, v_2, \ldots, v_n\}, E)$ и цветове C_1, …, $C_{\Delta(G)+1}$.
Изход: Оцветяване на G с не повече от $\Delta(G) + 1$ цвята.

❶ Оцветяваме v_1 в цвят C_1 и присвояваме $i \leftarrow 2$.

❷ Ако $i > n$, край.

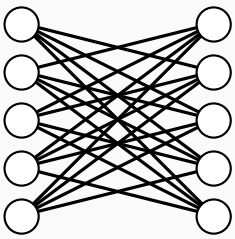
❸ В противен случай, нека C' е цветът с най-малък номер, такъв че за всеки $v_j \in N(v_i)$, такъв че $j < i$ (което означава, че v_j вече е оцветен от нашия алгоритъм) цветът на v_j не е C'. Такъв цвят C' очевидно съществува, защото $|N(v_i)| \leqslant \Delta(G)$. Оцветяваме v_i в цвят C', увеличаваме i с единица и отиваме на ❷.

Аргументацията, че такъв "свободен" цвят C' има за всеки пореден връх, е доказателство за коректността на **Алгоритъм 1**. □

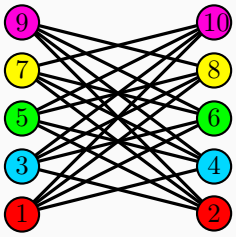
Алчно Оцветяване не е оптимален, тъй като може да използва повече цветове, отколкото са необходими. Наистина, да разгледаме графа G, показан на Фигура 2.45(а). Този граф е двуделен[†] и е 2-оцветим, съгласно Наблюдение 17. Ако приложим Алчно Оцветяване с подредба на върховете като на Фигура 2.45(б), ще използваме 5 цвята. Ако приложим Алчно Оцветяване с подредба на върховете като на Фигура 2.45(в), използваме само 2 цвята, което е оптимално.

[†]Граф като този на Фигура 2.45(а) се нарича граф-корона. Всеки граф-корона се получава от един пълен двуделен граф с равномощни дялове, от който са изтрити ребрата на едно перфектно съчетание. "Перфектно съчетание" е дефинирано в Секция 2.16.

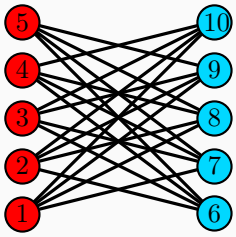
Фигура 2.45 : Алчно Оцветяване връща оцветявания с различна мощност при различни наредби на върховете.



Фигура 2.45(а): Граф-корона.



Фигура 2.45(б): 5-оцветяване.



Фигура 2.45(в): 2-оцветяване.

Допълнение 9: Алчни алгоритми

Това, което казахме за не-оптималността на Алчно Оцветяване върху графа-корона от Фигура 2.45(а) може да бъде обобщено за графи с произволна големина. Да си представим граф-корона, който има $2n$ върха (тогава всеки дял има точно n върха). В замисимост от реда, в който Алчно Оцветяване разглежда върховете, е възможно той да върне оцветяване с n цвята, докато всъщност има оптимално оцветяване със само 2 цвята (защото графът си остава двуделен).

Следователно, съществуват графи и наредби на върховете им, за които Алчно Оцветяване работи ужасно. "Ужасно" в смисъл, че резултатът, който намира, е много далече от оптималния—например, оцветяване с n цвята на граф, който е 2-оцветим. Алгоритъмът работи бързо, но освен бързината му, за нас има значение колко близо до оптималното е намереното решение.

Алгоритъм като Алчно Оцветяване се нарича *алчен алгоритъм*[а], което обяснява и избора на името му. Алчните алгоритми решават оптимизационни задачи правейки итеративно поредица от избирания, като критерият за избиране е прост и локално-оптимален. "Локално-оптимален" означава оптимален само от гледна точка на текущата итерация[б]. В случая с Алчно Оцветяване алгоритъмът избира произволен следващ връх v_i. Произволният избор е възможно най-простият избор, но в някакъв тривиален смисъл е оптимален избор, така че алгоритъмът действително може да се класифицира като алчен.

Както видяхме, алчният алгоритъм не решава задачата оптимално винаги. Задачата за оптимално оцветяване на графи е изключително трудна и не е изненада, че избирането на произволен следващ елемент води това, че върху някои примери намереният резултат е много далече от оптималния. Има задачи, които се решават оптимално с алчни алгоритми, но задачата за оцветяване на граф с минимален брой цветове не е от тях.

[а] На английски е *greedy algorithm*.

[б] По-изтънчената, но по-сложна за реализация идея всички тези избирания да се правят от единна глобална гледна точка.

2.7.2 Оцветяване на ребра

Да си представим комуникационна мрежа, която се състои от компютри и двупосочни (симетрични) връзки между някои двойки компютри. Двойка компютри, свързани директно, е съседна. Разглеждаме дискретно време от времеви интервали с дължина единица. В рамките на един времеви интервал, даден компютър комуникира **с най-много един свой съсед**. При това комуникацията е двупосочна. Искаме да направим разписание[†], при което се осъществяват всички комуникации в минимален брой времеви интервали. Без последното изискване задачата е тривиална: ако директните връзки са m на брой, m времеви интервала са достатъчни, за да може всяка съседна двойка да комуникира. Работата е там, че някои двойки може да комуникират едновременно: в един и същи времеви интервал. За да стане това, необходимо и достатъчно е нито две от тези двойки да нямат общ елемент (компютър).

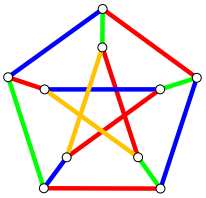
Задачата може да се моделира с граф. Върховете са компютрите, а ребрата са директните комуникационни линии между компютри. Искаме да разбием множеството от ребрата на **минимален брой** подмножества, такива че в нито едно подмножество да няма инцидентни ребра. Можем да мислим за тези подмножества като за "цветове", но сега цветове на ребрата, а не на върховете. Множеството от цветовете отговаря точно на интервалите от време: в първия интервал ще разрешим комуникация по линиите, които са оцветени в първия цвят; във втория интервал ще разрешим комуникациите по линиите, оцветени във втория цвят, и така нататък. Подредбата на цветовете няма никакво значение. Важното е, че ребрата от един и същи цвят точно отговарят на комуникациите, които са разрешени в съответния интервал от време. Тъй като няма инцидентни ребра от един и същи цвят, то няма как в това разписание да има интервал от време, в който някой компютър да комуникира с повече от един съседи.

Определение 37: Оцветяване на ребра

Нека $G = (V, E)$ е граф. *Оцветяване на ребрата на* G е функция $h : E \to C$, където C е множество, чиито елементи се наричат *цветове*. Иска се функцията да е такава, че $\forall e \in E\ \forall e' \in \mathcal{I}(e) : h(e) \neq h(e')$.

Има смисъл оцветяването на ребра да е сюрекция, за да няма неизползвани цветове.

Като алгоритмична задача, оцветяването на ребрата също е минимизационна задача. Минималният брой цветове, с които можем да оцветим ребрата на графа G, се бележи с $\chi'(G)$ и се нарича *хроматичен индекс на* G.



Фигура 2.46

При оцветяването на върхове, очевидна долна граница за хроматичното число е кликовото число (Наблюдение 16). При оцветяването на ребрата, очевидна долна граница за хроматичния индекс е максималната степен на връх $\Delta(G)$. Наистина, ако има поне един връх от степен d, трябва хроматичният индекс да е поне d. Лесно можем да измислим пример за граф G, такъв че $\chi'(G) = \Delta(G)$; например, четен граф-цикъл. От друга страна, лесно можем да измислим и пример за граф G, такъв че $\chi'(G) > \Delta(G)$; например, нечетен граф-цикъл. Графът на Petersen също е пример за такъв граф: той е 3-регулярен граф, но, съгласно Теорема 13, ребрата му не може да бъдат оцветени в 3 цвята по изискванията на Определение 37. Обаче 4 цвята са достатъчни за ребрата на графа на Petersen, както се вижда на Фигура 2.46 вляво. Ерго, хроматичният индекс на графа на Petersen е 4.

[†]На английски, *schedule*.

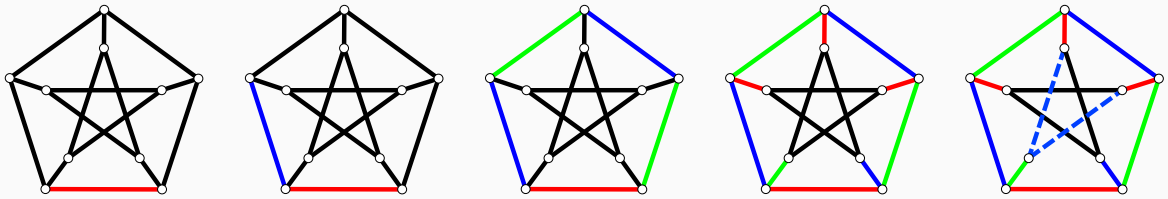
Теорема 13: Графът на Petersen има хроматичен индекс 4.

Графът на Petersen има хроматичен индекс 4.

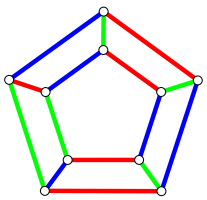
Доказателство: Да допуснем, че графът на Petersen има хроматичен индекс 3. Нека цветовете са червен, зелен и син. Да си представим оцветяването на ребрата с трите цвята като процес, който е показан на Фигура 2.47 отляво надясно.

Да разгледаме един от циклите на графа. Нека това е цикълът, който е нарисуван като външен. Тъй като той има дължина 5, той е нечетен и само за неговото реброво оцветяване са необходими 3 цвята. Лесно се вижда, че точно един от цветовете се среща само веднъж върху този цикъл, а другите два цвята се срещат по два пъти. Без ограничение на общността, нека цветът, който се среща върху точно едно ребро, е червен (което значи, че от останалите четири ребра, две са сини и две са зелени). Без ограничение на общността, червеното ребро да е реброто най-долу от външния цикъл. Без ограничение на общността, нека реброто вляво от него е синьо. Това форсира оцветяване на външния цикъл, показано в средата на фигурата. Това пък форсира оцветяването на петте "радиални" ребра. Сега се налага да оцветим в синьо и двете ребра, показани със син пунктир в десния край на фигурата, защото всяко от тях е инцидентно с едно зелено и едно червено ребро. Но тези две ребра са инцидентни! Това опровергава допускането, че графът на Petersen има хроматичен индекс 3.

Фигура 2.47 : Невъзможно е да оцветим ребрата на графа на Petersen в 3 цвята.



От друга страна, графът на Petersen може да бъде реброво оцветен в 4 цвята, както се вижда на Фигура 2.46. Тогава Хроматичният индекс на графа на Petersen е точно 4. □



Фигура 2.48

Заслужава да се отбележи, че графът, показан на Фигура 2.48 вляво, който много прилича на графа на Petersen, има хроматичен индекс 3. Това се вижда веднага на фигурата. И този граф, и графът на Petersen са нарисувани като външен цикъл с форма на правилен петоъгълник, още 5 "радиални" ребра навътре, и още един цикъл с дължина 5 във вътрешността. Разликата между двата графа е, че при графа на Petersen вътрешният цикъл е нарисуван като пентаграма, а при графа на Фигура 2.48 вътрешният цикъл е нарисуван като правилен петоъгълник. Това е от съществено значение! – забележете, че графът, показан на Фигура 2.48, има цикли с дължина 4, а графът на Petersen няма такива.

Допълнение 10: Хроматичният индекс е или $\Delta(G)$, или $\Delta(G)+1$

И така, видяхме примери за графи, чиито ребра може да бъдат оцветени в $\Delta(G)$ цвята, и примери за графи, чиито ребра не може да бъдат оцветени в $\Delta(G)$ цвята, но може да бъдат оцветени в $\Delta(G)+1$ цвята. Важен теоретичен резултат на съветския математик Визинг е, че други графи няма: можем да оцветим ребрата на графа с $\Delta(G)+1$ цвята

ребрата на всеки граф [62].

Теорема 14: Теорема на Визинг

За всеки граф G, $\chi'(G) \in \{\Delta(G), \Delta(G)+1\}$.

Доказателство Доказателството, което ще разгледаме, е модифицирана версия на доказателството на теоремата на Визинг от *тази статия на Robert Green*.
Това, че $\chi'(G) \geqslant \Delta(G)$, е очевидно. Ще докажем, че $\chi'(G) \leqslant \Delta(G)+1$ за всеки граф G.

Да допуснем противното. Нека G е граф-контрапример с минимален брой ребра. Това означава, че за всяко ребро $e \in E(G)$ е вярно, че $\chi'(G-e) \leqslant \Delta(G)+1$[a]. За краткост ще означаваме $\Delta(G)+1$ с Δ.
Допускането ни е, че $\chi'(G) > \Delta$. Тъй като G е реброво-минимален контрапример, заключаваме, че $\chi'(G) = \Delta+1$. Избираме произволно ребро $e \in E(G)$. Нека $H = G-e$. По допускане, $\chi'(H) = \Delta$. Нека $e = (x, y_0)$. Нека ϕ е произволно реброво оцветяване на H в Δ цвята.
За всеки $u \in V(H)$, за всеки цвят c:

- ако нито едно ребро от $\mathcal{I}(u)$ не е оцветено в c, казваме, че c *липсва в* u,
- в противен случай казваме, че c_i *присъства в* u.

Веднага забелязваме, че във всеки връх липсва поне един цвят, понеже са повече от максималната степен на връх в H.

Изпълняваме следния алгоритъм, който строи максимална по включване редица от два по два различни върхове $y_0, y_1, \ldots, y_k$ и от цветове $c_0, c_1, \ldots, c_k$. Такава редица се нарича *верига на Кемпе*.

- Знаем, че в y_0 липсва поне един цвят. Да речем, че в y_0 липсва c_0. Ако c_0 липсва и в x, веднага следва, че G е реброво оцветим в Δ цвята: просто допълваме ϕ с оцветяване на (x, y_0) в c_0. Но G не е реброво оцветим в Δ цвята по допускане. Следователно, c_0 присъства в x.
- Нека (x, y_1) е реброто, такова че $\phi((x, y_1)) = c_0$. Очевидно, $y_1 \neq y_0$. В y_1 обаче липсва някакъв цвят. Да кажем, че в y_1 липсва c_1. Забелязваме, че c_1 не може да липсва в x, понеже, ако c_1 липсва в x, то G е реброво оцветим в Δ цвята. Очевидно е, че $c_1 \neq c_0$.
- Нека (x, y_2) е реброто, такова че $\phi((x, y_2)) = c_1$. Очевидно $y_2 \neq y_1$. Освен това $y_2 \neq y_0$, защото (x, y_2) е ребро в H, а H няма ребро (x, y_0). Ерго, $y_2 \notin \{y_0, y_1\}$. В y_2 липсва цвят, да кажем c_2. Забелязваме, че $c_2 \neq c_1$, защото c_1 присъства в y_2, а c_2 липсва в y_2.

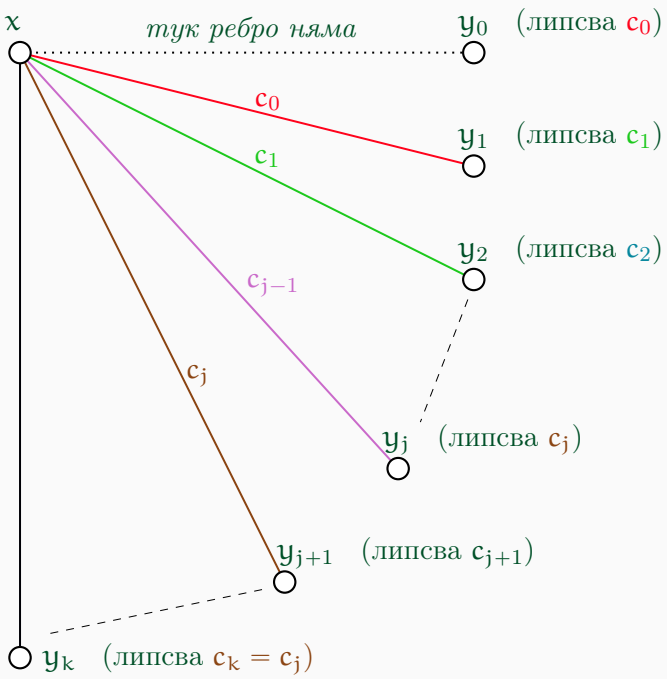
  Не е невъзможно обаче c_2 да е същият като c_0. Ако $c_2 = c_0$, спираме алгоритъма. В противен случай забелязваме, че c_2 не може да липсва в x, иначе G би бил реброво оцветим в Δ цвята.
- Нека (x, y_3) е ребро, такова че $\phi((x, y_3)) = c_2$. Очевидно $y_3 \neq y_0$, понеже (x, y_0) не е ребро в H. Тъй като $c_2 \notin \{c_0, c_1\}$, а c_0 и c_1 са цветовете съответно на (x, y_1) и (x, y_2), виждаме, че $y_3 \neq y_1$ и $y_3 \neq y_2$. Накратко, $y_3 \notin \{y_0, y_1, y_2\}$.

В y_3 липсва цвят, да кажем c_3. Очевидно $c_3 \neq c_2$. Не е невъзможно обаче c_3 да е същият като c_0 или c_1. Ако $c_3 \in \{c_0, c_1\}$, спираме алгоритъма. В противен случай забелязваме, че c_3 не може да липсва в x, иначе G би бил реброво оцветим в Δ цвята.

- Нека $e_4 = (x, y_4)$ е ребро, такова че $\phi((x, y_4)) = c_3$. И така нататък.
- Рано или късно ще стигнем до ребро $e_k = (x, y_k)$, такова че:
  - $\phi((x, y_k)) = c_{k-1}$, като $y_k \notin \{y_0, y_1, \ldots, y_{k-1}\}$ и $c_{k-1} \notin \{c_{k-2}, \ldots, c_1, c_0\}$;
  - в y_k липсва цвят c_k, такъв че $c_k \in \{c_{k-2}, \ldots, c_1, c_0\}$.

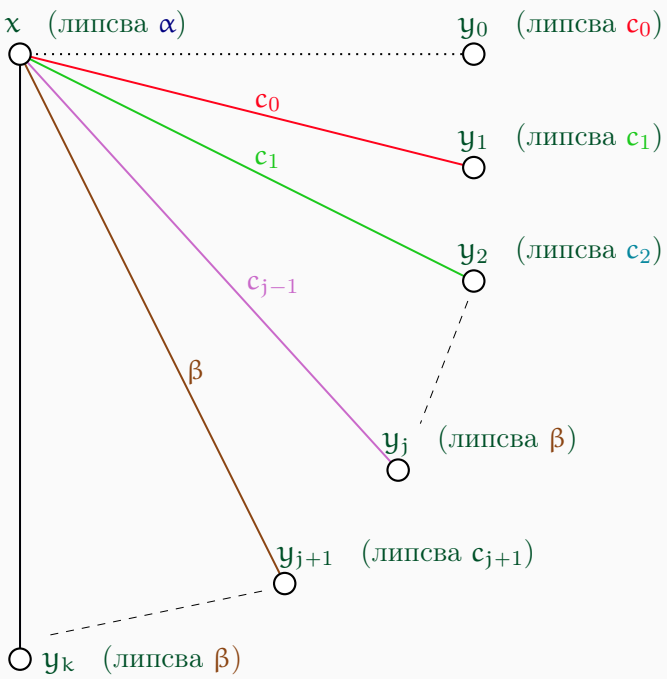
  Това е и причината за спиране на алгоритъма: за първи път сме се натъкнали на връх, а именно y_k, на който липсва цвят, който вече е липсвал в някой от досега сложените върхове в редицата; да кажем, че въпросният цвят липсва в y_j за някое $j \in \{0, \ldots, k-2\}$. Тогава $c_k = c_j$. Вижте Фигура 2.49.

Фигура 2.49 : Построяването на веригата на Kempe.



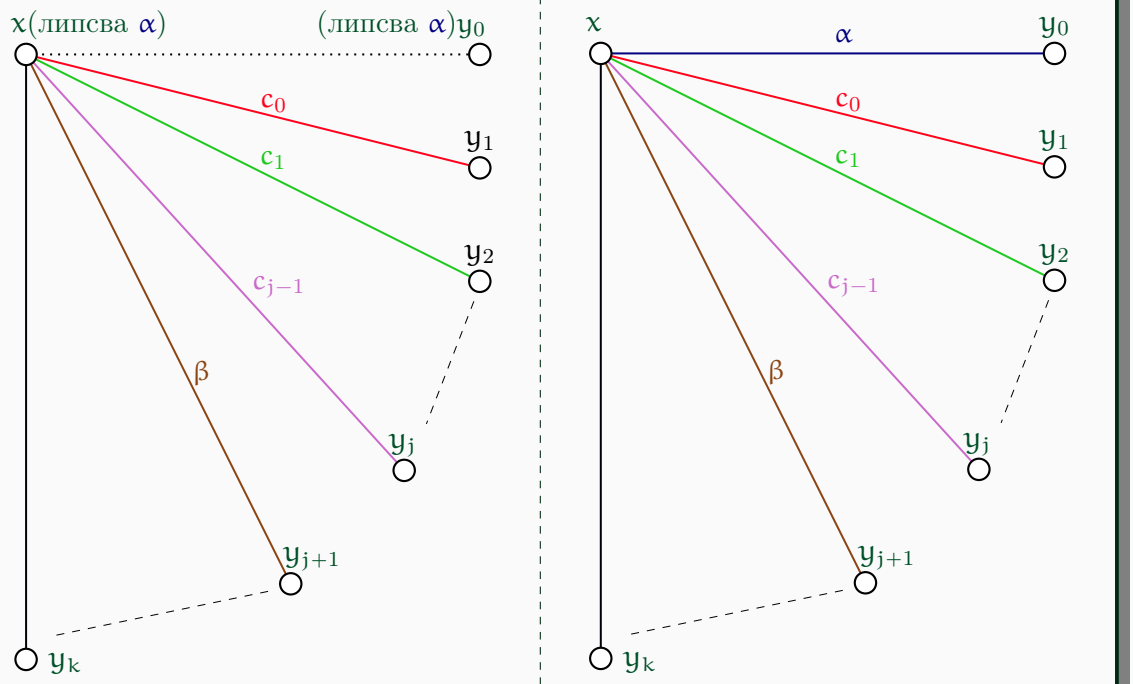
Във връх x липсва някакъв цвят. Да кажем, че липсва цвят α, където α е един от цветовете, с които са оцветени ребрата на H. Да преименуваме цвета $c_j = c_k$, който липсва в y_k и y_j, на β. Вижте Фигура 2.50.

Фигура 2.50 : Във връх x липсва цвят α.



Твърдим, че α присъства в y_0. Ако допуснем противното, веднага заключаваме, че можем да оцветим G в Δ цвята, използвайки цвят α за реброто (x, y_0) – ако α липсва и в x, и в y_0 в оцветяването ϕ на H, то α липсва и в x, и в y_0 в G в частичното оцветяване ϕ на G (ϕ е **частично** оцветяване върху G, понеже в него (x, y_0) няма цвят), ерго, ако оцветим (x, y_0) в α, няма да нарушим изискването никой връх да не е инцидентен с повече от едно ребро от даден цвят. Вижте Фигура 2.51.

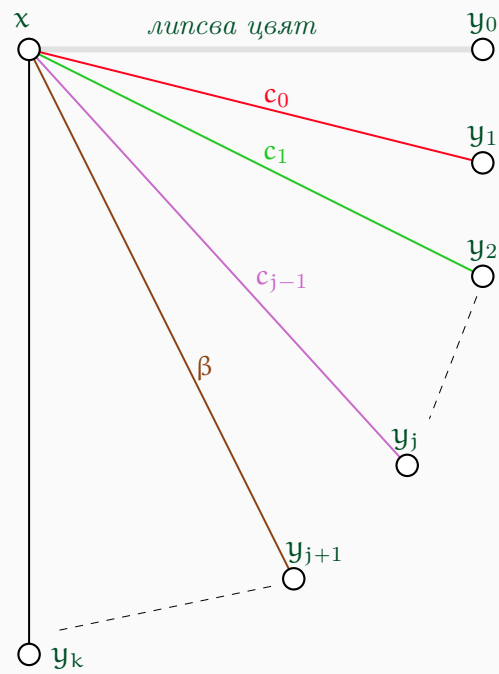
Фигура 2.51 : Ако α липсва в y_0 в H, то G е реброво оцветим в Δ цвята.



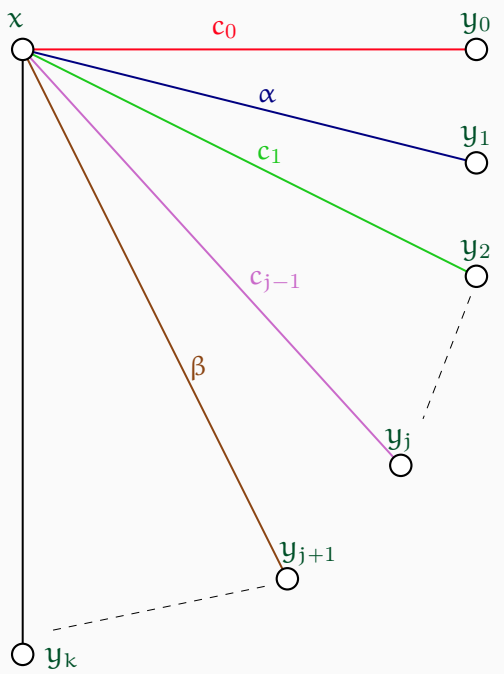
α липсва в x и y_0 в H. Оцветяваме (x, y_0) в α в G.

Твърдим, че α присъства в y_1 в H. Аргументацията е по същество същата. Да допуснем, че α липсва в y_1 в H. Да добавим реброто (x, y_0), получавайки G, но това е G с частично оцветяване, понеже (x, y_0) няма цвят. Оцветяваме (x, y_0) в цвят c_0 и махаме c_0 от (x, y_1), получавайки частично оцветяване на G – сега (x, y_1) няма цвят. Забелязваме, че в това частично оцветяване няма връх, инцидентен с повече от едно ребро от даден цвят, защото c_0 липсва в y_0 поначало. Тогава, ако α липсва в y_1, можем да оцветим (x, y_1) в α, получавайки оцветяване на G в Δ цвята. Неформално казано, ако α липсва в y_1, "завъртаме" цвят c_0 от (x, y_1) към (x, y_0) и си "отваряме възможност" да оцветим (x, y_1) в α. Вижте Фигура 2.52.

Фигура 2.52 : Ако α липсва в y_1 в H, може да оцветим G реброво в Δ цвята.



Поначало (x, y_0) няма цвят в G.



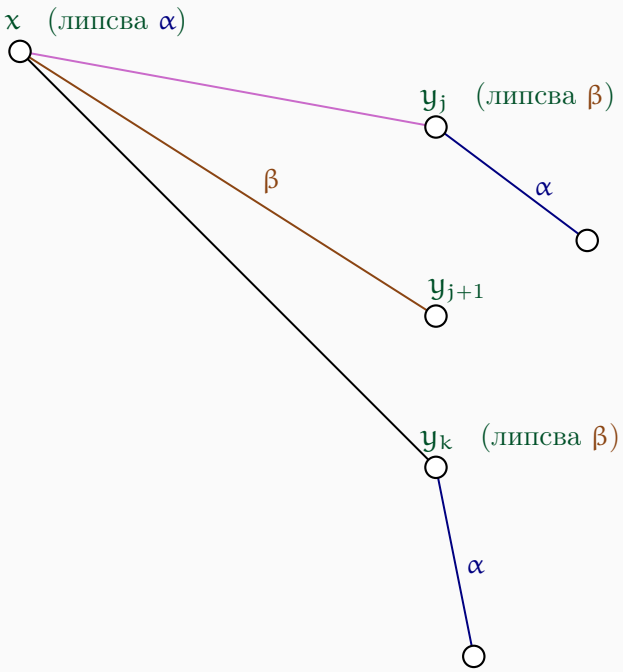
Оцветяваме (x, y_0) в c_0 и (x, y_1) в α.

Лесно се вижда, че α присъства в y_2, y_3, ..., y_k. Практически същата аргументация може да се използва за всеки от тези върхове: за y_2 "завъртаме" c_0 и c_1, и така нататък.

И така, α присъства във всеки връх от веригата на Кемпе. За доказателството ни е съществено само това, че α присъства в y_j и y_k в H. Да си припомним, че β липсва и в y_j, и в y_k. От друга страна, β присъства в x, но в x липсва α (вижте Фигура 2.50).

Да се фокусираме само върху y_j, y_k, всеки от които има инцидентно ребро (това може да са две различни ребра или едно и също ребро, това няма значение), оцветено в α, и върху реброто (x, y_{j+1}), оцветено в β (вижте Фигура 2.53).

Фигура 2.53 : Същностната част от графа в доказателството.



Нека J е подграфът на H, индуциран от ребрата (Определение 9), оцветени в α или β. Ключово наблюдение е, че J няма върхове от степен, по-голяма от две: ако имаше поне един такъв връх, то в H би имало връх, инцидентен с повече от едно ребро в един и същи цвят (α или β). Очевидно е, че граф, който има върхове от максимална степен не повече от две се състои от множество пътища и цикли, които два по два нямат общи върхове. С други думи, тези пътища и цикли са му свързаните компоненти. Нека $\mathcal{P}$ означава множеството от пътища-свързани компоненти на J.

Сега забелязваме, че върховете x, y_k и y_j са крайни точки на пътища от $\mathcal{P}$. Може това да са три различни пътя от $\mathcal{P}$, може два от тези върхове да са крайни точки на един и същи път от $\mathcal{P}$, а третият да е крайна точка на друг път от $\mathcal{P}$. Причината да са непременно крайни точки е, че всеки от тези три върха е инцидентен с точно едно ребро от E(J). Очевидно не може и трите върха да са крайни точки на един и същи път, защото един път има най-много две крайни точки.

Същността на доказателството е в това да "пренаредим" цветове върху ребрата на G по такъв начин, че реброто (x, y_j) да бъде оцветено в α. Този цвят липсва в x поначало, но присъства в y_j. Трябва пренареждането на цветове да е такова, че реброто, инцидентно с y_j, което има цвят α в първоначалното оцветяване (което е разширение на ϕ), да се окаже в цвят β, без да се появи инцидентно с y_j ребро в α. Тогава спокойно можем да оцветим (x, y_j) в α. До края на доказателството разглеждаме два случая.

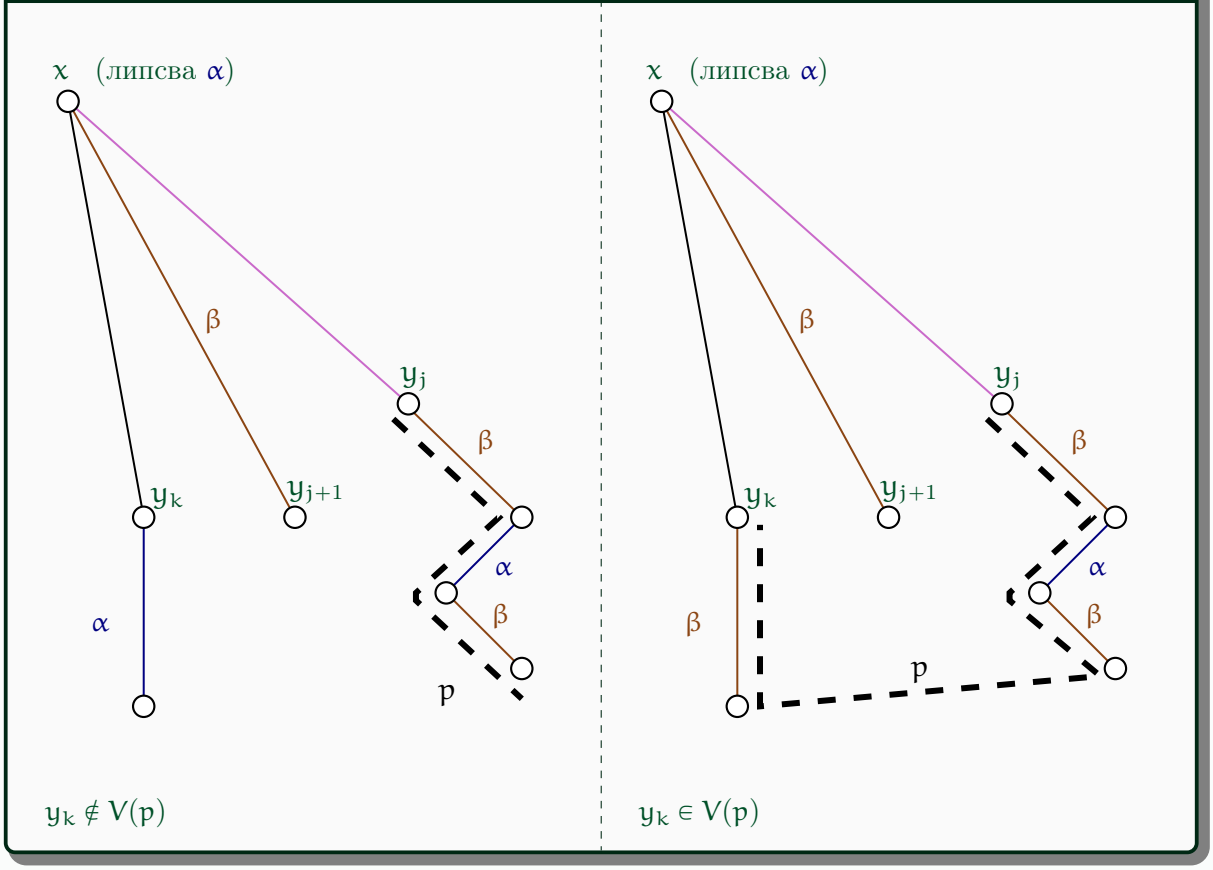
Случай 1. y_j е крайна точка на път $p \in \mathcal{P}$ и $x \notin V(p)$.

Припомняме си, че p е път от ребра, които са в цветовете α или β. Нещо повече, тези цветове алтернират върху ребрата на p, иначе би имало връх от p, инцидентен с две ребра в един и същи цвят. да обърнем цветовете върху p от α в β и обратно. Това "обръщане" на цветове по никакъв начин не нарушава ограничението в H да няма връх,

инцидентен с повече от едно ребро в даден цвят, защото p е "отдалечен" от другите свързани компоненти на J, нямайки общ връх с никой от тях. В частност, връх x остава инцидентен с точно едно ребро в цвят β след "обръщането" на цветовете. Фигура 2.54) илюстрира резултата от това обръщане на цветовете върху p, като се разглеждат двата възможни случая:

- y_k не е връх от p и реброто (x, y_k) не бива засегнато от обръщането на цветовете и си остава в цвят α
- y_k е връх от p, което означава, че y_k е другият край на p, при което реброто (x, y_k) получава цвят β.

Фигура 2.54 : След обръщането на цветовете върху p от α в β и обратно.

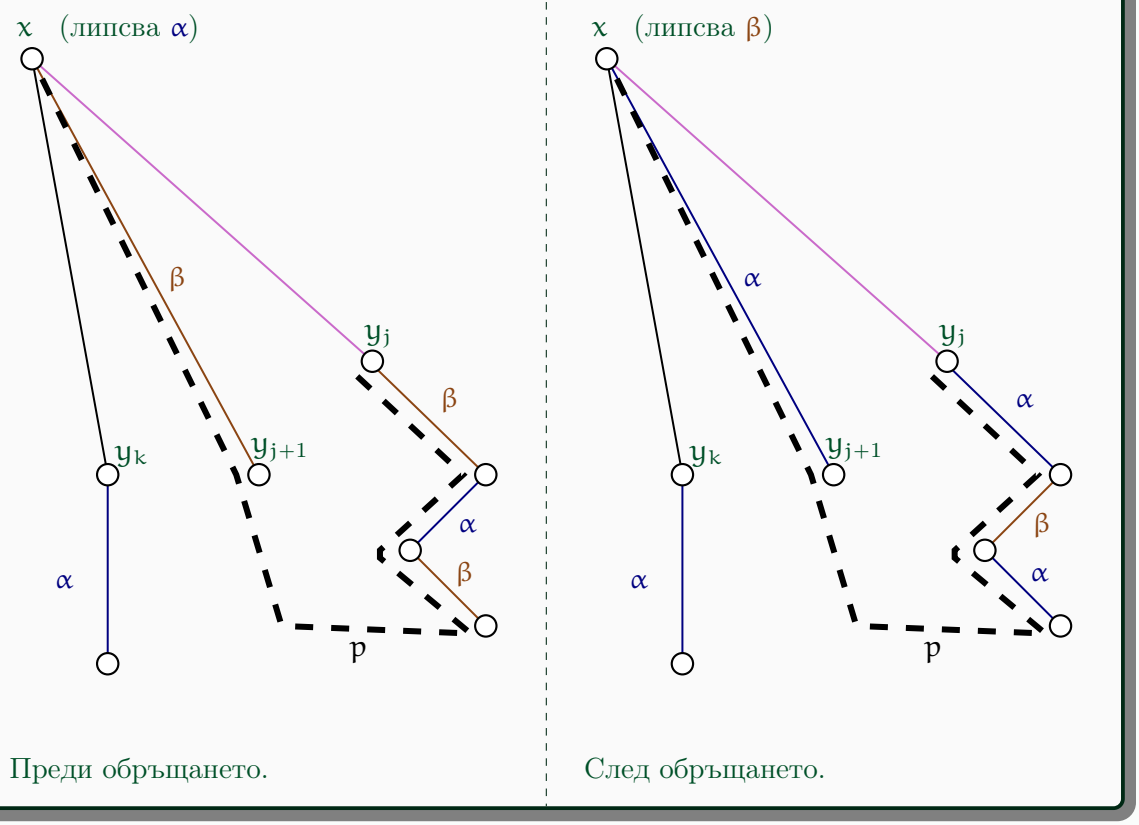


И в двата случая продължаваме така. Вземаме оцветяването ϕ на H и го пренасяме върху G като частично оцветяване. При това реброто (x, y_0) няма цвят. "Завъртаме" цветовете по веригата на Кемпе така: всяко от ребрата (x, y_0), ..., (x, y_{j-1}), да го наречем (x, y_i), получава цвета, който е бил върху (x, y_{i+1}) преди "завъртането", а (x, y_j) остава без цвят. Очевидно това не води до връх, инцидентен с повече от едно ребро в един и същи цвят. Оцветяваме (x, y_j) в α и получаваме реброво оцветяване на G в Δ цвята.

Случай 2. y_j е крайна точка на път $p \in \mathcal{P}$ и $x \in V(p)$.
Тогава x е другият край на p. Да поясним: единият край на p е връх y_j, следван от реброто в цвят α, а другият му край е връх y_{j+1}, следван от реброто (x, y_{j+1}) в цвят

β, следвано от връх x. Обръщаме цветовете върху p и сега единият край на p е връх y_j, следван от реброто в цвят β, а другият му край е връх y_{j+1}, следван от реброто (x, y_{j+1}) в цвят α, следвано от връх x. Това е показано на Фигура 2.55.

Фигура 2.55 : Преди и след обръщането на цветовете върху p.



Преди обръщането. След обръщането.

Довършваме доказателството по начин, аналогичен на **Случай 1**. Вземаме оцветяването ϕ на H и го пренасяме върху G като частично оцветяване. При това реброто (x, y_0) няма цвят. "Завъртаме" цветовете по веригата на Кемпе така: всяко от ребрата (x, y_0), $\ldots$, (x, y_{j-1}), да го наречем (x, y_i), получава цвета, който е бил върху (x, y_{i+1}) преди "завъртането", а (x, y_j) остава без цвят. Очевидно това не води до връх, инцидентен с повече от едно ребро в един и същи цвят. Реброто (x, y_{j+1}) не бива засегнато от тази промяна на цветове. Оцветяваме (x, y_j) в α и получаваме реброво оцветяване на G в Δ цвята. □

[a]Забележете, че $\Delta(G) > 1$, понеже празният граф и графите с максимална степен 1 не са контрапримери.

Допълнение 11: Линеен граф на граф

На пръв поглед, хроматично число на граф и хроматичен индекс на граф са принципно различни неща. Сега ще видим, че хроматичният индекс е частен случай на хроматичното число. За целта ни трябва следното определение.

Определение 38: Линеен граф на граф

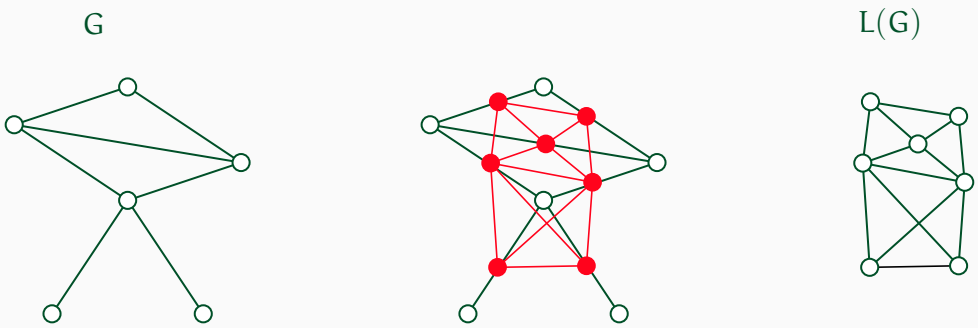
Нека $G = (V, E)$ е граф. *Линейният граф на* G е графът $G' = (E, E')$, където

$$E' = \{(e_1, e_2) \mid e_1, e_2 \in E \text{ и } e_1 \text{ и } e_2 \text{ са инцидентни в } G\}$$

Пишем $G' = L(G)$.
Казваме, че G *е линеен граф*, ако съществува граф H, такъв че $G = L(H)$.

Неформално, G' е графът, чиито върхове са **ребрата** на G, като два върха на G' са съседи тогава и само тогава, когато в G тези ребра имат общ връх. Понятието "линеен граф", в оригинал *line graph*, е наложено от Frank Harary [32, стр. 71]. Според Harary, за това понятие са били използвани много други термини, например *interchange graph*, *derived graph*, *edge-to-vertex dual* и други. В контекста на съвременната терминология на теорията на графите, терминът "line graph" не е особено удачен. Harary [32] нарича "lines" ребрата на графите, с което създава впечатлението, че графите са геометрични обекти—щом имат линии—а ние всячески се стремим да подчертаем, че графите са теоретико-множествени обекти. Но, така или иначе, терминът "line graph" е широко приет в света на графите в момента и ние не можем да го игнорираме. Доколкото е известно на автора на тези лекционни записки, на български няма възприет и утвърден термин, така че тук ще казваме "линеен граф".
Ето пример. Да разгледаме Фигура 2.56. Вляво е показан граф G. В средата е показан същия граф, като върху ребрата му са нарисувани нови върхове в червено и червените върхове са свързани с червени ребра тогава и само тогава, когато съответните им ребра са инцидентни. Този граф в червено е $L(G)$. Вдясно е показан само $L(G)$.

Фигура 2.56 : Граф G и линейният му граф $L(G)$.

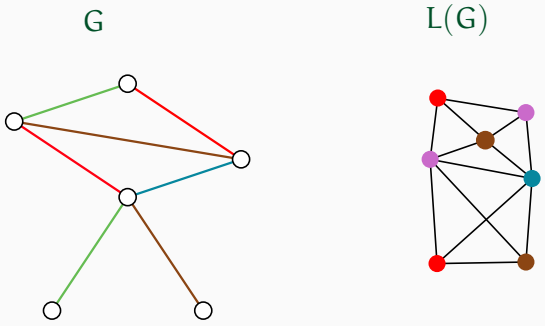


Наблюдение 18

За всеки граф G, оцветяването на ребрата на G в k цвята е същото като оцветяването на върховете на G' в k цвята. Следователно, $\chi'(G) = \chi(L(G))$.

Да разгледаме като пример G от Фигура 2.56. $\chi'(G) = 4$, тъй като, от една страна, $\Delta(G) = 4$, а от друга страна, G има реброво оцветяване в 4 цвята (вижте Фигура 2.57).

Фигура 2.57 : $\chi'(G) = \chi(L(G))$.



Погледнете пак Фигура 2.57. Това, че G има връх от степен 4 директно съответства на факта, че $L(G)$ има 4-клика. Върхът от степен 4 в G дава долна граница 4 за $\chi'(G)$. Съответно, 4-кликата в $L(G)$ дава долна граница 4 за $\chi(L(G))$.

Не всеки граф е линеен граф. Със сигурност за всеки граф G е дефиниран графът $L(G)$, но за даден граф G може да няма граф H, такъв че $G = L(H)$. Най-простият пример за граф, който не е линеен граф, е $K_{1,3}$ (графът G_1 на Фигура 2.58)[a]. За да се убедим, че $K_{1,3}$ не е линеен граф, да допуснем противното – съществува граф G, такъв че $L(G)$ е изоморфен на $K_{1,3}$. Тъй като $K_{1,3}$ има четири върха, G има четири ребра. Нещо повече: едно от ребрата на G е инцидентно с всяко от другите три ребра, а измежду тях нито две не са инцидентни. Това, разбира се, е невъзможно. Следователно, $K_{1,3}$ не е линеен граф.

Следната теорема (вижте [32, стр. 74]) дава нетривиални необходими и достатъчни условия за това, даден граф да е линеен граф.

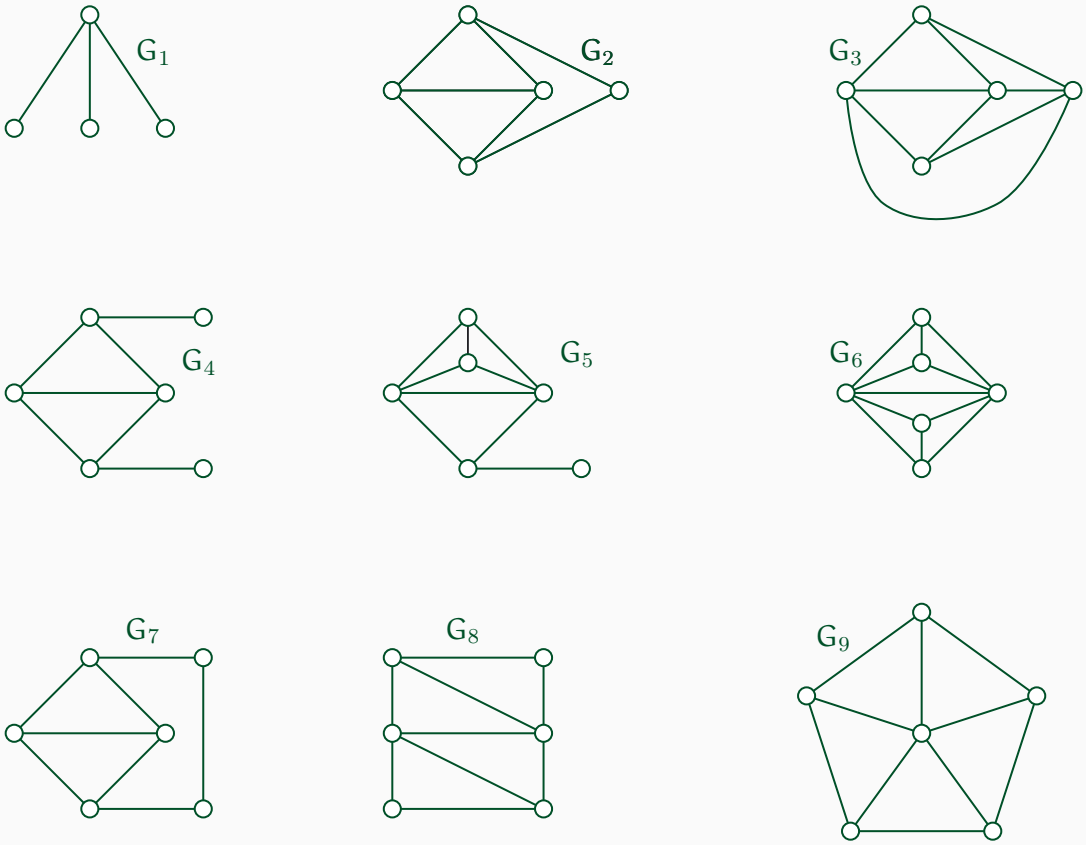
Теорема 15: Теорема 8.4 от [32] за линейните графи

Следните четири условия са еквивалентни:

1. G е линеен граф.
2. Ребрата на G може да се разбият на множества, всяко от което индуцира пълен подграф, като всеки връх на G се намира в най-много два такива подграфа.
3. G няма индуциран подграф $K_{1,3}$. Нещо повече, за всеки два подграфа K_3, за всеки от които е вярно, че има връх на G, съседен на нечетен брой от неговите (на подграфа K_3) върхове, които (два подграфа K_3) имат общо ребро, е вярно, че множеството от техните (на двата подграфа K_3) върхове индуцира подграф K_4.
4. Нито един от деветте графа на Фигура 2.58 не е индуциран подграф на G.

□

Фигура 2.58 : Деветте забранени индуцирани подграфи за лин. графи.



От Наблюдение 18 и от факта, че не всеки граф е линеен, следва, че хроматичният индекс е същински[б] частен случай на хроматичното число. С други думи, хроматичното число е строго по-общото понятие, като можем да избегнем въвеждането на хроматичен индекс и вместо да говорим за хроматичния индекс на G, да говорим за хроматичното число на $L(G)$.

[а] Друго име за $K_{1,3}$ е *claw graph*, защото може да бъде нарисуван като краче на птица.
[б] "Същински" в смисъл, че хроматичното число не е частен случай на хроматичния индекс.

2.8 Изоморфизъм на графи

2.8.1 Определение

Терминът "изоморфен" означава "със същата форма"[†]. Неформално казано, два графа са изоморфни, ако е изпълнено следното. Нека единият граф е нарисуван с произволна рисунка. Нека другият граф е материализиран от топчета и гумени ластици (от някаква идеална гума, която може да се разтяга и свива неограничено по наше желание), като топчетата отговарят на върховете, а ластиците, на ребрата. И така, графите са изоморфни, ако можем да наложим изработеното от топчета и ластици съответствие на втория граф върху рисунката на първия граф по такъв начин, че всяко топче от втория да "легне" върху точно една точка от рисунката на първия, и всяко гумено ребро на втория да легне точно върху точно едно от нарисуваните ребра на първия. Очевидно за целта трябва двата графа да имат един и същи брой върхове, а също така и един и същи брой ребра. Но, както ще видим след малко, това не е достатъчно, за да бъдат изоморфни.

Определение 39: Изоморфизъм на графи.

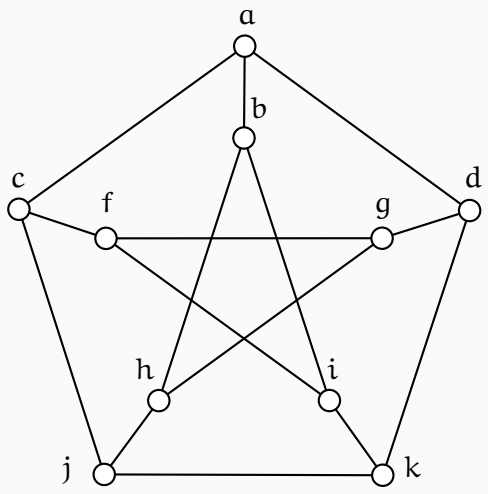
Нека $\mathsf{G}' = (\mathsf{V}', \mathsf{E}')$ и $\mathsf{G}'' = (\mathsf{V}'', \mathsf{E}'')$ са графи. *Изоморфизъм между* G' *и* G'' е всяка биекция $\phi : \mathsf{V}' \to \mathsf{V}''$, такава че

$$\forall u, v \in \mathsf{V}' : (u, v) \in \mathsf{E}' \leftrightarrow (\phi(u), \phi(v)) \in \mathsf{E}''$$

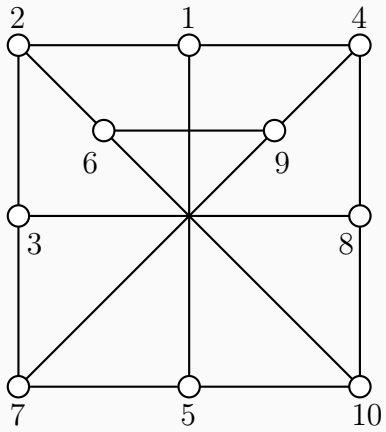
Казваме, че G' и G'' са изоморфни, ако съществува изоморфизъм между тях.

Пример за изоморфни графи е показан на Фигура 2.59.

Фигура 2.59 : Тези два графа са изоморфни.



Класически нарисуван граф на Petersen.



Нетипично нарисуван граф на Petersen.

Графът на Фигура 2.59 вляво е "класически" нарисуван граф на Petersen с имена на върховете. Графът на Фигура 2.59 вдясно на пръв поглед е доста различен от графа вляво, но

[†]Етимологията е следната: на гръцки ἴσος означава "равен", а μορφή означава "форма".

всъщност тези графи са изоморфни; графът вдясно е графът на Petersen, нетипично нарисуван, с имена на върховете. Една възможност—със сигурност не единствена—е следната:

$$\phi(a) = 6 \qquad \phi(b) = 10 \qquad \phi(c) = 2 \qquad \phi(d) = 9 \qquad \phi(f) = 1 \qquad (2.10)$$
$$\phi(g) = 4 \qquad \phi(h) = 8 \qquad \phi(i) = 5 \qquad \phi(j) = 3 \qquad \phi(k) = 7$$

Тогава съответствието на ребрата е следното:

(a, b) съответства на $(6, 10)$, (a, c) съответства на $(6, 2)$, (a, d) съответства на $(6, 9)$
(b, h) съответства на $(10, 8)$, (b, i) съответства на $(10, 5)$, (c, f) съответства на $(2, 1)$
(c, j) съответства на $(2, 3)$, (d, g) съответства на $(9, 4)$, (d, k) съответства на $(9, 7)$
(f, g) съответства на $(1, 4)$, (f, i) съответства на $(1, 5)$, (g, h) съответства на $(4, 8)$
(h, j) съответства на $(8, 3)$, (i, k) съответства на $(5, 7)$, (j, k) съответства на $(3, 7)$

Наблюдение 19

Понятието "изоморфизъм" задава бинарна релация над множеството на графите – два графа са в тази релация тогава и само тогава, когато
са изоморфни. Релацията на изоморфизъм е рефлексивна, симетрична и транзитивна, което означава, че е релация на еквивалентност.

Тази релация често се бележи с $\simeq$. Пишейки "$G_1 \simeq G_2$", имаме предвид, че G_1 и G_2 са изоморфни.

Определение 39 не е в сила за мултиграфи, понеже при тях и върховете, и ребрата са протоелементи и свързващата функция е тази, която отнася ребра към (ненаредени двойки) върхове.

Определение 40: Изоморфизъм на мултиграфи.

Нека $G_1 = (V_1, E_1, f_1)$ и $G_2 = (V_2, E_2, f_2)$ са мултиграфи съгласно Определение 13. Изоморфизъм между G_1 и G_2 е всяка наредена двойка (φ, ψ), където φ и ψ са биекции

$$\varphi : V_1 \to V_2$$
$$\psi : E_1 \to E_2$$

такива че

$$\forall e \in E_1 : f_2(\psi(e)) = \{a, b\}$$

където $a = \varphi(u)$ и $b = \varphi(v)$, където $\{u, v\} = f_1(e)$.

Казваме, че мултиграфите G_1 и G_2 са изоморфни, ако съществува изоморфизъм между тях, и бележим този факт с "$G_1 \simeq G_2$", точно както при обикновените графи.

Би трябвало да е очевидно, че $u, v \in V_1$ и $a, b \in V_2$.

2.8.2 Именувани и неименувани графи

Графи с именувани и неименувани върхове

Да си припомним основното определение в тези лекционни записки: Определение 1. Според него, "граф" е наредена двойка от две множества, като едното от тях е опорното множество

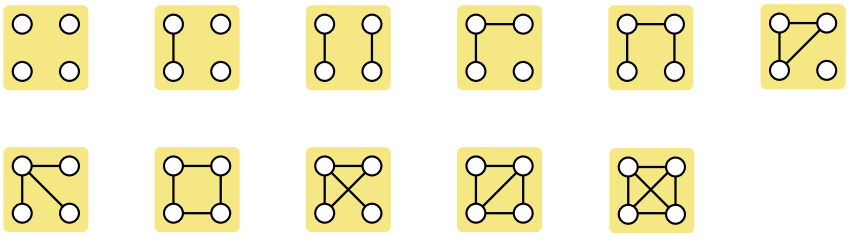
и това е множеството от върховете. Щом е множество, то неговите елементи–върхове имат идентичност в смисъл, че различаваме всеки връх от всеки друг връх. За целта върховете имат уникални имена.

На практика обаче, много често ни интересува само фо́рмата на графите, а не идентификаторите (имената) на върховете. Да разгледаме следното наблюдение.

Наблюдение 20

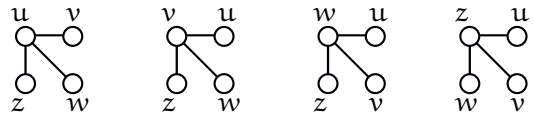
Съществуват точно единадесет графа с четири върха.

Това вярно ли е? Ако следваме буквата на Определение 1, то Наблюдение 20 е **невярно**, защото при 4 върха има $\binom{4}{2} = 6$ потенциални ребра, всяко от които може да присъства или не, така че броят на всички графи е точно $2^6 = 64$, а не 11. За да се убедим, че Наблюдение 20 не е безсмислица, да разгледаме следните 11 графа:



Тук разглеждаме графи с **анонимни** върхове. Те се различават един от друг само по фо́рмата си. Такива графи се наричат *неименувани*[†]. Това, че ги наричаме "графи" е формална злоупотреба с Определение 1, както ще видим по-долу, защото нито един от тях не е обект, дефиниран от Определение 1, но на практика понятието "неименуван граф" е много полезно и е абсолютно недвусмислено.

За по-ясно обяснение, да допуснем, че имената на четирите върха са u, v, w и z, и да разгледаме следните четири графа с такива имена на върховете:



G_1 G_2 G_3 G_4

Забележете, че това са различни графи! В G_1 единственият връх от степен три е u, в G_2 единственият връх от степен три е v, в G_3 единственият връх от степен три е w, и в G_4 единственият връх от степен три е z. Това не е един граф, нарисуван по четири начина, а са **четири различни графа** съгласно Определение 1. Такива графи наричаме *именувани графи*[‡], защото всеки връх има име. Забележете, че Определение 1 говори точно за именувани графи – в него върховете са елементи на множество, което означава, че са два по два различни. Следователно, именуваните графи са базовите обекти, които изучаваме в теорията на графите.

Сега да "изтрием" имената на върховете от $G_1, \ldots, G_4$. Ще получим четири копия на един и същи граф с анонимни върхове, а именно този:

[†]На английски терминът е *unlabeled graphs*.
[‡]На английски терминът е *labeled graphs*.

Този обект не е граф, ако прилагаме Определение 1 формално. Тук сме го нарисували, но вече отбелязахме, че рисунката на граф и самият граф са принципно различни неща. А как да запишем формално граф, чиито върхове са анонимни? Отговорът естествено се базира на понятието "изоморфизъм". Лесно се вижда, че $G_1 \simeq G_2 \simeq G_3 \simeq G_4$. Нещо повече, измежду 64-те именувани графа на 4 върха ($G_1, \ldots, G_4$ са измежду тях), няма други графи, които да са изоморфни на тях, така че $\{G_1, G_2, G_3, G_4\}$ е един клас на еквивалентост на релацията $\simeq$. И така, граф с анонимни върхове всъщност е точно един клас на еквивалентост на $\simeq$. Да повторим: формалната дефиниция на "неименуван граф" е: клас на еквивалентност на $\simeq$.

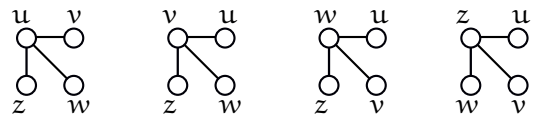
За упражнение да помислим колко класа на еквивалентност има $\simeq$, ако върховете са 4 на брой? Отговорът е 11: точно колкото са показаните по-горе неименувани графи на 4 върха. Всеки от тях съответства на точно един от класовете. Очевидно е, че въпросните класове на еквивалентност имат различни мощности: празният граф е единственият елемент в един от класовете, също така пълният граф е единствен, докато, както видяхме току-що, има четири графа в класа на еквивалентност, който отбелязахме с ⁜.

Графи с именувани върхове и ребра

Понятието "labeled graph" може да се разшири, като именувани биват и върховете, и ребрата. Според Определение 1 опорното множество се състои само от върховете, а ребрата са двуелементни подмножества от върхове. Това, че можем да въведем имена като "e_1", "e_2" и така нататък за ребрата (което правим на стр. 6) е извън дефиницията. Ако ограничим вниманието си до Определение 1, ще видим, че ребрата нямат други идентичности освен множествата от краищата си.

От друга страна, Определение 13 третира и върховете, и ребрата като опорни множества, използвайки свързващата функция като нещото, което асоциира ребро с върхове. Ако използваме Определение 13 за базово, то "именуван граф" е такъв, в който и върховете, и ребрата имат уникални имена. Размяната на имената на две ребра води до друг граф в този смисъл.

И така, има разлика между именуван граф, в който са именувани само върховете, и именуван граф, в който са именувани и върховете, и ребрата. Като пример да разгледаме отново всички графи с четири върха, с именувани върхове, като има един връх от степен 3 и три върха от степен 1. Ако имената на върховете са u, v, w и z, тези графи са точно следните:

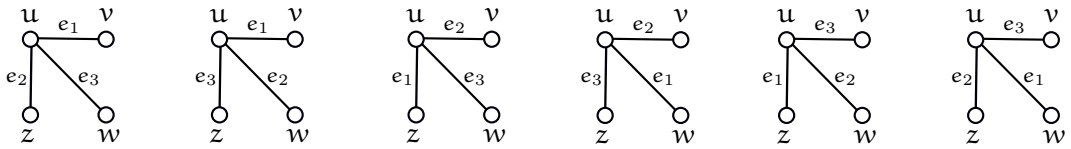


G_1 G_2 G_3 G_4

Заслужава да се натърти отново, че според Определение 1, ако $V = \{u, v, w, z\}$, то няма други графи с редица от степените $1, 1, 1, 3$ (Определение 6).

Ако обаче са именувани и върховете, и ребрата, графите с редица от степените $1, 1, 1, 3$ стават много повече. Да кажем, че имената на ребрата са e_1, e_2 и e_3. Тогава на всеки от тези четири

графа съответстват $3! = 6$ различни начина да бъдат "раздадени" имената на ребрата. Да вземем за пример G_1. Ето колко различни именувани графи му съответстват, ако ребрата също имат имена и тези имена са e_1, e_2 и e_3:



На въпроса "Колко именувани графи има на n върха?" типичният отговор е $2^{\binom{n}{2}}$, но това е само при положение, че именувани са само върховете. Ако са именувани и върховете, и ребрата, като множеството от имена за ребрата е $\{e_1, \ldots, e_{\binom{n}{2}}\}$, този брой нараства стремително.

Забележете, че дори в малкия пример с четири върха и редица от степените $1, 1, 1, 3$, ако множеството от имена на ребрата е $\{e_1, \ldots, e_6\}$, то именуваните графи (и върхове, и ребра) са 480. Защо?

- По $\binom{6}{3} = 20$ начина можем да изберем кои три от общо шестте ребра да вземем, и за всяко от тези вземания има шест различни начина да "раздадем" трите имена на трите ребра (което се илюстрира от последната рисунка), което дава общо 120 начина.
- Но тези 120 начина са по отношение на един единствен фиксиран връх от степен 3. Има 4 различни възможности за това, кой връх да е от степен 3, така че броят на графите скача до $4 \cdot 120 = 480$.

И това 480 съответства само на един клас на изоморфизъм, а именно (като неименуван граф):

Читателят лесно може да си изведе формула за броя на именуваните графи, където се именуват и върховете, и ребрата, като

- или имената на ребрата се вземат от $\{e_1, \ldots, e_{\binom{n}{2}}\}$, но разглеждаме всеки възможен брой ребра m, като от $0 \leqslant m \leqslant \binom{n}{2}$,
- или броят m на ребрата е фиксиран за някое $m \in \{0, \ldots, \binom{n}{2}\}$ и всички имена на ребра от $\{e_1, \ldots, e_m\}$ трябва да присъстват.

2.9 Хамилтонови пътища и цикли

През 1857 г. великият ирландски математик и физик William Hamilton измисля развлекателна игра, която нарича "The Icosian Game", след което продава интелектуалната собственост на играта за значителната за времето си сума от 25 британски лири. Играта е следната: върху дъска е нарисуван граф-додекаедър[†], като върховете са малки еднакви дупки, именувани с буквите A, ..., T (това са двадесет букви). Има двадесет "запушалки" (на английски, *plugs*), изработени от слонова кост и номерирани с числата 1, ..., 20, които могат да влизат в дупките. Идеята е да се направи "обиколка" на графа, която минава през всяка дупка точно веднъж и завършва в дупката, в която е започнала. Номерата върху "запушалките" указват реда, в който се минава през върховете. На сайта *puzzlemuseum.com* се вижда една от малкото запазени оригинални изработки на тази игра. Името на играта идва от факта, че додекаедърът има точно двадесет[‡] върха, така че обиколката трябва да мине през всичките двадесет дупки, преди да се върне там, откъдето е започнала.

В съвременната терминология са приети следните дефиниции.

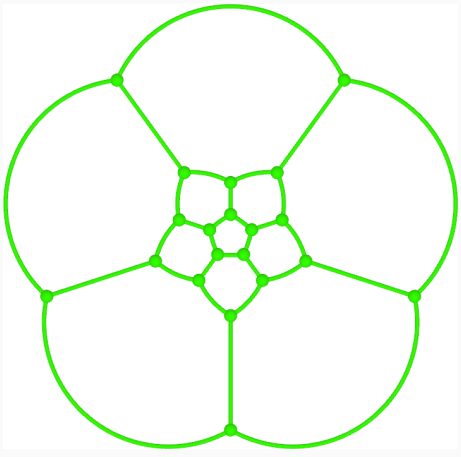
Определение 41: Хамилтонов цикъл и Хамилтонов път.

Нека е даден граф G. *Хамилтонов цикъл* в G е всеки цикъл в G, който съдържа всички върхове на G. *Хамилтонов път* в G е всеки път в G, който съдържа всички върхове на G. Казваме, че G е *Хамилтонов граф*, ако в него има Хамилтонов цикъл.

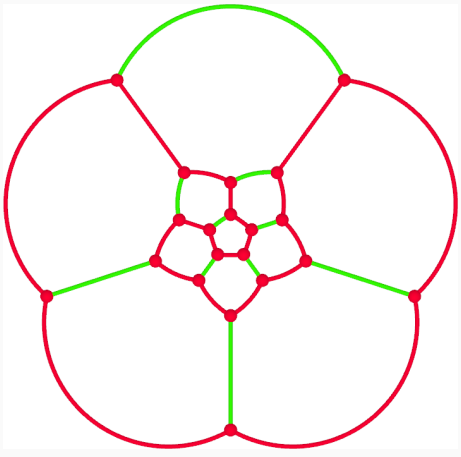
Обикновено в учебниците по теория на графите тези дефиниции съдържат и уточнението, че върховете в Хамилтоновия цикъл или път трябва да се срещат точно веднъж. Но ние вече приехме[§], че казвайки "цикъл" или "път" без определения, имаме предвид съответно прост цикъл или прост път, така че това уточнение за нас е излишно.

Фигура 2.60 съдържа изображение на графа-додекаедър—графът от играта на Hamilton—и Хамилтонов цикъл в него.

Фигура 2.60 : Граф-додекаедър и Хамилтонов цикъл в него.



Граф-додекаедър.



Хамилтонов цикъл в него.

[†]Да си припомним Фигура 2.78, която показва многостен-додекаедър, и Фигура 2.77, която показва граф-додекаедър, нарисуван върху сфера, и неговата стереографска проекция.

[‡]Както вече бе споменато на страница на стр. 149, *икоса-* идва от старогръцката дума εἴκοσι, която означава *двадесет*.

[§]Конвенция 5 на стр. 31 и Конвенция 7 на стр. 33.

Наблюдение 21

Ако граф е Хамилтонов, то той съдържа Хамилтонов път. Еквивалентно, ако няма Хамилтонов път, то той не е Хамилтонов. Конверсното не е вярно – може да има Хамилтонов път и да няма Хамилтонов цикъл.

Наблюдение 22

Нека $G = (V, E)$ е граф и $G' = (V, E')$ е негов подграф. Ако в G' има Хамилтонов цикъл или път, то в G също има съответно Хамилтонов цикъл или път. Ако в G няма Хамилтонов цикъл или път, то в G' също няма съответно Хамилтонов цикъл или път.

Не всеки граф е Хамилтонов и не всеки граф има Хамилтонов път. Ако графът е Хамилтонов, то всеки Хамилтонов цикъл има дължина точно n. Пълният граф K_n за $n \geqslant 3$ има $\frac{1}{2}(n-1)!$ Хамилтонови цикъла, ако мислим за тях като за подграфи (вж. Наблюдение 8). Пълният граф K_n за $n \geqslant 2$ има $\frac{1}{2}n!$ Хамилтонови пътя, ако мислим за тях като за подграфи (вж. Наблюдение 7).

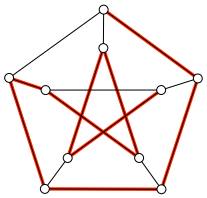
За някои **специфични** видове графи можем веднага да кажем дали има или няма Хамилтонови цикли или пътища, например:

- ако графът не е свързан, в него няма Хамилтонов път;
- ако графът има срязващ връх, той не е Хамилтонов;
- ако графът е пълен и $n \geqslant 3$, той е Хамилтонов;
- ако графът е двуделен и дяловете нямат една и съща мощност, той не е Хамилтонов;
- ако графът е пълен двуделен и дяловете са с еднаква мощност, която е поне 2, то той е Хамилтонов;
- ако графът е n-мерен хиперкуб (вж. Секция 2.15) и $n \geqslant 2$, то той е Хамилтонов;

но за **общия случай** не е известно просто необходимо и достатъчно условие за съществуване на Хамилтонов цикъл или път.

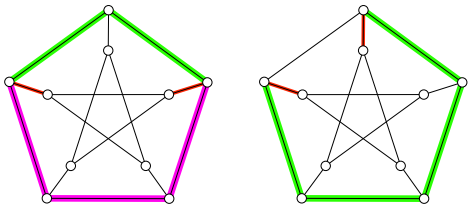
Допълнение 12: NP-трудност на Хамилтонов цикъл и път

Задачата дали даден граф има Хамилтонов цикъл е **NP**-трудна (вижте Допълнение 6). Това означава почти сигурно, че за тази задача няма практични алгоритми **в общия случай**. Същото е в сила за задачата дали даден граф има Хамилтонов път. Тези две задачи се свеждат бързо една до друга, така че, ако имахме ефикасен алгоритъм за едната, щяхме да имаме и за другата; и, обратно, ако се докаже, че за едната от тях няма бърз алгоритъм, ще следва, че за другата също няма бърз алгоритъм.
Колкото и да е странно, първото доказателство, че задачата за Хамилтоновия цикъл е **NP**-трудна, се базира на свеждане на задачата за върховото число (Секция 2.5) на граф до нея! За подробности вижте [26].



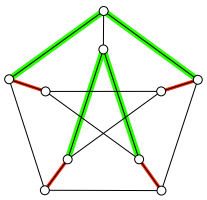
Фигура 2.61

Нека P означава графа на Petersen. Ще покажем, че P не е Хамилтонов, но в него има Хамилтонов път. Истинността на второто твърдение се вижда от Фигура 2.61. Да допуснем, че P е Хамилтонов[†]. Можем да мислим за P като за един външен 5-цикъл и един вътрешен 5-цикъл, като още пет ребра, да ги наречем *радиалните ребра*, свързват върховете във външния цикъл със съответни върхове във вътрешния цикъл. Изборът на външен цикъл и вътрешен цикъл е чисто условен, разбира се, понеже P има много симетрии. Всеки 5-цикъл в него може да бъде избран за външен, който избор определя напълно вътрешния цикъл. Нека по произволен начин е избран външен цикъл. Това определя еднозначно вътрешния цикъл и радиалните ребра. Нека Хамилтоновият цикъл се нарича c. Нека външният цикъл се нарича O, а вътрешният цикъл се нарича I. Да започнем мислено да "вървим" по c, тръгвайки от някой връх от $u \in V(\mathsf{O})$. Тъй като c е цикъл, ще се върнем в u края на обиколката. При това ще минаваме от върхове от $V(\mathsf{O})$ във върхове от $V(\mathsf{I})$—защото c съдържа и върховете от $V(\mathsf{I})$—и ще се връщаме обратно във върхове от $V(\mathsf{O})$, защото обиколката ни трябва да приключи в u. Тези минавания от $V(\mathsf{O})$ във $V(\mathsf{I})$ и от $V(\mathsf{I})$ в $V(\mathsf{O})$ стават през, и само през, радиалните ребра. Очевидно е, че на всяко минаване от от $V(\mathsf{O})$ във $V(\mathsf{I})$ и от $V(\mathsf{I})$ във $V(\mathsf{O})$ съответства точно едно радиално ребро измежду тези радиални ребра, които са в c. Поради това, броят на радиалните ребра в c е равен на броя въпросните минавания. Щом обиколката започва и завършва във връх от $V(\mathsf{O})$, то има четен брой минавания от външния във вътрешния цикъл и от вътрешния във външния, следователно c съдържа четен брой радиални ребра, който не е 0. Тъй като радиалните ребра са общо 5, то c съдържа или точно 2, или точно 4 радиални ребра.



Фигура 2.62

Да допуснем, че c съдържа точно 2 радиални ребра, както е показано на Фигура 2.62 вляво (въпросните две радиални ребра са в червено). Ако двете червени ребра съдържат върхове от O, които не са съседи в O, както е показано на лявата от двете подфигури, то при всяко "излизане" на c върху външния O, c трябва хем да съдържа зелените ребра, хем трябва да съдържа лилавите ребра, което е невъзможно. Сега да разгледаме възможността двете радиални ребра да съдържат върхове от O, които са съседи в O. Тази възможност е показана на дясната подфигура. Тогава c задължително съдържа четирите зелени ребра. Но тогава двете червени ребра трябва да имат върхове от I, които са съседи в I; това следва от напълно аналогични съображения на тези, от които изведохме, че краищата на червените ребра върху O са съседи в O. Но в графа на Petersen няма две радиални ребра, чиито краища са съседни както върху външния O, така и върху вътрешния цикъл I. И така, отхвърлихме възможността Хамилтоновият цикъл да съдържа точно две радиални ребра.



Фигура 2.63

Тогава c съдържа точно 4 радиални ребра. Ще илюстрираме тази част от доказателството на Фигура 2.63. Без ограничение на общността, нека четирите радиални ребра, които са в c, са ребрата, оцветени в червено. Разглеждаме останалото радиалното ребро, което ще наричаме "черното". Всеки от неговите два върха се съдържа в c. Тогава задължително в c се съдържат и двете ребра от O, инцидентни с връх от черното ребро, както и двете ребра от I, инцидентни с черното ребро. Става дума за общо четири ребра, които на Фигура 2.63 са маркирани със зелено. И така, c съдържа в себе си четирите червени и четирите зелени ребра. Но веднага се вижда, че това

[†]Доказателството, което ще покажем тук, е добре известно и може да бъде видяно например в онлайн *решенията на задачите* от учебника на Cameron [15].

е невъзможно. Възможно е да конструираме два 5-цикъла, които нямат общи върхове и съдържат червените и зелените ребра, но няма как да направим един цикъл, който да ги съдържа. Това е краят на доказателството.

От общи съображения е ясно, че при фиксиран брой на върховете, колкото повече ребра има в графа, толкова по-възможно[†] е в него да има Хамилтонов цикъл. Екстремният пример е K_n, който, както вече отбелязахме, е Хамилтонов (при $n \geqslant 3$). Аргументацията, че K_n е Хамилтонов, е тривиална—можем да "отидем" от който искаме в който искаме връх. Сега ще формулираме и докажем няколко нетривиални достатъчни условия за съществуване на Хамилтонов цикъл.

Теоремата на Dirac [20] е публикувана през 1952.

Теорема 16: Теорема на Dirac

Нека $G = (V, E)$ е граф, такъв че $n \geqslant 3$ и $\delta(G) \geqslant \left\lceil \frac{n}{2} \right\rceil$. Тогава G е Хамилтонов.

Доказателство: Съгласно Теорема 2, G е свързан. Нека $p = u_1, u_2, \ldots, u_k$ е максимален път в G.

Ще докажем, че всички съседи на крайните върхове на p са върхове от p. Ако допуснем, че u_1 има съсед w, който не е връх от p, можем да конструираме път $p_1 = w, u_1, u_2, \ldots, u_k$, който е по-дълъг от p, в противоречие с факта, че p е максимален път. Аналогично показваме, че всички съседи на u_k също са върхове от p.

Твърдим, че за поне едно i, такова че $1 \leqslant i < k$, е вярно, че u_i е съсед на u_k, а u_{i+1} е съсед на u_1. Да допуснем, че няма такова i. Тогава за всеки съсед на u_1, който е в p (а току-що показахме, че всички съседи на u_1 са в p) е вярно, че върхът вляво от него не е съсед на u_k. Тогава p съдържа поне следните върхове:

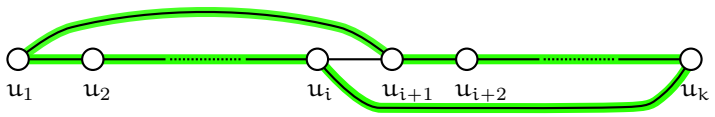
- върховете, които са вляво от съседите на u_1. Те **не са** съседи—съгласно последното допускане—на u_k. Съседите на u_1 са $d(u_1)$ на брой, следователно въпросните върхове също са $d(u_1)$ на брой;
- върховете, които **са** съседи на u_k, а те са на брой $d(u_k)$;
- връх u_k, който нито е вляво от съсед на u_1, нито е съсед на себе си.

Тези множества от върхове имат две по две празно сечение, следователно p има поне $d(u_1) + d(u_k) + 1 \geqslant 2\left\lceil \frac{n}{2} \right\rceil + 1$ върха. Тогава $|p| \geqslant n + 1$, което е невъзможно.

Следователно съществува поне едно i, такова че $1 \leqslant i < k$ и u_i е съсед на u_k, а u_{i+1} е съсед на u_1. Тогава в G съществува цикълът

$$c = u_1, u_{i+1}, u_{i+2}, \ldots, u_{k-1}, u_k, u_i, u_{i-1}, u_{i-2}, \ldots, u_2, u_1$$

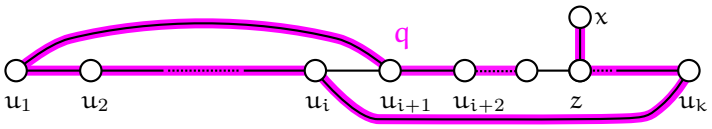
Ето илюстрация на цикъла c (в зелено):



[†]Нарочно не казваме "по-вероятно", защото терминът *вероятност* има строг математически смисъл. Каква е вероятността да съществува Хамилтонов цикъл като функция от броя на ребрата при фиксиран брой на върховете е въпрос, който е далече отвъд обхвата на тези лекционни записки.

Да допуснем, че $V(c) \neq V$. Тъй като G е свързан, съществува връх $x \in V \backslash V(c)$, който е съсед на връх z от p. z не може да е някой от u_1 или u_k, защото за всеки от тях, всички негови съседи са в p.

Да разгледаме пътя q, който започва с x, следващият връх е z, следващият е единият от двата съседа на z в p и после "върви" по цикъла c, докато не достигне другия съсед на z в p:



Очевидно $|q| = |p| + 1$, в противоречие с това, че p е максимален път. Тогава $V \backslash V(c) = \varnothing$ и c е Хамилтонов цикъл. □

Теоремата на Ore [45] е резултат от 1960.

Теорема 17: Теорема на Ore

Нека $G = (V, E)$ е граф, такъв че $n \geqslant 3$ и $\forall u, v \in V : (u, v) \notin E \rightarrow d(u) + d(v) \geqslant n$. Тогава G е Хамилтонов.

Доказателство: Да допуснем противното – G не е Хамилтонов. Очевидно G не е K_n. Да конструираме редица от графи, започвайки с G и завършвайки с K_n, като на всяка стъпка добавяме ребро между произволна двойка върхове, които до момента не са съседи. Иначе казано, ребро по ребро допълваме G до K_n. Тази редица се състои от $t + 1$ графа, където $t = \binom{n}{2} - m$. Да ги преименуваме G с името G_0 и нека редицата от получените графи е:

$$G_1, G_2, \ldots, G_t$$

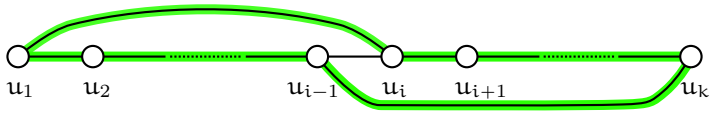
Очевидно G_t е K_n.

Щом G_0 не е Хамилтонов, а G_t е Хамилтонов, съществува i, такова че $1 \leqslant i \leqslant t$, G_{i-1} не е Хамилтонов и G_i е Хамилтонов. С други думи, на i-тата стъпка за първи път получаваме Хамилтонов граф. Тогава, съгласно Наблюдение 22, всички графи G_{i+1}, G_{i+2}, $\ldots$, G_t също са Хамилтонови, а никой от графите G_{i-2}, G_{i-3}, $\ldots$, G_0 не е Хамилтонов. Накратко и неформално, преходът от $i - 1$ към i е преход от не-Хамилтоновост към Хамилтоновост.

Очевидно е, че щом Хамилтоновият граф G_i се получава от не-Хамилтоновия G_{i-1} чрез добавяне на едно единствено ребро, в G_{i-1} има Хамилтонов път, чиито краища не са съседи в G_{i-1}, и въпросното ребро се добавя именно между тях. Също така очевидно е, че за графа G_{i-1} е изпълнено $\forall u, v \in V : (u, v) \notin E(G_{i-1}) \rightarrow d_{G_{i-1}}(u) + d_{G_{i-1}}(v) \geqslant n$, защото тези неравенства няма как да се нарушат при добавяне на ребра—и оттам, ненамаляване на степените на върховете—към първоначалния граф $G = G_0$.

Това, което ще опровергаем е, че съществува не-Хамилтонов граф[†] H, който съдържа Хамилтонов път, не е Хамилтонов, и сумата от степените на всеки два несъседни върха е поне n. Нека съществува такъв граф H. Нека Хамилтоновият път в него е $p = u_1, u_2, \ldots, u_n$. Нека $d_H(u_1) = k$. Очевидно $N(u_1) \subseteq V(p)$, защото $V(p) = V(H)$ по дефиниция. Нещо повече, $u_1 \notin N(u_1)$ и $u_k \notin N(u_1)$—ако $u_k \in N(u_1)$, щеше да има и Хамилтонов цикъл—така че $N(u_1)$ са вътрешни върхове в p. Твърдим, че $\forall u_i \in N(u_1) : u_{i-1} \notin N(u_k)$. Допускането на противното директно води съществуване на Хамилтонов цикъл с конструкция, напълно аналогична на конструкцията от Теорема 16:

[†]В случая, G_{i-1} е такъв граф.



Тогава $V(p)$ съдържа следните три множества върхове като подмножества: върховете от $N(u_k)$, върховете вляво от върховете от $N(u_1)$, и връх u_k. Две по две тези множества имат празно сечение, и сумата от мощностите им е поне $n+1$, понеже $|N(u_1)|+|N(u_k)| \geqslant n$. Тогава $|p| \geqslant n+1$, което е невъзможно. □

Заслужава да се отбележи, че Теорема 16 може да се докаже елементарно чрез Теорема 17: ако $\forall v \in V : d(v) \geqslant \left\lceil \frac{n}{2} \right\rceil$, то $\forall u, v \in V : (u,v) \notin E \to d(u) + d(v) \geqslant n$.

В сила е и следната теорема, която дава горна граница $\binom{n-1}{2} + 1$ за m, такава че ако m е над нея, гарантирано графът е Хамилтонов. Както ще видим след малко, границата е точна. Авторът на теоремата не е известен на автора на тези лекционни записки.

Теорема 18: Точна долна граница за m в гарантирано Хамилтонов граф

Нека $G = (V, E)$ е граф, такъв че $n \geqslant 3$ и $m > \binom{n-1}{2} + 1$. Тогава G е Хамилтонов.

Доказателство: Да допуснем противното. Ако G е пълен граф, той би бил Хамилтонов. Така че G не е пълен. Тогава в G има поне една двойка несъседни върхове x и y. Тъй като x и y не са съседи, $|N(x) \cup N(y)| = |N(x)| + |N(y)|$. Нека $H = G - x - y$. Очевидно, $m = |E(H)| - |N(x)| - |N(y)|$, следователно:

$$|E(H)| > \binom{n-1}{2} + 1 - |N(x)| - |N(y)|$$

За графа H знаем, че има $n-2$ върха. Тогава, от най-общи съображения, той има не повече от $\binom{n-2}{2}$ ребра. Тоест, $|E(H)| \leqslant \binom{n-2}{2}$. Тогава:

$$\begin{aligned}
\binom{n-2}{2} &> \binom{n-1}{2} + 1 - |N(x)| - |N(y)| \ \leftrightarrow \\
\frac{(n-2)(n-3)}{2} &> \frac{(n-1)(n-2)}{2} + 1 - |N(x)| - |N(y)| \ \leftrightarrow \\
n^2 - 5n + 6 &> n^2 - 3n + 2 + 2 - 2|N(x)| - 2|N(y)| \ \leftrightarrow \\
-2n &> -2 - 2|N(x)| - 2|N(y)| \ \leftrightarrow \\
|N(x)| + |N(y)| &> n - 1 \ \leftrightarrow \\
|N(x)| + |N(y)| &\geqslant n
\end{aligned}$$

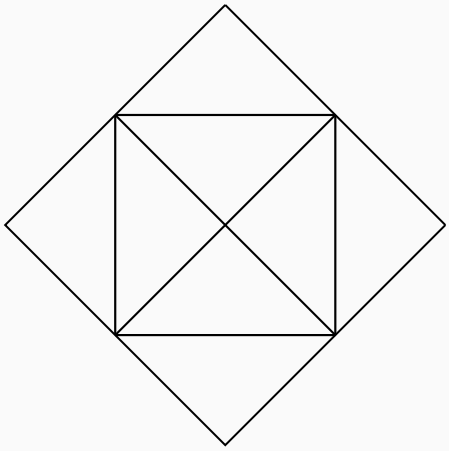
Показахме, че за **произволни** два несъседни върха сумата от степените им е поне n. Тук става дума за първоначалния граф G; H беше само помощна конструкция. Щом това е в сила за произволни два несъседа, то то е в сила за всяка двойка несъседи. Прилагаме Теорема 17 и получаваме, че G е Хамилтонов, противно на направеното допускане. □

И накрая ще покажем, че $\binom{n-1}{2}+1$ е точна горна граница за броя на ребрата на не-Хамилтонов граф. С други думи, че съществува не-Хамилтонов граф с толкова ребра. Такъв граф наистина има, например един K_{n-1} плюс още един връх плюс още едно ребро между новия връх и кой да е връх от старите.

2.10 Ойлерови пътища и цикли

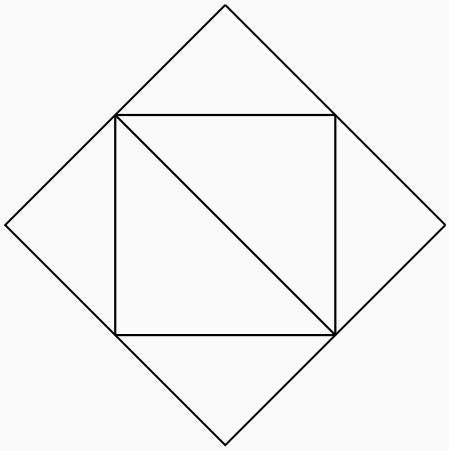
Ще започнем с една занимателна задача. Можете ли да нарисувате това, което е показано на Фигура 2.64, наведнъж; тоест, без да вдигате молива от листа и без да повтаряте линии?

Фигура 2.64 : Не може да се нарисува наведнъж.



Отговорът е, че не можете. А можете ли да нарисувате наведнъж това, което е показано на Фигура 2.65?

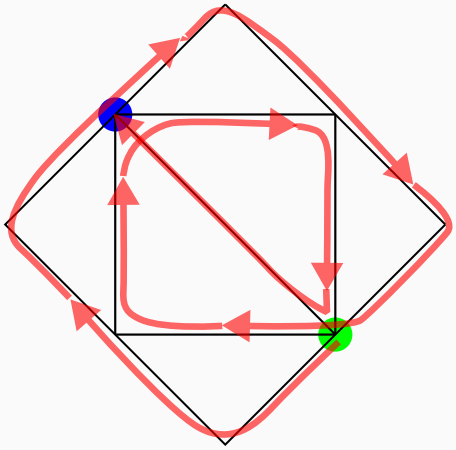
Фигура 2.65 : Може да се нарисува наведнъж.



Това вече може да се нарисува наведнъж. Вижте Фигура 2.66: започвайки от зелената точка рисуваме външния квадрат, после пак от зелената точка рисуваме вътрешния, завъртян на

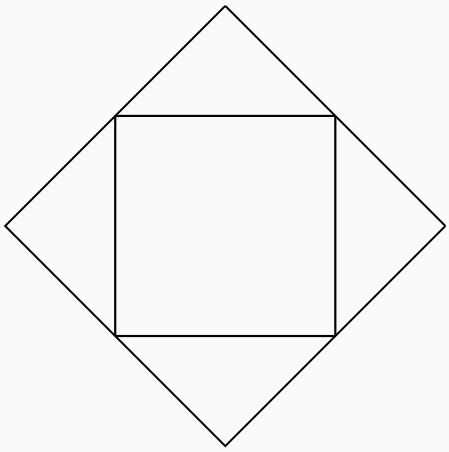
45 градуса квадрат, и накрая рисуваме диагонала от зелената до синята точка. Лесно се вижда, че фигурата може да се нарисува наведнъж само ако започнем от зелената точка и завършим в синята, или обратното. По-долу ще дадем формално доказателство на това наблюдение.

Фигура 2.66 : Как да нарисуваме наведнъж Фигура 2.65.



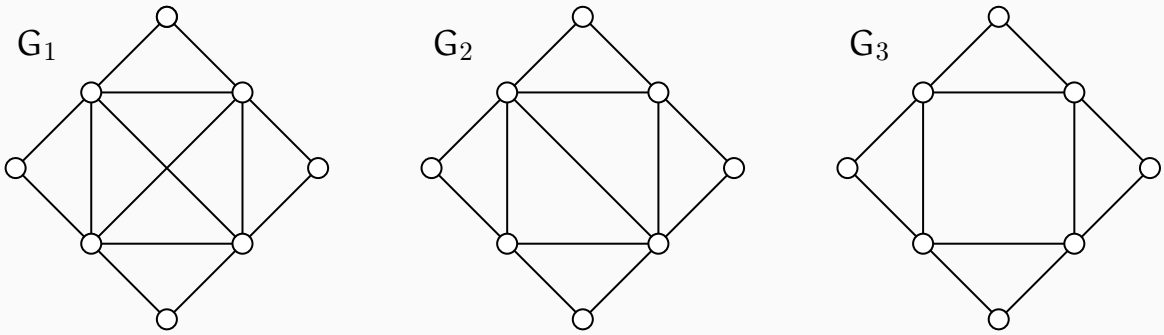
Ако се опитаме да нарисуваме Фигура 2.65 наведнъж, като освен това завършим в тази точка, която сме започнали, ще установим, че това е невъзможно (което следва веднага от факта, че трябва да започнем в зелената точка и да завършим в синята, или обратно). Но ако премахнем диагонала на вътрешния квадрат, получавайки това, което е показано на Фигура 2.67, с лекота можем да нарисуваме обекта наведнъж със затворена линия (тоест, да вдигнем молива от точката, от която сме започнали).

Фигура 2.67 : Може да се нарисува наведнъж като затворена крива.



Тази задача може да се моделира с граф. Отсечките са ребрата, а върховете са точките на пресичане на отсечките. Фигура 2.68 показва графите, съответстващи на трите фигури, които току-що разгледахме (Фигури 2.64, 2.65 и 2.67).

Фигура 2.68 : Графите, съответстващи на Фиг. 2.64, 2.65 и 2.67.



Веднага се вижда, че G_1 има четири върха от нечетна степен, G_2 има два върха от нечетна степен, а G_3 няма върхове от нечетна степен. Както ще видим след малко, наличието на върхове от нечетна степен, техният брой и свързаността на графа определят еднозначно това дали фигурата може да се нарисува наведнъж и дали това може да стане със затворена линия или не.

Оттук нататък в тази секция G е неориентиран свързан мултиграф.

Определение 42: Ойлеров цикъл и Ойлеров път

Ойлеров цикъл в G е цикъл, не непременно прост, който съдържа всяко ребро точно веднъж. *Ойлеров път* в G е път, не непременно прост, който съдържа всяко ребро точно веднъж. G е *Ойлеров*, ако има Ойлеров цикъл.

Съгласно нашите дефиниции на "път" и "цикъл" (Определение 16 или Определение 18), цикъл е частен случай на път, така че Ойлеров цикъл се явява частен случай на Ойлеров път в Определение 42.

Припомнете си Конвенция 6 на стр. 31. Тя е безсмислена за Ойлерови цикли и пътища! Забележете, че ако G има Ойлеров цикъл c или Ойлеров път p, то, ако G няма изолирани върхове, $(V(c), E(c))$ или $(V(p), E(p))$ съвпада с G. Поради това разглеждаме Ойлеровите цикли и пътища като алтернириращи редици от върхове и ребра, точно както се казва в Определение 16 и Определение 18.

Теорема 19: Ойлеров цикъл в свързан мултиграф

G има Ойлеров цикъл тогава и само тогава, когато всеки връх има четна степен.

Доказателство, I: Първо да допуснем, че G има Ойлеров цикъл c. Ще покажем, че G има върхове само от четна степен. Разглеждаме произволен $u \in V(G)$. Очевидно $u \in V(c)$.

Да си представим c като кръгова, а не линейна, наредба. Определение 18 говори за цикъла като за линейна наредба, в която първият и последният връх съвпадат, което съвпадане

задава цикличността. Може да си мислим обаче за всеки цикъл с ненулева дължина като за истинска **кръгова наредба**, в която цикличността е естествена; тоест алтерниращата редица от върхове и ребра, по равен брой от всеки вид, е записана върху окръжност. Всяка поява на u в c отговаря на точно две ребра – това са съседните елементи на u в c.

Ако u няма примки, няма как два съседни върха в c (естествено, между тях в c има ребро) да са u. Следователно, на всяка поява на u в c съответстват точно две ребра—съседните елементи на u в цикъла—като за всеки две различни появи на u, двете двойки ребра нямат общ елемент. Следователно, множеството от всички тези двойки ребра, върху всички появи на u в c, е точно $\mathcal{I}(u)$. Нека u е появява точно t пъти в c. Тогава $|\mathcal{I}(u)| = 2t$. Имайки предвид, че $d(u) = |\mathcal{I}(u)|$, заключаваме, че степента на u е четно число.

Да разгледаме по-общия случай, в който u има примки. Нека u има q примки e_1,e_2, $\ldots$, e_q. Нека u се появява точно t пъти в c, както в предния случай. Очевидно, $t \geqslant q$. При наличието на примки на u има съседни появи на u в c (естествено, с ребро между тях, което ребро е някоя от примките). По-точно казано, в c има точно q подредици $u\,e_i\,u$, $1 \leqslant i \leqslant q$ (по една за всяка примка на u), като някои от тези подредици може да имат общ край u. Всяка такава триелементна подредица, отговаряща на дадена примка, има принос $+2$ към степента на u. Максималните по включване подредици са от вида $u\,e_{j_1} \cdots e_{j_r}\,u$, където $e_{j_1}, \ldots, e_{j_r} \in \{e_1, \ldots, e_q\}$, са точно $t - q$ на брой. Нека E' е множеството от всички ребра, които са вляво и вдясно от всяка от тези подредици. E' е точно множеството от ребрата, инцидентни с u, които не са примки. Тъй като тези максимални по включване подредици никога не са съседни, в смисъл, че между всеки две от тях в c има поне един връх, който не е u, очевидно $|E'| = 2(t-q)$. Имайки предвид, че $d(u)$ е сумата от $|E'|$ и $2q$ (вж. Определение 15), заключаваме, че $d(u)$ е четно число.

Доказателство, II: Сега допускаме, че G е свързан и всеки връх е от четна степен. Ще докажем съществуването на Ойлеров цикъл конструктивно: ще покажем алгоритъм, който строи Ойлеров цикъл c във всеки свързан граф с върхове само от четни степени. Този алгоритъм е много стар. Измислен е от Carl Hierholtzer през 19 век [12, стр. 10], като Hierholtzer очевидно не е бил запознат с изследванията на Euler върху съществуването на такива цикли през 18 век. Идеята на алгоритъма е проста. В началото всяко ребро е *неизползвано*, а c е празен (празната редица). Използваме *временен цикъл* s и *текущ връх* u (u е променлива от тип връх, а не фиксиран връх от графа). Започвайки от произволен връх x, който е инцидентен с поне едно неизползвано ребро (в началото всички върхове са такива), инициализираме $u \leftarrow x$ и $s \leftarrow x$. После изпълняваме, докато е възможно, следното: избираме произволно неизползвано ребро $e \in \mathcal{I}(u)$, маркираме e като *използвано* и "преминаваме" през e; тоест, ако другият край на e е v, добавяме e и v към s като $s \leftarrow s, e, v$; текущият връх u става v, тоест правим $u \leftarrow v$.

Правим това, докато можем; тоест, докато не стигнем до текущ връх u, който няма инцидентно необходено ребро. Тъй като ребрата са краен брой и ние променяме статусите на ребрата само от неизползвани в използвани, рано или късно ще се окажем във връх u, който няма неизползвани инцидентни ребра. В този момент u е точно този връх x, от който започнахме обхождането (това ще докажем по-долу формално). Ерго, в този момент s е цикъл. *Вмъкваме* s в c (по-долу ще уточним какво точно значи това). Ако всички ребра са използвани, връщаме c и терминираме алгоритъма. В противен случай продължаваме така: тъй като G е свързан, в c задължително има връх b, който е инцидентен с поне едно необходено ребро. Тогава присвояваме $u \leftarrow b$, $s \leftarrow b$ и отново изпълняваме итерираното добавяне на ребра и върхове към s, докато можем.

Формално, алгоритъмът може да се запише така. ϵ означава празната редица. Б.о.о., допускаме, че мултиграфът има ребра (инак той би бил един единствен изолиран връх).

Алгоритъм 2: Алгоритъм на Hierholtzer

Вход: свързан мултиграф G с поне едно ребро.
Изход: Ойлеров цикъл c в G.
$\mathfrak{u}$ и $\mathfrak{x}$ са променливи тип връх, а $\mathfrak{s}$ е променлива тип цикъл.

❶ $\mathsf{c} \leftarrow \epsilon$. Маркирай всички ребра на G като неизползвани.

❷ Избери произволен връх $\mathfrak{a}$, инцидентен с неизползвано ребро.

❸ Присвои $\mathfrak{x} \leftarrow \mathfrak{a}$, $\mathfrak{u} \leftarrow \mathfrak{a}$, $\mathfrak{s} \leftarrow \mathfrak{a}$.

❹ Ако $\mathfrak{u}$ има поне едно инцидентно неизползвано ребро $e = (\mathfrak{u}, \mathfrak{v})$:
① присвои $\mathfrak{s} \leftarrow \mathfrak{s}, e, \mathfrak{v}$ и маркирай e като използвано.
② присвои $\mathfrak{u} \leftarrow \mathfrak{v}$ и иди на ❹.

❺ В противен случай, вмъкни $\mathfrak{s}$ в c.

❻ Ако няма неизползвани ребра, върни c и прекрати алгоритъма.

❼ В противен случай, избери произволен връх $\mathfrak{b}$ от c, инцидентен с неизползвано ребро.

❽ Присвои $\mathfrak{x} \leftarrow \mathfrak{b}$, $\mathfrak{u} \leftarrow \mathfrak{b}$, $\mathfrak{s} \leftarrow \mathfrak{b}$ и иди на ❹.

Едно доуточнение. “Вмъкни $\mathfrak{s}$ в c” на ред ❺ означава следното:

- ако c е празен, то присвои $\mathsf{c} \leftarrow \mathfrak{s}$,
- в противен случай c съдържа поне една поява на върха $\mathfrak{y}$, който е начало и край на $\mathfrak{s}$; замени коя да е поява на $\mathfrak{y}$ в c с редицата $\mathfrak{s}$.

Формално и пълно доказателство за коректност на този итеративен алгоритъм би било твърде дълго и извън основната тема на тези лекционни записки. Алгоритъмът има два вложени цикъла—външният строи c, вътрешният строи $\mathfrak{s}$—така че би трябвало с отделни инварианти да се докаже коректността на вътрешния цикъл и после на външния. Тук само ще скицираме основното от доказателството за коректност. Едно пояснение преди това: променливата $\mathfrak{x}$ е излишна от гледна точка на работата на алгоритъма. Тя е въведена с цел по-проста аргументация за коректност. $\mathfrak{x}$ съдържа началната стойност на $\mathfrak{u}$ при всяка серия от итерации на цикъла ❹.

Ключовото, макар и напълно очевидно, наблюдение е следното.

> Всяко изпълнение на ❹ променя статуса на точно едно ребро (а именно e) от неизползвано на използвано, с което намалява с единица броя на инцидентите неизползвани ребра **на точно два върха**, а именно на $\mathfrak{u}$ и на $\mathfrak{v}$.

Веднага следва, че докато се “върти” цикълът ❹ и връх $\mathfrak{v}$ (ред ①) е различен от $\mathfrak{x}$ е вярно, че точно два върха в G, а именно $\mathfrak{x}$ и $\mathfrak{v}$, имат нечетен брой инцидентни неизползвани ребра. На ред ② $\mathfrak{u}$ получава стойност $\mathfrak{v}$, така че при следващото достигане на ❹:

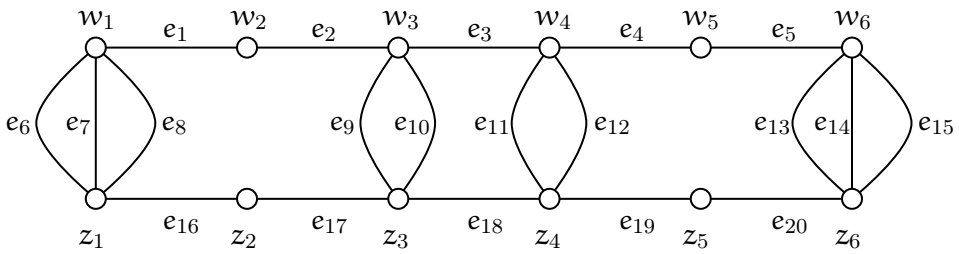
- ако $\mathfrak{u} \neq \mathfrak{x}$, то $\mathfrak{u}$ и $\mathfrak{x}$ са точно тези върхове, които имат нечетен брой инцидентни неизползвани ребра; в този случай $\mathfrak{u}$ има поне едно инцидентно неизползвано ребро и итерарането на ❹ продължава;

- ако $u = x$, то всички върхове в G имат четен брой инцидентни неизползвани ребра; ако u има нула инцидентните неизползвани ребра, итерирането спира и изпълнението отива на ❺, в противен случай итерирането на ❹ продължава.

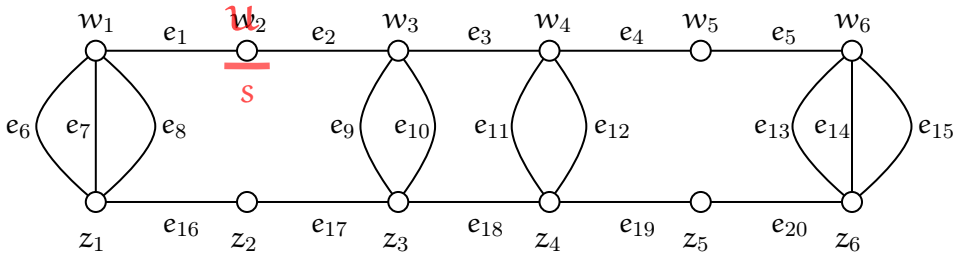
От това следва, че при последното достигане на ❹ е изпълнено $u = x$, а от това следва, че при всяко достигане на ❺, променливата u съдържа точно x – върха, от който започна изпълнението на вътрешния цикъл ❹. От това следва, че на ред ❺ наистина s е цикъл. Щом s е цикъл, то вмъкването на s в c е добре дефинирано, независимо от това дали c е празна редица или не.

Ако при достигането на ред ❻ не са останали необходени ребра, очевидно c е Ойлеров цикъл, така че алгоритъмът наистина връща Ойлеров цикъл. В противен случай, изпълнението отива на ред ❼. Щом има неизползвани ребра, то задължително поне един връх от текущия цикъл c е инцидентен с неизползвано ребро (допускането на противното веднага влече, че G не е свързан), така че връх b е добре дефиниран. □

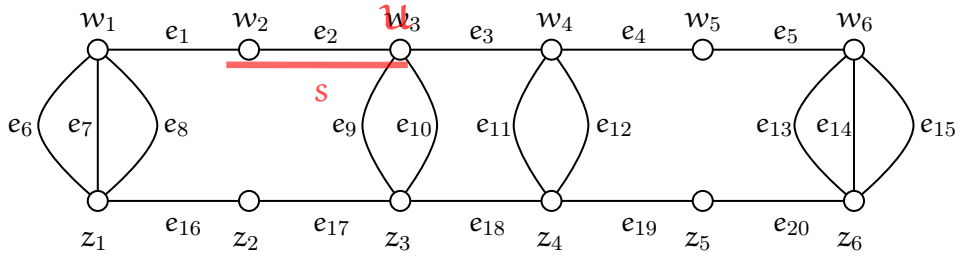
Ще илюстрираме работата на алгоритъма на Hierholzter върху следния мултиграф. Ще игнорираме променливата x, понеже, както споменахме, тя имаше смисъл само за по-ясна аргументация за коректност.



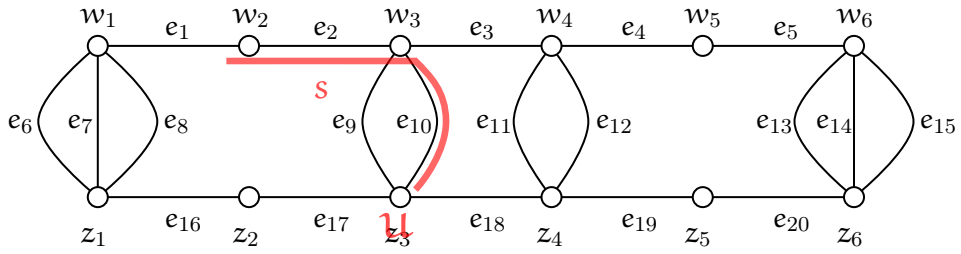
Първоначално c е празен, а всички ребра са неизползвани (ред ❶). На ред ❷ избираме произволен връх a. Да кажем, че избираме w_2. На ред ❸ присвояваме $u \leftarrow w_2$ и $s \leftarrow u$. Да си представим s като път в мултиграфа (s ще стане цикъл, но в процеса на работата s е път, който не е непременно цикъл), който нараства само в единия си край и този край е върхът, който се съдържа в променливата u.



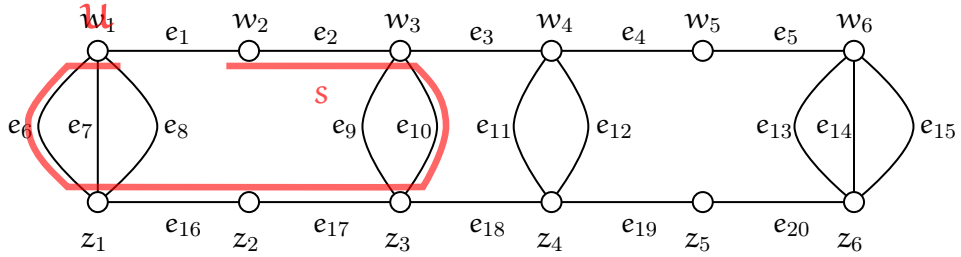
Изпълнението отива на ред ❹. u има инцидентни неизползвани ребра. Да кажем, че изберем e_2. Тогава $e = (u, v)$ на ред ❹ е реброто e_2, като $u = w_2$ и $v = w_3$. На ред ① s "нараства" с e_2, w_3, а на ред ② u получава стойност w_3.



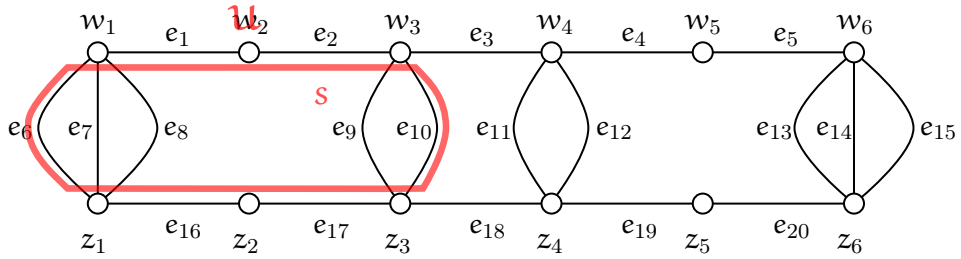
Изпълнението отново е на ред ❹. Да кажем, че избираме e_{10} като e. Тогава $v = z_3$. След изпълнението на ① и ② имаме:



В момента s е пътят $w_2, e_2, w_3, e_{10}, z_3$. Нека през следващите няколко итерации на ❹ s стане $w_2, e_2, w_3, e_{10}, z_3, e_{17}, z_2, e_{16}, z_1, e_6, w_1$:



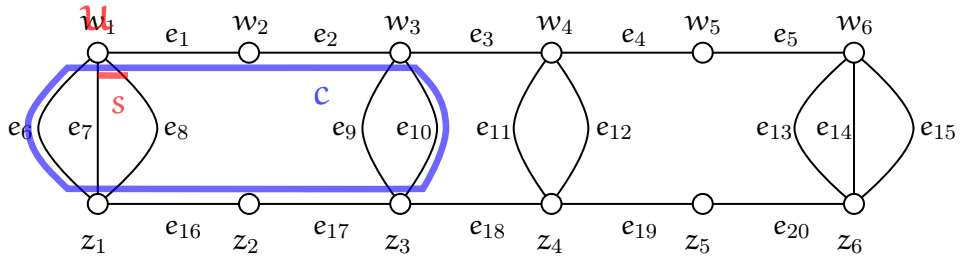
Сега изборът за e е измежду e_7, e_8 и e_1. Да кажем, че изберем e_1. Тогава следващата стойност на u е w_2, върхът, от който тръгнахме. Нещо повече, текущият u няма инцидентни необходени ребра и изпълнението на ❹ спира. Както виждаме на илюстрацията, в момента s е цикъл.



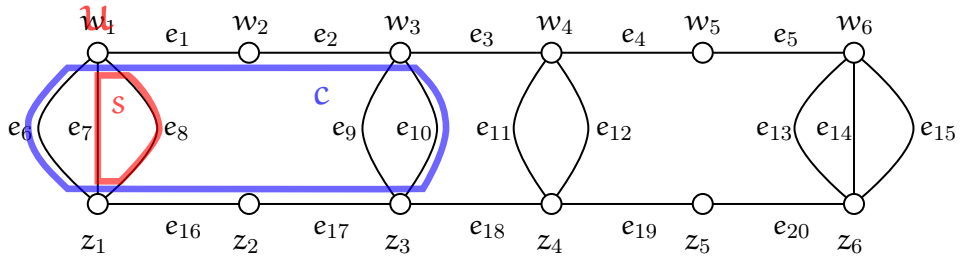
Вмъкваме s в c (ред ❺). Тъй като c е празен, това означава, че c приема стойността на текущия s. Тоест,

$$c = w_2, e_2, w_3, e_{10}, z_3, e_{17}, z_2, e_{16}, z_1, e_6, w_1, e_1, w_2$$

Тъй като има неизползвани ребра, ред ❻ не се изпълнява. Изпълнява се ред ❼, където избираме w_1 като връх b. Присвояваме $u \leftarrow w_1$ и $s \leftarrow w_1$ и отиваме на ред ❹. Нещата изглеждат така:



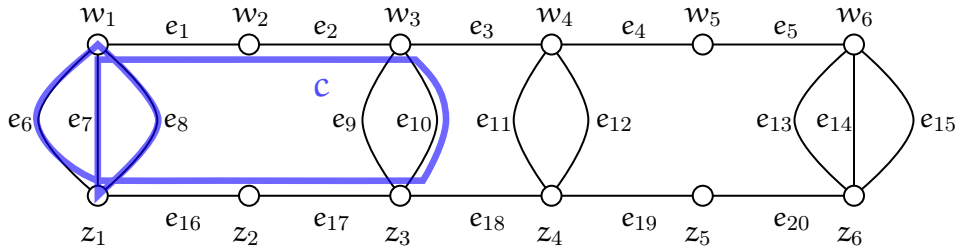
След две изпълнения на цикъла ❹ имаме $s = w_1, e_8, z_1, e_7, w_1$:



Текущият u отново е w_1 и s е цикъл. u няма инцидентни необходени ребра и изпълнението отива на ред ❺. Вмъкваме s в c. Тъй като c не е празен, вмъкването става, като заменяме някоя поява на w_1—върхът, с който започва и завършва s—в c с целия s. Получаваме:

$$c = w_2, e_2, w_3, e_{10}, z_3, e_{17}, z_2, e_{16}, z_1, e_6, w_1, e_8, z_1, e_7, w_1, e_1, w_2$$

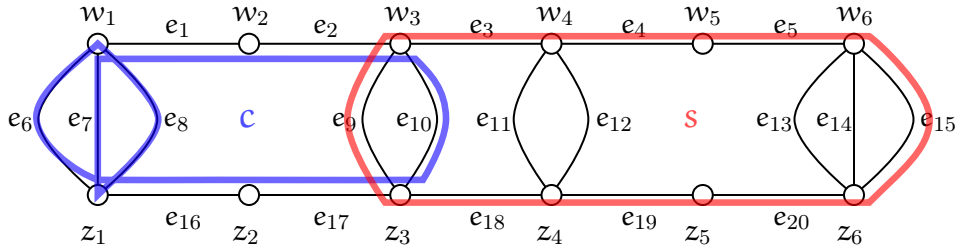
Нагледно, резултатът от влагането изглежда така:



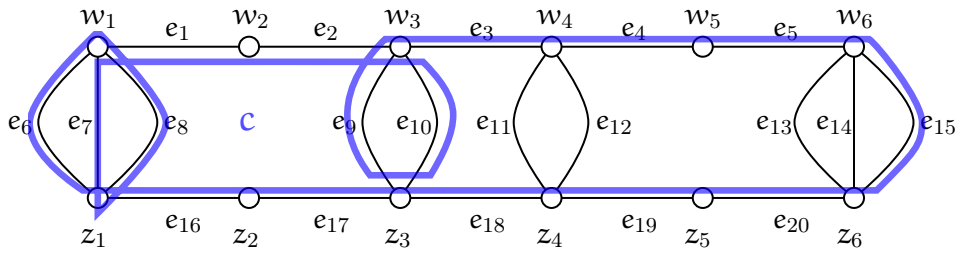
Тъй като има неизползвани ребра, алгоритъмът пордължава. Да кажем, че следващото изпълнение на ❹ започва от w_3 и след приключването му имаме

$$s = w_3, e_3, w_4, e_4, w_5, e_5, w_6, e_{15}, z_6, e_{20}, z_5, e_{19}, z_4, e_{18}, z_3, e_9, w_3$$

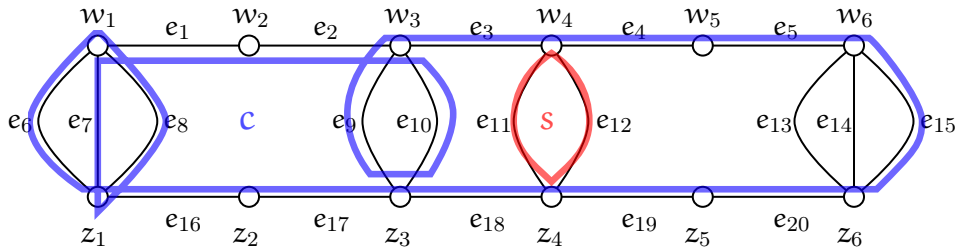
Нагледно, нещата изглеждат така:



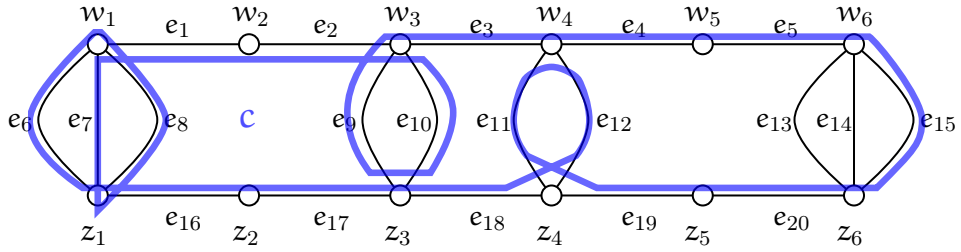
Влагаме s в c:



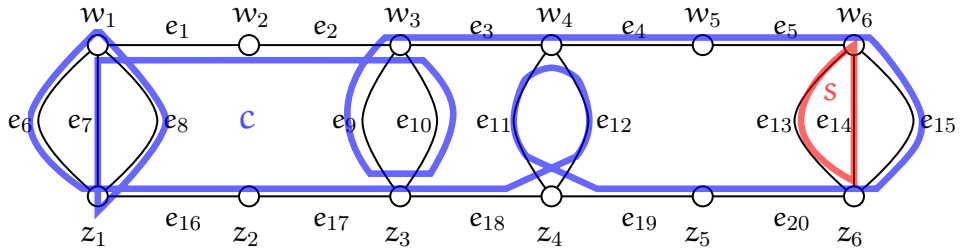
Все още има неизползвани ребра. При още едно "завъртане" на цикъла ❹ получаваме:



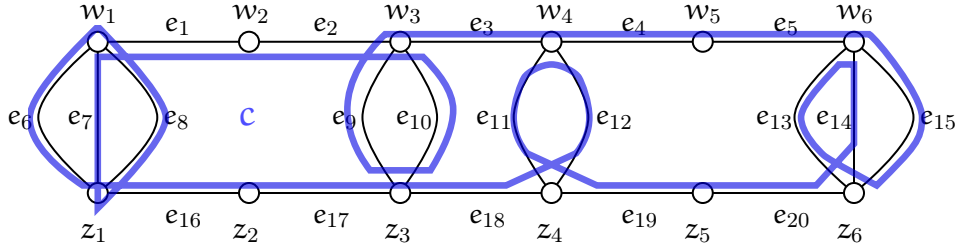
Влагаме s в c:



Все още има необходени ребра. Пак построяваме s:



Влагаме s в c:



Сега вече няма неизползвани ребра и алгоритъмът връща c.

Има и друг известен алгоритъм за построяване на Ойлерови цикли: алгоритъмът на Fleury. Тук няма да го описваме. Той е описан, например, в учебника на Bondy и Murty [13, стр. 86].

Теорема 20: Ойлеров път, който не е цикъл, в свързан мултиграф

G има Ойлеров път, който не е цикъл, тогава и само тогава, когато точно два върха са от нечетна степен.

Доказателство, I: Нека G има път p, който съдържа всяко ребро точно веднъж и има различни краища u и v. Ще покажем, че $d(u)$ и $d(v)$ са нечетни, а всички върхове освен u и v имат четни степени.

Имаме $p = u, \cdots, v$. Добавяме едно ново ребро e между u и v, получавайки мултиграф G'. Очевидно G' има Ойлеров цикъл c, състоящ се от p плюс новото ребро e:

$$c = p, e, u$$

Съгласно Теорема 19, в G' всички върхове са от четна степен. Очевидно $d_{G'}(u) = d_G(u) + 1$, $d_{G'}(v) = d_G(v) + 1$ и $\forall x \in V(G) \backslash \{u, v\} : d_{G'}(x) = d_G(x)$. Веднага следва, че $d_G(u)$ и $d_G(v)$ са нечетни, а останалите върхове в G са с четни степени.

Доказателство, II: Нека в G има точно два върха от нечетна степен. Ще покажем, че в G има Ойлеров u-v път.

Добавяме едно ново ребро e между u и v, получавайки мултиграф G''. Очевидно в G'' всички върхове са от четна степен. Съгласно Теорема 19, в G'' има Ойлеров цикъл c. Изтривайки e от c, получаваме Ойлеров u-v път. □

Допълнение 13: Хамилтонови цикли, Ойлерови цикли и линейни графи

Задачата да се намери дали даден граф е Хамилтонов или не, е изключително трудна. Както вече споменахме в Допълнение 12, тя е **NP**-трудна. От друга страна, задачата дали даден граф е Ойлеров или не, е алгоритмично лесна – Алгоритъм 2 решава тази задача бързо дори за огромни графи.

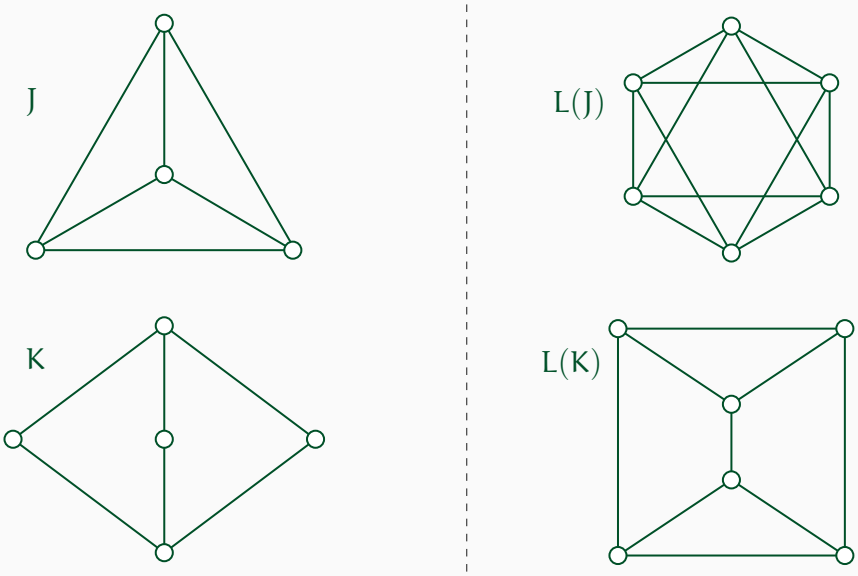
Да разсъждаваме върху следното. Нека е даден граф G. Искаме да разберем дали G е Хамилтонов. Да допуснем, че G е линеен граф (Определение 38). Това означава, че $G = L(H)$ за някакъв граф H. Ако H е Ойлеров, то G е Хамилтонов; нещо повече, Ойлеровият цикъл в H директно отговаря на Хамилтоновия цикъл в G. Но въпросът дали G е Хамилтонов не е същият като въпросът дали H е Ойлеров. Следната теорема от [32] хвърля светлина върху всичко това.

Теорема 21: Теорема 8.8 в [32, стр. 80] за линейните графи

Ако G е Ойлеров, то $L(G)$ е и Ойлеров, и Хамилтонов. Конверсното твърдение не е вярно. Ако G е Хамилтонов, то и $L(G)$ е Хамилтонов. Конверсното твърдение не е вярно.

Че нито едно от конверсните твърдения не е истина се вижда от Фигура 2.69 (взета от Фигура 8.9 на стр. 80 в [32]).

Фигура 2.69 : Ойл. и Хам. цикли в графи и линейните им графи.



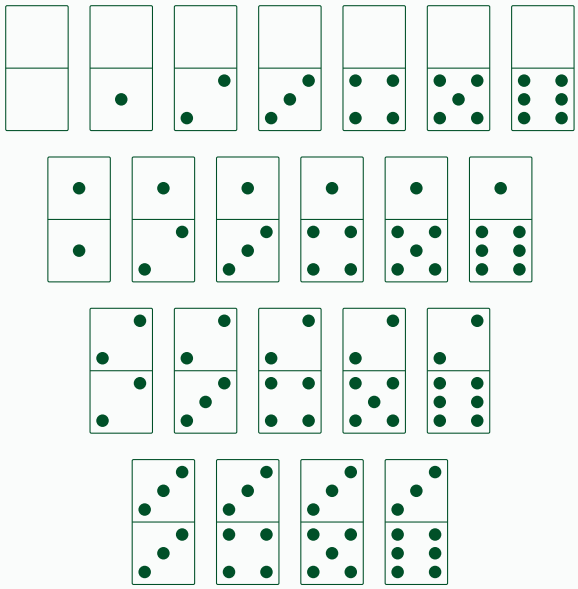
J не е Ойлеров и K не е нито Хамилтонов, нито Ойлеров.

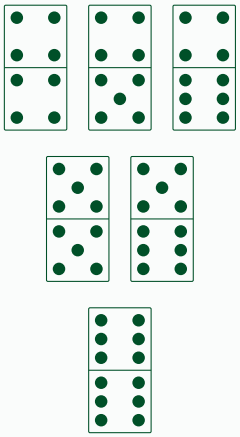
L(J) е Хамилтонов и Ойлеров и L(K) е Хамилтонов.

Заключаваме, че дори G да е линеен граф, задачата дали G е Хамилтонов не се свежда до задачата дали H е Ойлеров, където H е граф, такъв че $G = L(H)$.

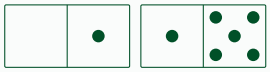
Допълнение 14: Домино, Хамилтонови цикли и Ойлерови цикли

Играта *домино* се играе с 28 плочки. Всяка плочка е правоъгълник с размери 1×2 и е разделена на два квадрата 1×1. Върху квадратите има точки, чиито брой варира от 0 до 6. Всички 28 плочки са показани тук:

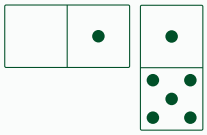




Когато плочките се редят в редица, те се слагат една до друга така, че страна на квадрат на едната плочка е долепена точно до страна на квадрат на другата плочка, като двата квадрат имат един и същи брой точки. Примерно така:



Допустимо е единият от правоъгълниците да е под прав ъгъл спрямо другия:



По този начин може да строим *редица от плочки*, при която всяка следваща плочка е долепена така до предишната, че квадратите, които се оказват един до друг, имат един и същи брой точки. *Затворена редица от плочки* е редица, в която последният квадрат на последната плочка е долепен по първия квадрат на първата плочка и за тези два квадрата също е вярно, че имат един и същи брой точки.

Иска се да се докаже, че е възможно да се направи затворена редица от всички плочки. Има елегантно доказателство, използващо теорията на графите. Очевидно доказателството ще покаже, че има някакъв цикъл в някакъв граф, но какъв е графът и за какъв цикъл става дума?
Сякаш най-естествено е върховете на графа да са плочките, а ребро между две плочки да се слага тстк двете плочки имат квадратче с еднакъв брой точки. Задачата се свежда до това, да се докаже, че в този граф има Хамилтонов цикъл.
Има обаче и друг начин да построим графа. Нека върховете са числата от 0 до 7. Те отговарят на възможностите за точки в едно квадратче. Нека ребрата са плочките в смисъл, че за всяка плочка (x, y), където $x, y \in \{0, \ldots, 7\}$, слагаме ребро с краища връх x и връх y. Забележете, че всеки връх получава примка, като примките съответстват на плочките от вида (x, x), за $0 \leqslant x \leqslant 7$. Задачата е дали има Ойлеров цикъл в този граф. Ойлеров цикъл има, защото графът е пълният граф на 7 върха, като всеки връх е с примка. Тогава всеки връх е от степен 8, което е четно число (всяка примка допринася две, а не едно, към степента на своя връх!). Графът очевидно е и свързан, така че съгласно Теорема 19, в него има Ойлеров цикъл. Алгоритъмът на Hierholtzer (Алгоритъм 2) построява Ойлеров цикъл, който директно ни "казва" как да подредим плочките на доминото в затворена редица.

2.11 Дървета

2.11.1 Определение

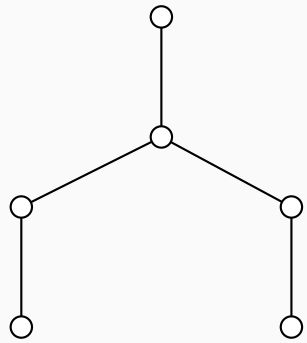
Определение 43: Дърво, не-индуктивна дефиниция.

Дърво е всеки граф, който е свързан и ацикличен.

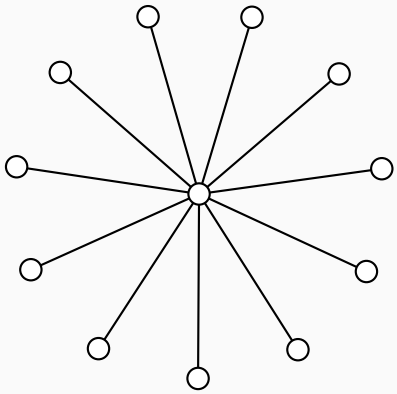
На английски терминът е *tree*.

Графът, показан на Фигура 2.1, не е дърво, защото не е свързан и има цикли. Дървета (като неименувани графи) са изобразени на Фигура 2.70.

Фигура 2.70 : Два графа, които са дървета.



Дърво с 6 върха.



Дърво с 12 върха.

Следното определение на "дърво" е еквивалентно на Определение 44.

Определение 44: Дърво, индуктивна дефиниция.

Множеството от дърветата се дефинира така:

❶ **База** Всеки тривиален граф е дърво.

❷ **Индуктивна стъпка** Ако $\mathsf{T} = (\mathsf{V}, \mathsf{E})$ е дърво и u е връх в T и w е връх, който не е в T, то $\mathsf{T}' = (\mathsf{V} \cup \{w\}, \mathsf{E} \cup \{(u, w)\})$ е дърво.

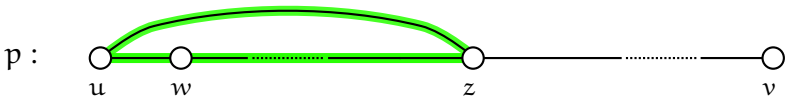
Определение 44 задава процедура[†] за генериране на графи. Започвайки от един единствен връх (базовото множество) и използвайки присъединителната операция добавяне на нов връх, когото свързваме към точно един от вече съществуващите върхове (с точно едно ребро), ние генерираме безкрайно множество графи. Това множество графи са точно дърветата, факт, който сега ще докажем. Първо ще докажем следната лема. Да си припомним определението на висящ връх на стр. 9: това е връх от степен едно.

[†]Терминът "алгоритъм" не бил удачен, защото алгоритмите трябва да терминират, докато тази процедура работи безкрайно.

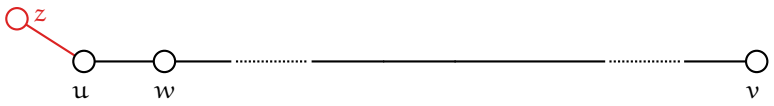
Лема 8: Наличие на висящи върхове в дърветата.

Нека $G = (V, E)$ е дърво съгласно Определение 43, такова че $n \geqslant 2$. Тогава G има поне два висящи върха.

Доказателство: Разглеждаме произволен най-дълъг път p в G. Очевидно $|p| \geqslant 1$. Нека u и v са краищата на p. Ще докажем, че и u, и v са висящи върхове. Без ограничение на общността, ще докажем това само за u. Връх u има поне един съсед w: това е върхът, който е до него в p. Ще докажем, че u не може да има други съседи. Да допуснем, че u е съседен с връх z, такъв че $z \neq w$. Ако z също е връх от p, то очевидно в графа има цикъл, както е показано на следната фигура:



Но в дърветата цикли няма по дефиниция. Ако връх z не е връх от p, която ситуация е илюстрирана на следната фигура:



то тогава в G има път, по-дълъг от p с единица – а именно пътят, който се получава от p, следван от реброто, изобразено в червен цвят, последвано от връх z, също нарисуван в червено. Това противоречи на допускането, че p е най-дълъг път.

И така, допускането, че u не е висящ връх, води до противоречие. □

Теорема 22: Глобалната и индуктивната дефиниции на "дърво" са еквивалентни

Определение 43 и Определение 44 са еквивалентни.

Доказателство, I: Ще докажем, че всеки граф, генериран от процедурата на Определение 44, е свързан и ацикличен. Тъй като Определение 44 е индуктивно, ще направим доказателството със структурна индукция точно по него.

Базовият случай е базовият случай от определението. Разглеждаме граф $(\{u\}, \varnothing)$. Очевидно той е свързан и няма цикли. ✓

Допускаме, че $T = (V, E)$ от индуктивната стъпка е свързан и ацикличен. Ще докажем, че полученият T' е свързан и ацикличен. Първо ще докажем, че T' е свързан. За да бъде свързан, трябва за всеки два върха $x, y \in V(T')$ да е вярно, че има път между тях. Но $V(T') = V \cup \{w\}$. Разглеждаме три случая.

- Нито един от x, y не е връх w. Но тогава е вярно, че $x, y \in V$. Съгласно индуктивното предположение, между всеки два върха в T има път, от което веднага следва, че в T' има път между всеки два върха от V.

- Точно единият от x, y е връх w. Без ограничение на общността, нека това е връх x. Тогава задължително $y \in V$. Но тогава и y, и u са върхове в T. Съгласно индуктивното предположение, в T между тях има път, който ще наречем p. Очевидно p е път между y и u и в T'. Тогава пътят в T', който се получава от слепването на p, реброто (u, w) и връх w, е път между y и $x = w$ в T'.

- И $x = y = w$. Тогава очевидно между x и y има тривиален път с дължина 0.

Доказахме, че T' е свързан. Сега ще докажем, че е ацикличен. Забелязваме, че нито един връх от T' не може да бъде връх в цикъл:

- връх w е от степен 1, а всеки връх, който е връх в цикъл, е от степен поне 2;
- останалите върхове на T' са върховете от T, а съгласно индуктивното предположение T е ацикличен и освен това, очевидно добавянето на реброто (u, w) не може да създаде цикъл от върхове на $V(T)$.

Доказахме, че T' е свързан и ацикличен, тоест, дърво.

Доказателство, II: Ще докажем, че всяко дърво съгласно Определение 43 може да бъде конструирано от процедурата от Определение 44. Нека D е произволно дърво съгласно Определение 43. Ако D има точно един връх, то D може да бъде конструиран от базата на Определение 44. В противен случай, съгласно Лема 8, в D има поне един висящ връх u_1. Нека $D_1 = D - u_1$. Ако D_1 има точно един връх то D_1 може да бъде конструиран от базата на Определение 44. В противен случай, съгласно Лема 8, в D_1 има поне един висящ връх u_2. Нека $D_2 = D - u_2$. Ако D_2 има точно един връх то D_2 може да бъде конструиран от базата на Определение 44. И така нататък. Очевидно, при всяко от тези изтривания на висящ връх, графът остава свързан и ацикличен, тоест дърво. Следователно има редица от дървета D, D_1, D_2, ..., D_k за някое k. Ясно е, че това последователно изтриване на върхове може да се прави само краен брой пъти, защото началният D има краен брой върхове. И така, за някое k[†] е вярно, че D_k има точно един връх. Изтритите върхове са u_1, u_2, ..., u_k, в реда на триенето. Тогава графът с точно един връх D_k може да бъде конструиран от базата на Определение 44, а след това с $|V(D)| - 1$ прилагания на индуктивната стъпка добавяме изтритите върхове в обратния ред на изтриването им, като свързваме всеки от тях към точно този връх, който е бил единственият му съсед точно преди изтриването. Очевидно по този начин получаваме D. □

Определение 45: Гора.

Гора е всеки ацикличен граф.

На английски терминът е *forest*. Очевидно това е обобщение на "дърво": всяка гора е дърво, ако е свързан граф. Освен това, свързаните компоненти на всяка гора са дървета[‡]. Всяка гора е обединение на дърветата, които нямат общи върхове помежду си.

2.11.2 Свойства на дърветата

За доказателството на Лема 9 ще използваме Определение 43.

Лема 9

Граф е дърво тогава и само тогава, когато между всеки два негови върха има точно един път.

Доказателство, I: Да разгледаме произволно дърво T и произволни $u, v \in V(T)$. Ще докажем, че има точно един u-v път. Че има поне един u-v път следва от това, че T е свързан.

[†] А именно, $k = |V(D)| - 1$.

[‡] Също както в реалния свят гората се състои от дървета...

Сега ще използваме и факта, че T е ацикличен, за да покажем, че не може да има повече от един u-v път.

Допускаме противното: между u и v има поне два пътя p и q. Връх u се явява край и на p, и на q. Разглеждаме p и q като крайни редици от върхове, като търсим първия връх от u нататък, който е различен за p и q. Такъв трябва да има, иначе p и q биха били един и същи път. Може само в единия от p и q да се среща такъв връх, но трябва да се среща. Без ограничение на общността, нека w е първият връх от u нататък в p, който не се среща в q. Нека a е върхът преди w, в посока от u нататък, в p. С други думи, a е последният от u нататък в p, който е общ за p и q. Нека b е първият връх след a в p, в посока от u нататък, който е връх и в q. Очевидно е, че такъв общ връх трябва да има, понеже p и q имат друг (освен u) общ край, а именно връх v, така че няма как след a, в посока от u нататък, да имат само върхове, които не са общи.

И така, установихме, че в p има подпът p' с краища a и b, който не е подпът на q. Аналогично, в q има подпът q' с краища a и b, който не е подпът на p. Както споменахме, единият от p' и q' може да няма вътрешни върхове[†], но това няма значение. Важното е, че $p' \cup q'$ е цикъл в T, а ние знаем, че в дърветата няма цикли. Следователно, допускането, че има поне два различни u-v пътя, е погрешно.

Доказателство, II: Да разгледаме произволен граф G, в който между всеки два върха има точно един път. Веднага се вижда, че G е свързан.

Ако допуснем, че в G има поне един цикъл, веднага следва, че в G има поне два върха, между които има два различни пътя, което е невъзможно при по-рано направеното допускане, че между всеки два върха има точно един път. Но тогава допускането, че в G съществува цикъл, е грешно. С други думи, G е ацикличен.

Следователно, G е дърво. □

За доказателството на Лема 10 ще използваме Определение 43.

Лема 10

Нека $T = (V, E)$ е дърво и $u, v \in V$ са различни върхове, които не са съседи. Тогава графът $G = (V, E \cup \{(u, v)\})$ има точно един цикъл.

Доказателство: Ще покажем, че G има поне един цикъл. Наистина, u и v са свързани с (един единствен) път p в T съгласно Лема 9, като p съдържа поне един вътрешен връх, понеже u и v не са съседи в T. Нека

$$p = u, e_1, w_1, \ldots, e_k, w_k, e_{k+1}, v$$

като $\{w_1, \ldots, w_k\} \subseteq V$ е множеството от вътрешните върхове на p, $\{e_1, \ldots, e_{k+1}\} \subseteq E$ е множеството от ребрата на p, а $k \geqslant 1$. Нека реброто (u, v) в G бъде наречено e'. Нека $q = (\{u, v\}, \{e'\})$. Тогава $c = p \cup q$ е цикъл в G.

Ще покажем, че в G не може да има повече от един цикъл. Да допуснем, че в G има цикъл c', който е различен от c. Ясно е, че c' също съдържа реброто e', защото и той се е появил с добавянето на e' към T. Тогава c' съдържа и върховете u и v. Следва, че u, e', v е общ подпът за c и c'. Ако "тръгнем" от u през e', v и така нататък в c и c', все някъде трябва да има разлика; трябва има някакъв връх z, общ за c и c', след който в c има ребро $\widehat{e}$, а в c' има ребро $\widetilde{e}$, като $\widehat{e} \neq \widetilde{e}$. Този връх z може да съвпада с v или да не съвпада с v, това няма значение, важното е, има такъв връх, след който следва бифуркация – двата

[†]Не може и p', и q' да нямат вътрешни върхове. Това би било възможно само ако има две различни ребра с краища a и b, а това е възможно само при мултиграфите, каквито в момента не разглеждаме.

цикъла се разделят. Ако нямаше такъв връх, след който се разделят, те биха били един и същи цикъл. От друга страна обаче, след "раздялата" при z, циклите трябва да се съберат отново в някакъв връх ω; това е очевидно предвид факта, че u е общ връх за двата цикъла. Обаче подпътят на c от z до ω образува цикъл с подпътя на c' от z до ω, като този цикъл (той не е нито c, нито c') се намира изцяло в T. Знаем, че T е ацикличен, така че такъв трети цикъл не може да има, така че и c' не може да съществува. □

Определение 46: Уницикличен граф

Свързан граф, който има точно един цикъл, се нарича *уницикличен граф*.

На английски терминът е *unicyclic graph*. Лема 10 казва, че добавянето на ребро към несъседи на дърво води до уницикличен граф.

За доказателството на Лема 11 ще използваме Определение 44.

Лема 11

Във всяко дърво, $m = n - 1$.

Доказателство: Със структурна индукция. В базовия случай на Определение 44, очевидно за графа с един връх и нула ребра, твърдението е вярно. ✓

Нека твърдението е вярно за дървото T от индуктивната стъпка. С други думи, допускаме, че

$$|E(T)| = |V(T)| - 1 \tag{2.11}$$

Трябва да докажем, че твърдението е вярно за дървото T'. С други думи, да докажем, че

$$|E(T')| = |V(T')| - 1 \tag{2.12}$$

Но това е съвършено очевидно предвид факта, че $|E(T')| = |E(T)| + 1$ и $|V(T')| = |V(T)| + 1$; ако заместим $|E(T')|$ с $|E(T)| + 1$ и $|V(T')|$ с $|V(T)| + 1$ в (2.12), ще получим равенство, еквивалентно на (2.11).

Лема 8 казва, че дърво с поне два върха има поне два висящи върха. Лема 12 дава точния брой на висящите върхове, изразен чрез броя на върховете от степен поне 3. Забележете, че върховете от степен 2 нямат значение за броя на висящите върхове; в сумата бихме могли да сумираме и по тях, понеже, ако $d(v) = 2$, очевидно $d(v) - 2 = 0$, така че техният "принос" би бил нула.

Лема 12

Нека $T = (V, E)$ е дърво с поне два върха. Броят p на висящите върхове в T е

$$p = 2 + \sum_{\substack{v \in V \\ d(v) \geqslant 3}} (d(v) - 2)$$

Доказателство: От Лема 1 знаем, че $\sum_{v \in V} d(u) = 2m$, а от Лема 11 знаем, че $m = n - 1$.

Тогава

$$\begin{aligned}
&\sum_{v\in V} d(v) = 2n - 2 \quad \leftrightarrow \\
&-2 = \left(\sum_{v\in V} d(v)\right) - 2n \quad \leftrightarrow \\
&-2 = \sum_{v\in V} (d(v) - 2) \quad \leftrightarrow \\
&-2 = \sum_{\substack{v\in V \\ d(v)=1}} (d(v) - 2) + \sum_{\substack{v\in V \\ d(v)=2}} (d(v) - 2) + \sum_{\substack{v\in V \\ d(v)\geqslant 3}} (d(v) - 2) \quad \leftrightarrow \\
&-2 = -p + 0 + \sum_{\substack{v\in V \\ d(v)\geqslant 3}} (d(v) - 2) \quad \leftrightarrow \\
&p = 2 + \sum_{\substack{v\in V \\ d(v)\geqslant 3}} (d(v) - 2) \qquad (2.13)
\end{aligned}$$

Това остава в сила дори $\Delta(T) = 2$. Тогава, от една страна, дървото е прост път и има точно два висящи върха, а, от друга страна, $\sum_{\substack{v\in V \\ d(v)\geqslant 3}} (d(v) - 2) = 0$, понеже $\{v \in V \mid d(v) \geqslant 3\} = \varnothing$, така че (2.13) остава в сила. □

2.11.3 Коренови дървета

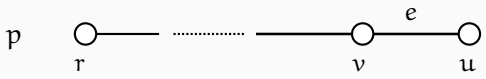
В Компютърните науки следната разновидност на понятието "дърво" се използва много повече повече от общото понятие "дърво" от Определение 43.

Кореново дърво, не-индуктивна дефиниция

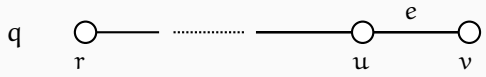
Определение 47: Кореново дърво, глобална дефиниция. Родител и дете.

Нека $T = (V, E)$ е дърво. Избираме произволен връх $r \in V$ и го наричаме *корен*. След избора на корен T става *кореново дърво*. Изборът на корен еднозначно определя релация на *родителство* върху всяко ребро. Нека $e \in E$ е произволно ребро и нека $e = (u, v)$. Съгласно Лема 9, съществува точно един път p между r и u и съществува точно един път q между r и v. Нещо повече.

- Или $V(p) = V(q) \cup \{u\}$, в който случай v е предпоследният връх преди u в p и казваме, че v *е родителят на* u, а u *е дете на* v:



- или $V(q) = V(p) \cup \{v\}$, в който случай u е предпоследният връх преди v в q и казваме, че u *е родителят на* v, а v *е дете на* u.



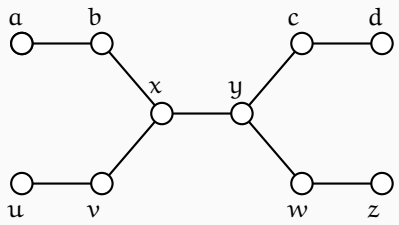
На английски терминът е *rooted tree*.

Конвенция 9: "Дърво" без "кореново" означава не-кореново дърво

Въпреки че кореновите дървета се ползват много повече в Компютърните науки от обикновените, не-коренови дървета, ако кажем само "дърво" без прилагателното "кореново", разбираме дърво, което не е кореново. Ако искаме да кажем дърво, което е кореново, трябва да кажем "кореново" експлицитно.

Ето пример за кореново дърво. Първо да разгледаме обикновеното (не-кореново) дърво T на Фигура 2.71.

Фигура 2.71 : Обикновено дърво T.

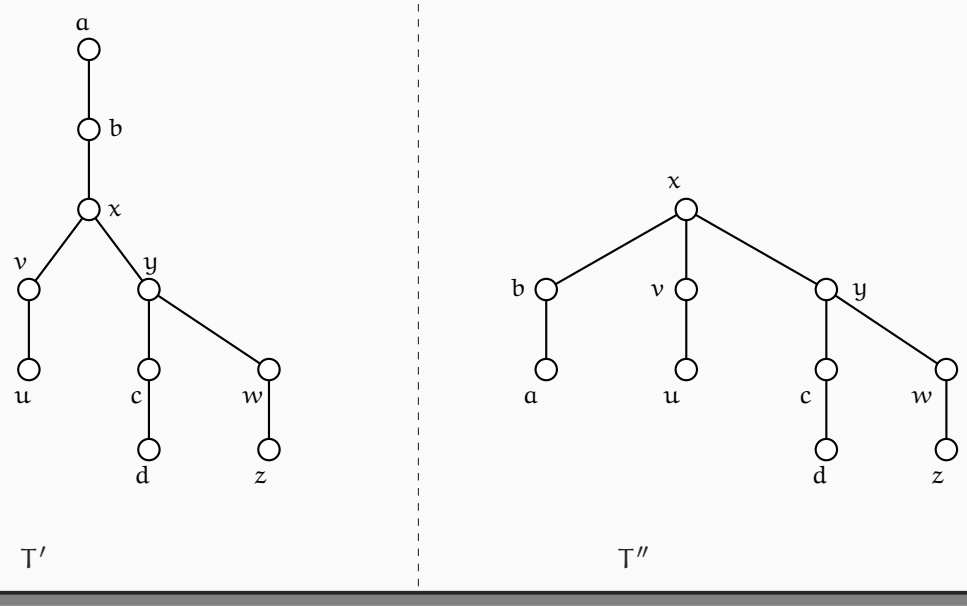


Фигура 2.72 илюстрира две различни коренови дървета, които се получават от не-кореновото дърво от Фигура 2.71 след различни избирания на корен:

- Дървото T' се получава от T след избор на $\mathtt{a}$ за корен.
- Дървото T'' се получава от T след избор на $\mathtt{x}$ за корен.

Прието е кореновите дървета да се рисуват така, както е показано на фигурата: върховете се рисуват по нивá съгласно разстоянието до корена, като коренът се рисува най-горе, а всяко следващо ниво е под предното.

Фигура 2.72 : Коренови дървета, получени от дървото на Фигура 2.71 при различни избори на корен.



Върховете в кореново дърво, които нямат деца, се наричат *листата на дървото*. В последния пример, листата на T' са $\mathfrak{u}$, $\mathfrak{d}$ и $\mathfrak{z}$, а листата на T'' са $\mathfrak{a}$, $\mathfrak{u}$, $\mathfrak{d}$ и $\mathfrak{z}$. Всяко листо в кореновото дърво е висящ връх в съответното не-кореново дърво, но обратното не винаги е вярно. Възможно е (точно един) висящ връх в не-кореновото дърво да не е листо в полученото кореново, и това е точно тогава, когато този връх е избран за корен. Примерно, в T' връх $\mathfrak{a}$ не е листо, въпреки че $\mathfrak{a}$ е висящ връх в T.

За удобство дефинираме, че ако D е не-кореново дърво с един единствен връх $\mathfrak{f}$ и направим D кореново дърво (при което нямаме друг избор, освен да направим $\mathfrak{f}$ корен), то $\mathfrak{f}$ е и листо. И така, коренът е листо в един единствен случай – дървото има точно един връх.

Няма единно мнение за това, кои върхове са вътрешните върхове на кореново дърво. Много автори, например Rosen [54, стр. 748], използват термина "вътрешен връх" (на английски е "internal vertex") за всеки връх в кореново дърво, който не е листо. Други автори като Epp [22] ползват "internal vertex" и в контекста на не-коленовите дървета, (стр. 688), и в контекста на кореновите дървета, като явно не смята корена за вътрешен връх (вижте определението на стр. 694 в [22]). Ние ще приемем определението на Rosen.

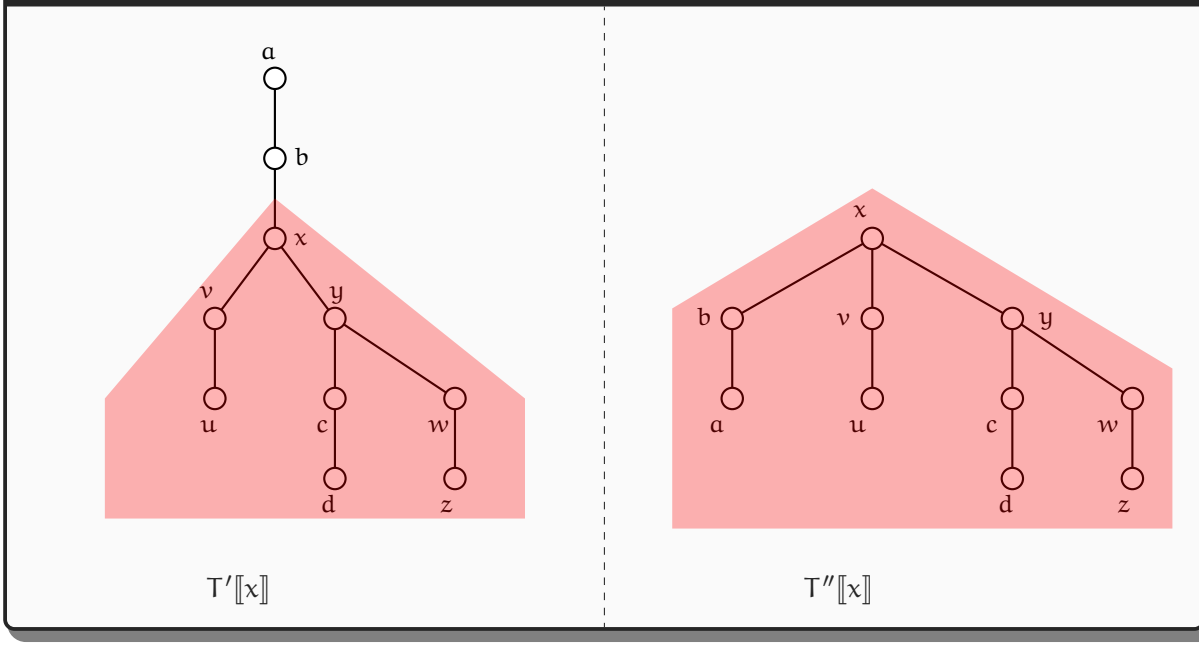
Определение 48: Вътрешни върхове в кореново дърво

Вътрешните върхове на кореново дърво са точно тези върхове, които не са листа.

В тези записки въведохме понятието "вътрешен връх" и при пътищата (Определение 16). Нека не бъркаме двете употреби на "вътрешен връх" – те са различни! Пътищата наистина са частен случай на дървета, но не-коренови. Ако път с дължина $\geqslant 1$ бъде направен кореново дърво чрез избор на един от краищата за корен, то този връх (коренът) се явява вътрешен връх съгласно Определение 48, ако гледаме на обекта като на кореново дърво, но не е вътрешен връх, ако гледаме на обекта като на не-кореново дърво.

В контекста на дадено кореново дърво, дефинираме *поддърво, вкоренено във връх*. Нефомално, то се състои от този връх и всичко, което се намира "под него" (и върхове, и ребра). Формално, ако T е кореново дърво с корен $\mathfrak{r}$ и $\mathfrak{u} \in \mathsf{V}(\mathsf{T})$, то поддървото на T, вкоренено в $\mathfrak{u}$, е подграфът на T, индуциран от множеството от всички върхове $\mathfrak{u}_1$, за които $\mathfrak{u}$ е връх от уникалния $\mathfrak{r}$-$\mathfrak{u}_1$ път. Поддървото на T, вкоренено в $\mathfrak{u}$, се смята за кореново дърво с корен $\mathfrak{u}$ и се бележи с "$\mathsf{T}[\![\mathfrak{u}]\!]$". Очевидно е, че $\mathsf{T}[\![\mathfrak{u}]\!] = \mathsf{T}$ тогава и само тогава, когато $\mathfrak{u} = \mathfrak{r}$. Като пример да разгледаме T' и T'' от Фигура 2.72. На Фигура 2.73 са показани $\mathsf{T}'[\![\mathfrak{x}]\!]$ и $\mathsf{T}''[\![\mathfrak{x}]\!]$, очертани полупрозрачен червен цвят съответно върху T' и T''. Ясно е, че $\mathsf{T}'[\![\mathfrak{x}]\!]$ и $\mathsf{T}''[\![\mathfrak{x}]\!]$ са различни обекти, въпреки че съответното не-кореново дърво е едно и също.

Фигура 2.73 : Поддърветата, вкоренени в x във всяко от кореновите дървета на Фигура 2.72.



В кореновите дървета се дефинират и релации на *предшествие* и *наследство* над върховете. За произволен връх, образно казано, предшествениците му са върховете "над него", а наследниците му са върховете "под него". Ето и формалните дефиниции.

Определение 49: Предшественици и наследници на връх.

Нека T е кореново дърво с корен r и u е произволен връх в него. За всеки връх v казваме, че v *е предшественик на* u, ако v е връх от u-r пътя. За всеки връх v казваме, че v *е наследник на* u, ако u е предшественик на v.

Така дефинираните релации са рефлексивни и транзитивни, защото:

- предшествениците на u са самият u, родителят на u, ако има такъв, неговият родител, ако има такъв, и така нататък чак до корена r.
- наследниците на u са самият u, децата на u, ако има такива, техните деца, ако има такива, и така нататък чак до листата.

Предшествието се явява рефлексивното и транзитивно затваряне на родителството, а другата релация (може ли да се каже "наследствието"?) се явява рефлексивното и транзитивно затваряне на релацията "___ е дете на ___". Лесно се вижда, че множеството от наследниците на u е $V(T⟦u⟧)$.

Примерно, в T' от Фигура 2.72, предшествениците на x са самият x, b и a, а наследниците на x са x, v, y, u, c, w, d и z.

Дали кореновите дървета са ориентирани графи[†], или не? Обикновените дървета категорично са неориентирани графи. При кореновите дървета говорим не за "съседство" на два върха, а за "родител" и "дете". Това е терминология, характерна за ориентираните графи. Някои автори, например Rosen [54, стр. 747], казват, че кореновите дървета са ориентирани графи: избирайки корен, избираме и ориентация (посока) на всяко ребро, като ориентацията на ребрата е от корена навън (към листата). Това е смислено съображение, но, от друга

[†]Вж. Глава 3.

страна, когато строим покриващо кореново дърво на неориентиран граф, то трябва да е неориентиран граф, бивайки подграф на неориентиран граф. Трудно е да се даде категоричен еднозначен отговор, приложим навсякъде, дали кореновите дървета са ориентирани или неориентирани.

Може обаче и да фиксираме ориентация на ребрата в явен вид и тогава говорим за ориентирани коренови дървета. Подсекция 3.1.9 разглежда подробно ориентираните дървета.

Определение 50: Височина на връх и на кореново дърво. Дълбочина на връх.

Нека T е кореново дърво с корен $\mathtt{r}$. Нека $\mathtt{u}$ е произволен връх в T. *Височината на* $\mathtt{u}$ е максималното разстояние между $\mathtt{u}$ и кое да е листо в $\mathsf{T}[\![\mathtt{u}]\!]$. *Височината на* T е височината на $\mathtt{r}$.

Дълбочината на $\mathtt{u}$ е разстоянието между $\mathtt{u}$ и $\mathtt{r}$.

На английски съответните термини са *height* и *depth*. Очевидно височината на T може да се дефинира и като максималната дълбочина на връх в T. Забележете, че терминът "ниво в кореново дърво", който използвахме на стр. 111, означава "множеството от върховете, които имат една и също дълбочина".

Да разгледаме кореновите дървета T' и T'' на Фигура 2.72. В T': височината на $\mathtt{a}$ е 5, височината на $\mathtt{b}$ е 4, височината на $\mathtt{x}$ е 3, и така нататък, височината на $\mathtt{d}$ е 0 и височината на $\mathtt{z}$ е 0, а височината на самото T' е 5. В T'': височината на $\mathtt{a}$ е 0, височината на $\mathtt{b}$ е 1, височината на $\mathtt{x}$ е 3, което е и височината на самото T'', и така нататък. В T': дълбочината на $\mathtt{a}$ е 0, на $\mathtt{x}$ е 2, на $\mathtt{d}$ е 5, и така нататък. В T'': дълбочината на $\mathtt{a}$ е 2, на $\mathtt{x}$ е 0, на $\mathtt{d}$ е 3, и така нататък.

Определение 51: Разклоненост на кореново дърво.

Нека T е кореново дърво. *Разклонеността на* T е максималният брой деца на кой да е връх в дървото.

На английски терминът е *branching factor*.

Да разгледаме кореновите дървета T' и T'' на Фигура 2.72. Разклонеността на T' е 2, а разклонеността на T'' е 3. Очевидно разклонеността е свързана с $\Delta(\mathsf{T})$, където T е съответното не-кореново дърво; а именно, разклонеността на кореновото дърво е равна на $\Delta(\mathsf{T})$, ако коренът на кореновото дърво е връх от максимална степен в не-кореновото (това е случаят с T''), или е равна $\Delta(\mathsf{T}) - 1$, в противен случай (това е случаят с T').

На английски често се казва "k-ary tree" като синоним на "кореново дърво с разклоненост k". На български ще казваме "k-ично дърво". От особено значение е случаят, в който $\mathtt{k} = 2$.

Определение 52: Двоично дърво.

Двоично дърво е кореново дърво с разклоненост 2.

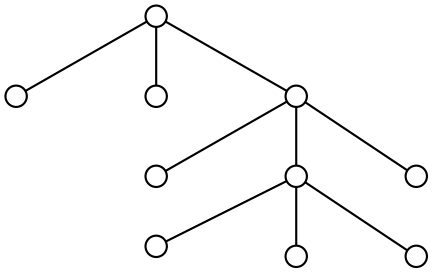
На английски терминът е *binary tree*.

Определение 53: Пълно дърво.

Пълно k*-ично дърво* е кореново дърво, в което всеки вътрешен връх има точно k деца. *Пълно дърво* е кореново дърво, което е пълно k-ично дърво за някое k.

На английски термините са съответно *full* k*-ary tree* и *full tree*.

Ето пример за пълно троично дърво.



Наблюдение 23: Какви са степените на върховете в съответното не-кореново дърво

Нека T е пълно кореново k-ично дърво с повече от един връх. Нека T' е съответното му не-кореново дърво (в очевидния смисъл). Тогава за степените на върховете на T' е вярно следното.

- Точно един връх има степен k. Това е върхът, който е коренът в T.
- Нула или повече върхове имат степен $\mathsf{k}+1$. Това са точно върховете, които са вътрешни, но различни от корена, в T.
- Поне k върха са висящи. Това са точно листата в T.
- Върхове от други степени няма.

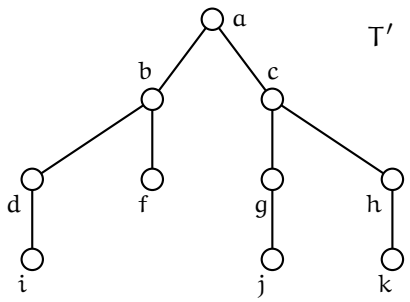
Забележете, че това остава в сила дори когато $\mathsf{k}=1$: пълното 1-ично кореново дърво е прост път, вкоренен в единия си край.

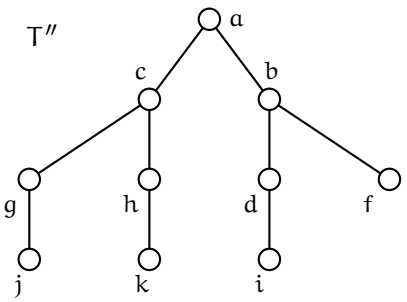
Определение 54: Подредено дърво.

Подредено дърво е кореново дърво, в което децата на всеки вътрешен връх са подредени отляво надясно.

На английски терминът е *ordered tree*.

Следната фигура показва две коренови дървета T' и T''. Ако гледаме на тях като на подредени дървета, те са **различни** подредени дървета, защото в T' децата на $\mathtt{a}$ са подредени по различен начин, отляво надясно, в сравнение с T''. Но ако гледаме на T' и T'' на обикновени коренови дървета, а не подредени дървета, те са **едно и също** дърво, защото имат едно и също множество от върхове и децата на всеки връх в T' са точно същите върхове, които са неговите деца и в T''.





Кореново дърво, индуктивна дефиниция Следните определения на "кореново дърво" са еквивалентни на Определение 47. Доказателството на това остава на читателя.

Определение 55: Кореново дърво, първа индуктивна дефиниция.

❶ **База** Всеки тривиален граф $(\{u\}, \varnothing)$ е кореново дърво с корен u, множество от листа $\{u\}$, разклоненост 0 и височина 0.

❷ **Индуктивна стъпка** Нека $T_1 = (V_1, E_1)$, ..., $T_k = (V_k, E_k)$ са коренови дървета, които две по две нямат общи върхове, с корени съответно $r_1, \ldots, r_k$, множества от листа съответно $W_1, \ldots, W_k$, разклонености съответно $b_1, \ldots, b_k$ и височини съответно $h_1, \ldots, h_k$. Нека r е връх, който не се намира в никое от тях. Нека $E' = \{(r, r_i) \mid 1 \leqslant i \leqslant k\}$. Тогава

$$T = \left(\{r\} \cup \bigcup_{i=1}^{k} V_i, E' \cup \bigcup_{i=1}^{k} E_i\right)$$

е кореново дърво с корен r, множество от листа $\bigcup_{i=1}^{k} W_i$, разклоненост $\max\{k, b_1, \ldots, b_k\}$ и височина $\max\{h_1, \ldots, h_k\} + 1$.

Определение 56: Кореново дърво, втора индуктивна дефиниция.

❶ **База** Всеки тривиален граф $(\{u\}, \varnothing)$ е кореново дърво с корен u, множество от листа $\{u\}$, разклоненост 0 и височина 0.

❷ **Индуктивна стъпка** Нека $T_1 = (V_1, E_1)$, и $T_2 = (V_2, E_2)$ са дървета, които нямат общи върхове, с корени съответно r_1 и r_2, множества от листа съответно W_1 и W_2, разклонености съответно b_1 и b_2 и височини съответно h_1 и h_2. Нека $v \in V_1$, нека v има f_v деца и нека дълбочината на v е t_v. Тогава

$$T = (V_1 \cup V_2, E_1 \cup E_2 \cup \{(v, r_2)\})$$

е кореново дърво с корен r_1. Множеството от листата на T е $(W_1 \backslash \{v\}) \cup W_2$, разклоненността му е $\max\{b_1, b_2, f_v + 1\}$, а височината му е $\max\{h_1, t_v + 1 + h_2\}$.

Определение 57: Кореново дърво, трета индуктивна дефиниция.

❶ **База** Всеки тривиален граф $(\{u\}, \varnothing)$ е кореново дърво с корен u, множество от листа $\{u\}$, разклоненост 0 и височина 0.

❷ **Индуктивна стъпка** Нека $T = (V_T, E_T)$, е коренови дървета с корен съответно r, множества от листа W, разклоненост b и височини съответно h. Нека z_1, ..., z_q са върхове, които не са във V_T, и нека $\ell \in W$. Тогава

$$D = (V_T \cup \{z_1, \ldots, z_q\}, E_T \cup \{(\ell, z_1), \ldots, (\ell, z_q)\})$$

е кореново дърво с корен r и множество от листа $(W \backslash \{\ell\}) \cup \{z_1, \ldots, z_q\}$. Разклоненността на D е $\max\{b, q\}$. Височината на D е или h, ако дълбочината на ℓ в T е по-малка от h, или $h + 1$, ако ℓ в T е h.

Изоморфизмът между коренови дървета има една особеност: разговорно казано, той трябва да "съблюдава корените", освен върховете и ребрата. Поради това Определение 58 се получава от Определение 39 с допълнително изискване.

Определение 58: Изоморфизъм между коренови дървета.

Нека $T' = (V', E')$ и $T'' = (V'', E'')$ са коренови дървета с корени съответно r' и r''. *Изоморфизъм между кореновите дървета* T' *и* T'' *с корени съответно* r' *и* r'' е всяка биекция $\phi : V' \to V''$, такава че

$$\forall u, v \in V' : (u, v) \in E' \leftrightarrow (\phi(u), \phi(v)) \in E''$$
$$\phi(r') = r''$$

И така, изоморфизъм между коренови дървета е всеки изоморфизъм между съответните не-коренови дървета, който освен това изобразява корен в корен. Като пример, кореновите дървета на Фигура 2.72 **не са изоморфни**, въпреки че съответните не-коренови дървета са изоморфни (нещо повече, те са едно и също не-кореново дърво): забележете, че нито една биекция от $V(T')$ във $V(T'')$, която изобразява a (коренът на T') в x (коренът на T''), не е изоморфизъм.

Свойства на кореновите дървета

Теорема 23: Брой на върховете на пълно кореново k-ично дърво

Във всяко пълно кореново k-ично дърво T с повече от един връх е вярно, че $n = kp + 1$, където p е броят на вътрешните върхове.

Доказателство: Първо ще разгледаме случая, в който $k \geqslant 2$. Да разгледаме не-кореновото дърво T', съответно на T. Очевидно броят на висящите върхове на T' е равен на броя на листата на T, тоест, $n - p$. Съгласно Лема 12:

$$\begin{aligned} n - p &= 2 + \sum_{\substack{v \in V \\ d(v) \geqslant 3}} (d(v) - 2) \quad // \text{ съгласно Наблюдение 23} \\ &= 2 + \sum_{\substack{v \in V \\ d(v) = k}} (d(v) - 2) + \sum_{\substack{v \in V \\ d(v) = k+1}} (d(v) - 2) \end{aligned}$$

Но, съгласно Наблюдение 23, T' има точно един връх от степен k и точно $p - 1$ върха от степен $k + 1$. Тогава

$$n - p = 2 + 1 \cdot (k - 2) + (p - 1)(k - 1) = 2 + k - 2 + kp - k - p + 1$$

Тогава $n = kp + 1$.

Сега да разгледаме случая $k = 1$. Тогава твърдението, което доказваме, става $n = p + 1$. Но при $k = 1$ кореновото дърво е прост път, вкоренен в един от краищата си, и T има точно едно листо и точно $n - 1$ вътрешни върхове, така че $p = n - 1$. Тогава твърдението става $n = n - 1 + 1$, което очевидно е вярно.

Твърдението е в сила и за $n = 1$, независимо от това дали $k > 1$ или $k = 1$. Ако $n = 1$, то $p = 0$ и $n = kp + 1$ следва веднага. □

Ето едно лесно следствие от Теорема 23. Нека е дадено пълно k-ично кореново дърво и разклонеността $k \geqslant 2$ е фиксирана. Могат да варират параметрите **брой върхове** n, **брой листа** и **брой нелиста**. Тогава, ако фиксираме и кой да е от тези три параметъра, това определя напълно и останалите два.

Следствие 5

Нека T е пълно k-ично кореново дърво, като $k \geqslant 2$. Тогава,

❶ Изразено в n:

- броят на вътрешните върхове е $\dfrac{n-1}{k}$,
- а броят на листата е $\dfrac{(k-1)n+1}{k}$.

❷ Изразено в броя на вътрешните върхове p:

- $n = kp + 1$,
- а броят на листата е $(k-1)p + 1$.

❸ Изразено в броя на листата ℓ:

- $n = \dfrac{k\ell - 1}{k-1}$,
- а броят на вътрешните върхове е $\dfrac{\ell - 1}{k-1}$.

Доказателство: Доказателствата са елементарни. Винаги тръгваме оттам, че $n = kp + 1$, където p е броят на вътрешните върхове.

Първо, $n - 1 = kp$, така че $p = \frac{n-1}{k}$, а оттук броят на листата е $n - p = n - \frac{n-1}{k} = \frac{(k-1)n+1}{k}$, с което доказахме ❶.

Съждение ❷ съдържа буквално $n = kp + 1$, откъдето пък броят на листата е $n - p = (k-1)p + 1$.

Нека ℓ означава броя на листата. Очевидно, $\ell = n - p$. Тогава $n = k(n-\ell)+1 = kn - k\ell + 1$, откъдето $k\ell = (k-1)n + 1$, откъдето $n = \frac{k\ell-1}{k-1}$. Тъй като $n = kp + 1$ и $n = \ell + p$, вярно е, че $\ell + p = kp + 1$, откъдето $\ell - 1 = (k-1)p$, откъдето $p = \frac{\ell-1}{k-1}$, с което доказахме ❸.

Съжденията ❶ и ❷ не съдържат $k-1$ в знаменател на израз и остават верни и при $k = 1$. Нека $k = 1$, тоест, T е прост път, вкоренен в единия си край. Съждение ❶ казва, че вътрешните върхове са $n - 1$ на брой, а листата са 1 на брой. Това е вярно дори при $n = 1$.

Съждение ❷ казва, че $n = p + 1$, а листата са 1 на брой, което очевидно е вярно, дори при $n - 1$. □

Следствие 6 се получава лесно от Следствие 5 след заместване на k с 2.

Следствие 6

Ако T е пълно двоично дърво, броят на върховете n е нечетно число, като броят на вътрешните върхове е $\dfrac{n-1}{2}$, а броят на листата е $\dfrac{n+1}{2}$. Тогава листата са с точно едно повече от вътрешните върхове.

Теорема 24: Броят на листата е $\leqslant k^h$ при k-ично дърво с височина h

Нека T е k-ично дърво с височина h. Нека ℓ е броят на листата. Тогава $\ell \leqslant k^h$. Алтернативно, $\log_k \ell \leqslant h$.

Доказателство: Що докажем твърдението с индукция по h. Базата е за $h = 0$. Но $h = 0$ тстк $n = 1$, което означава да има един единствен връх, който е и корен, и (единственото) листо. Следователно, $\ell = 1$ и неравенството става $1 \leqslant k^0$. ✓

Да допуснем, че броят на листата е не по-голям от k^h за всяко дърво с височина h. Но очевидно всяко дърво D с височина $h + 1$ се получава от някое дърво T с височина h, като към поне едно листо на T с дълбочина h се добавят нови листа (като в индуктивната стъпка на Определение 57). Нещо повече. Броят на листата на D е максимален, когато на всяко листо на T "му поникват" k листа. Тъй като листата на T са не повече от k^h, то листата на D са не повече от $k \cdot k^h = k^{h+1}$. □

Забележете, че Теорема 24 е вярна дори в дегенеративния случай $k = 1$. Тогава дървото има точно едно листо независимо от височината. От друга страна, очевидно $1 \leqslant 1^h$.

Следствие 7: Височината на k-ично дърво е поне логаритмична в n

Нека T е k-ично дърво с височина h и $k \geqslant 2$. Тогава $h \geqslant (\log_k n) - 1$, като тази долна граница е точна.

Доказателство: Искаме да покажем, че

$$h \geqslant (\log_k n) - 1$$

Но това е същото като да покажем, че

$$h \geqslant \log_k \frac{n}{k}$$

От Следствие 5 знаем, че $\ell = \frac{(k-1)n+1}{k}$. От Теорема 24 знаем, че $h \geqslant \log_k \ell$. Тогава

$$h \geqslant \log_k \frac{(k-1)n+1}{k}$$

Дали е вярно, че

$$\frac{(k-1)n+1}{k} \geqslant \frac{n}{k} \, ?$$

Да, със сигурност, понеже

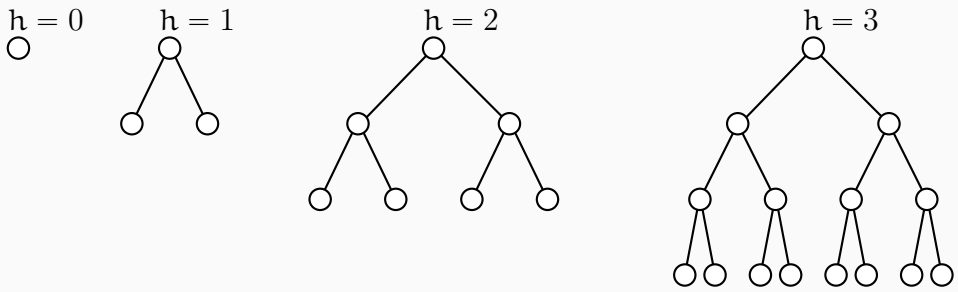
$$\frac{(k-1)n+1}{k} \geqslant \frac{n}{k} \leftrightarrow (k-1)n + 1 \geqslant n$$

при $k \geqslant 2$. Но $\frac{(k-1)n+1}{k} \geqslant \frac{n}{k}$ влече $\log_k \frac{(k-1)n+1}{k} \geqslant \log_k \frac{n}{k}$. От това, че $h \geqslant \log_k \frac{(k-1)n+1}{k}$, следва, че $h \geqslant \log_k \frac{n}{k}$.

А това, че долната граница $(\log_k n) - 1$ за височината е точна, се доказва тривиално: $\log_k n$ не е непременно долна граница за височината, примерно при $k = 2$ и $n = 14$, височината е 3, докато $\log_2 14 \approx 3.807354922$. □

Интересен е частният случай, в който в пълно k-ично дърво всички листа имат една и съща дълбочина. Иначе казано, във всяко ниво t има точно k^t върха, за $0 \leqslant t \leqslant h$, където h е височината на дървото. Такова дърво се нарича *съвършено k-ично дърво*, на английски *perfect k-ary tree*. За всяко h има точно едно съвършено k-ично дърво, ако дървото е с анонимни върхове. Фигура 2.74 показва съвършените двоични дървета с височини 0, 1, 2 и 3.

Фигура 2.74 : Съвършени двоични дървета.



Нека читателят формулира еквивалентна индуктивна дефиниция на множеството от съвършените k-ични дървета и докаже еквивалентността на двете дефиниции. Тук само ще отбележим, че съвършените k-ични дървета се явяват максималните пълни k-ични дървета, при фиксирана височина, и за тях нестрогото неравенство в Теорема 24 става равенство: $\ell = k^h$.

В частния случай $k = 2$ става дума за съвършено двоично дърво. Броят на листата на съвършено двоично дърво е точно 2^h, а $n = 2^{h+1} - 1$, като вътрешните върхове са точно $2^h - 1$, тоест, с едно по-малко от броя на листата. От това, че $n = 2^{h+1} - 1$, веднага имаме $h = (\log_2(n+1)) - 1$ за съвършените двоични дървета. Лесно се вижда, че това е същото като $h = \lceil \log_2 n \rceil - 1$. Забележете, че това се явява частен случай на твърдението от Следствие 7.

2.11.4 Покриващи дървета

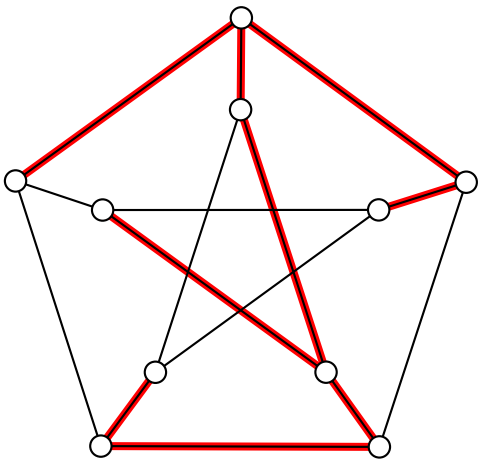
"Покриващо дърво" е частен случай на "покриващ граф" (Определение 10), но тъй като ще разглеждаме най-често покриващи графи, които са дървета, заслужава си да направим отделно определение.

Определение 59: Покриващо дърво.

Нека $G = (V, E)$ е свързан граф. *Покриващо дърво* на G е всяко дърво $T = (V, E')$, където $E' \subseteq E$.

На английски терминът е *spanning tree*.

Ето пример за покриващо дърво на графа на Petersen. Върховете са анонимни.



Разглеждаме покриващи дървета само в обикновени графи, въпреки че понятието лесно може да се обобщи и за покриващи дървета на мултиграфи; ако разглеждаме мултиграф с възможни примки, то примките никога не участват в покриващо дърво (понеже всяка примка е цикъл), а от всеки сноп паралелни ребра може да участва най-много едно ребро, в противен случай покриващия граф би съдържал цикъл.

Теорема 25: Свързаност и наличие на покриващо дърво

За всеки граф $G = (V, E)$, G има поне едно покриващо дърво тогава и само тогава, когато G е свързан.

Доказателство: Ако G има покриващо дърво, то очевидно G е свързан, понеже между всеки два върха в покриващото дърво има път, което влече съществуване на път между тези два върха и в G.

Ще докажем твърдението в другата посока. Ако G е свързан, то следният алгоритъм строи покриващо дърво на G.

Алгоритъм 3: Построяване на покриващо дърво

Вход: свързан граф $G = (\{v_1, v_2, \ldots, v_n\}, E)$.
Изход: Покриващо дърво на G.

❶ Ако G няма цикли, върни G и край.

❷ В противен случай, нека c е произволен цикъл в G и нека e е произволно ребро от c.

❸ Направи $G \leftarrow G - e$ и после отиди на ❶.

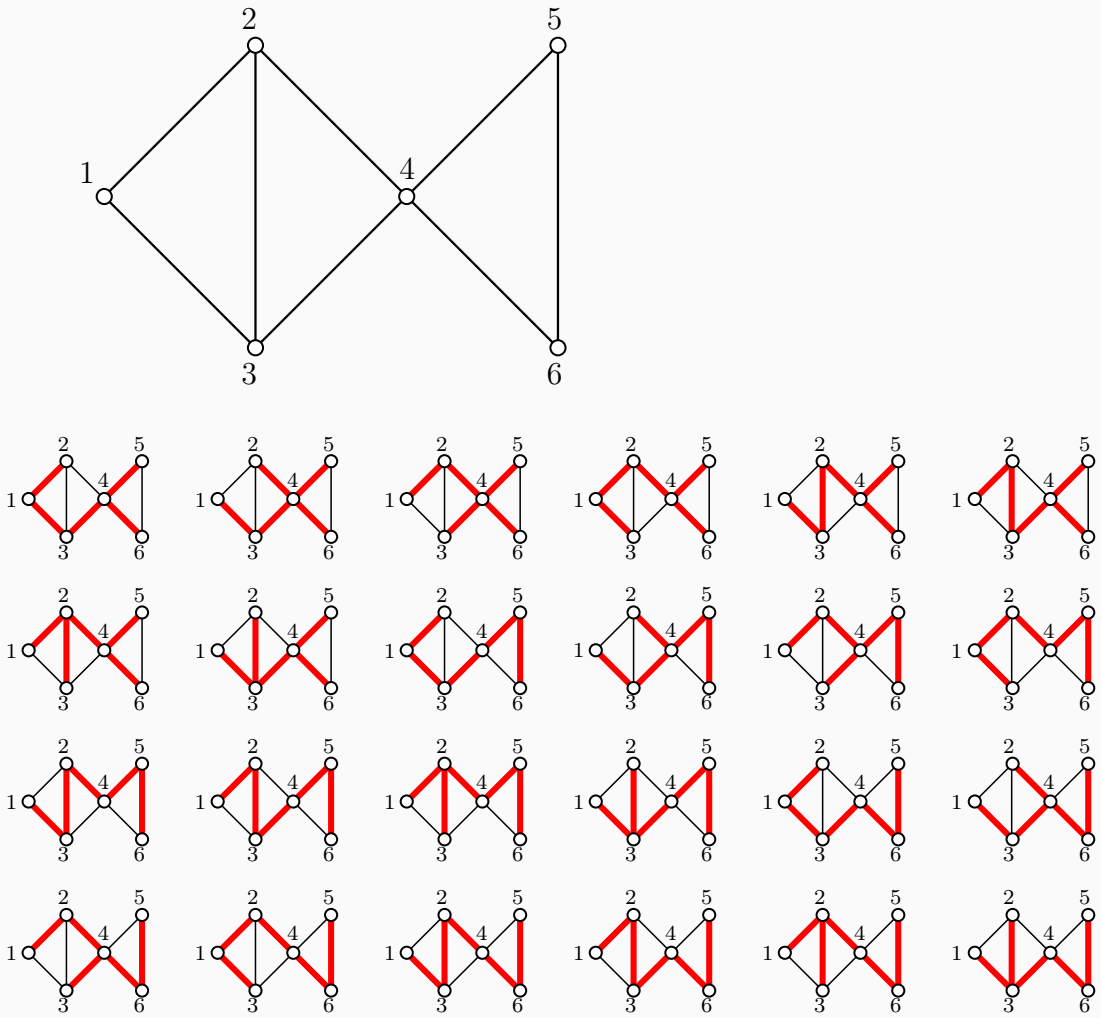
Доказателството за коректността на алгоритъма е много лесно. Поначало G е свързан граф. Съгласно Теорема 4, изтриването на ребро от цикъл на свързан граф оставя графа свързан, така че изтриването на e на ред ❸ не нарушава свързаността на графа. Тривиално е да се покаже по индукция, че при всяко достигане на ред ❶ графът е свързан.

Очевидно е, че множеството от върховете на графа не се променя при изтриване на ребра. След това забелязваме, че броят на циклите в първоначалния граф G е краен, а след всяко минаване през тялото на цикъла на алгоритъма [†] броят на циклите намалява поне с единица, така че, рано или късно, при достигането на ред ❶ графът ще е ацикличен и алгоритъмът ще го върне и терминира. Следователно, алгоритъмът връща граф, който е свързан, ацикличен и чиито върхове са същите като на първоначалния граф, така че резултатът е покриващо дърво на първоначалния граф. □

Какъв е максималният брой на покриващите дървета? Нека G е свързан граф. От Теорема 25 знаем, че G има поне едно покриващо дърво. Но колко точно са покриващите дървета на G? Нека $\kappa(G)$ е броят на именуваните покриващи дървета на G. Забележете, че става дума за именувани покриващи дървета, а не за покриващи дървета с анонимни върхове (припомнете си Подсекция 2.8.2). Като пример да разгледаме графа на Фигура 2.75. Той има точно 24 покриващи дървета, показани под него.

[†] На български има опасност от двусмислие в това изречение заради употребата на "цикъл" в два напълно различни смисъла – цикъл в графа и цикъл в алгоритъма, работещ върху графа. На английски термините са съответно "cycle" и "loop", така че двусмислие не може да има.

Фигура 2.75 : Граф и неговите покриващи дървета.



Очевидно $\kappa(G) = 1$ тстк G е дърво; в този случай, графът съвпада с единственото си покриващо дърво, понеже самият той е дърво. Ако започнем да добавяме ребра към дървото G, по този начин създавайки цикли, броят на покриващите дърветa расте стремително. Примерно, графът на Фигура 2.75 има 4 цикъла и 24 покриващи дървета. Лесно се вижда, че при фиксирано множество от върхове на G и възможност да се добавят ребра, $\kappa(G)$ е максимално тстк G е пълен граф, защото, колкото повече ребра има, толкова повече възможности има за различни покриващи дървета.

Пълният граф на n върха има точно n^{n-2} покриващи дървета. Този резултат е известен като *формула на Cayley*. И така, $1 \leqslant \kappa(G) \leqslant n^{n-2}$, като тези граници са точни.

Има доста начини за извеждане на формулата на Cayley. Сякаш най-лесният е чрез Теорема 26, която има алгоритмично доказателство. В Допълнение 15 има две други извеждания на формулата на Cayley (Теорема 28 и Теорема 31).

Теорема 26: Формула на Cayley, доказателство с кодове на Prüfer

Броят на именуваните дървета на n върха е n^{n-2}, за всяко $n \geqslant 2$.

Доказателство: Следното доказателство дължим на математи́ка Heinz Prüfer [50], който открива биекция между дърветата на n върха и редиците с дължина $n-2$ над $\{1,\ldots,n\}$. Това е интересно приложение на комбинаторния принцип на биекцията, понеже дърветата се броят трудно, ако работим от общи съображения, а редиците се броят изключително лесно. Доказателството е алгоритмично: конструира се алгоритъм, който по дадено дърво генерира редица. Поотделно се доказва, че функцията, която реализира алгоритъмът, е инекция (различни дървета пораждат различни редици) и сюрекция (всяка редица е образ на някое дърво). От това следва съществуването на желаната биекция.

Въпросните редици са известни като *кодове на Prüfer*, на английски *Prüfer codes*, понеже всеки от тях кодира биективно някое дърво.

Ще игнорираме случая $n = 1$, въпреки че, формално погледнато, от една страна $1^{1-2} = 1^{-1} = 1$, а от друга страна има точно едно дърво с един връх. Нека $n \geqslant 2$.

Първа част на доказателството: генериране на код от дърво Ако $\mathbf{a}$ е редица от числа и b е число, то "$\mathbf{a}, b$" означава редицата $\mathbf{a}$ с още един елемент, а именно b, "залепен" накрая. "$\langle\,\rangle$" означава празната редица.

Алгоритъм 4: Код на Prüfer от дърво

Вход: дърво $T = (\{1,2,\ldots,n\}, E)$, като $n \geqslant 2$.
Изход: Кодът на Prüfer на T.

❶ $\mathbf{w} \leftarrow \langle\,\rangle$.

❷ Ако $|E| = 1$, край. Върни $\mathbf{w}$.

❸ В противен случай, нека u е висящият връх на T с най-малка стойност.
① Нека v е съседът на u в T.
② $\mathbf{w} \leftarrow \mathbf{w}, v$.
③ Изтрий u от T и иди на ❷.

Забележете, че върхът, който се "записва" в кода, не е изтритият u, а неговият единствен **съсед** v.

Ще докажем коректността на алгоритъма. Първо да се убедим, че операциите му са добре дефинирани.

- Ако изпълнението е на ред ❸, T има повече от два върха и, съгласно Лема 8, T има поне два висящи върха; ерго, u е добре дефиниран. Щом u е висящ, той има точно един съсед, така че и v на ред ① е добре дефиниран.
- Резултатът от изтриването на висящия връх u от дървото T (ред ③) е дърво, така че при следващото достигане на ред ❷, T пак е дърво.

Да отбележим два факта. Нека T означава първоначалното (входното) дърво.

Факт1 Всеки връх x на T се появява $d(x) - 1$ пъти в кода-изход. В частност, висящите върхове не се появяват изобщо (нула пъти).

Факт2 След първото изпълнение на ❸, връх u се оказва изтрит, а в началото на кода се появява идентификаторът на неговия съсед v и той остава там до края. Останалата част от кода-изход е кодът на $T - u$.

Факт1 е очевиден. Ако **Факт2** не е очевиден, да съобразим, че върховете на $T - u$ не са $\{1, \ldots, n\}$, а подмножество на $\{1, \ldots, n\}$, което в общия случай съдържа "дупка" (дупката е изтритият u). Това, че множеството от върховете на $T-u$ не се състои само от последователни естествени числа, няма никакво значение. За конструирането на кода няма значение дали върховете са последователни естествени числа или не; важното е да са две по две различни и тогава минималният висящ връх, който е следващият връх за изтриване, е добре дефиниран.

Ще докажем по индукция по броя на върховете n във входното дърво, че алгоритъмът връща различни кодове за различни дървета. Обаче алгоритъмът, който описахме, не е рекурсивен, а е итеративен, а индукция по големината на входа (броя на върховете) не е подходяща техника за итеративни алгоритми. Индукцията е подходяща техника за доказване на коректността на **рекурсивни** алгоритми. За да ползваме индукция по големината на входа, модифицираме алгоритъма в рекурсивен така:

Алгоритъм 5: Код на Prüfer от дърво, рекурсивен

Вход: дърво $T = (V, E)$, $|E| \geqslant 1$, $V \subset \mathbb{N}^+$.
Изход: Кодът на Prüfer на T.

❶ Ако $|E| = 1$, върни $\langle\,\rangle$ и край.

❷ В противен случай, нека u е висящият връх на T с най-малка стойност.
① Нека v е съседът на u в T.
② Извикай алгоритъма върху $T - u$. Нека върнатият код е $\mathbf{w}$.
③ Върни $v, \mathbf{w}$ и край.

Лема 13: Инективността на генерирането на код на Prüfer

За всеки две различни дървета T_1, T_2 върху едно и също множество от върхове V, алгоритъмът Код на Prüfer от дърво, рекурсивен връща различни кодове.

Доказателството на Лема 13 е по индукция по n. Базата е $n = 2$. На пръв поглед, доказателството се чупи заради това, че при $n = 2$, алгоритъмът връща задължително $\langle\,\rangle$ и нищо друго. На втори поглед обаче, проблем с доказателството няма! Твърдението е, че за всеки две **различни** дървета над едно и също множество върхове, върнатите кодове са различни. Но, ако множеството от върхове е двуелементно, то дървото над него е едно единствено, ерго не съществуват две различни дървета над него, така че твърдението е за елемент на празното множество (двойка различни дървета е един обект). А ние знаем, че $\forall x : P(x)$ е истина винаги, когато променливата x взема стойности от празното множество, независимо какъв е предикатът. Така че базата "излиза". ✓

Да допуснем, че твърдението е вярно за някакво $n \geqslant 2$. Разглеждаме две различни дървета T_1, T_2 над едно и също множество върхове V, като $|V| = n + 1$.

- Да допуснем, че минималният висящ връх в T_1 е x, в T_2 е y, и $x \neq y$. БОО, нека $x < y$. Тогава x не е висящ връх в T_2 и, съгласно **Факт1**, x се присъства в кода на T_2, но няма да присъства в кода на T_1. Ерго, кодовете на T_1 и T_2 се различават.

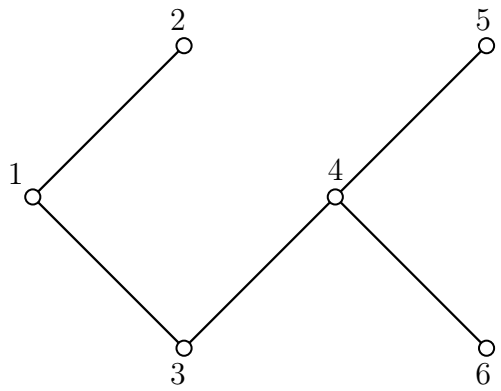
- Да допуснем, че T_1 и T_2 имат един и същи минимален висящ връх x, но съседите на x са различни в T_1 и T_2. Но кодовете на T_1 и T_2 започват със съседа на x. Тогава те се различават пак.

- Да допуснем, че T_1 и T_2 имат един и същи минимален висящ връх x и съседът на x в T_1 и T_2 е един и същи. В този случай кодовете на T_1 и T_2 започват с един и същи връх. Но $T_1 - x$ и $T_2 - x$ трябва да са различни дървета, инак T_1 и T_2 биха били едно и също дърво. Щом $T_1 - x$ и $T_2 - x$ са различни и имат по n върха всеки, то кодовете им са различни съгласно индуктивното предположение. Ерго, изходите $v, \mathbf{w}$ на ред ③ за T_1 и T_2 се различават. □

Следствие 8

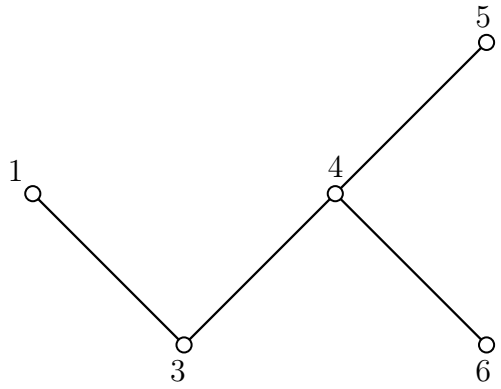
Алгоритъмът Код на Prüfer от дърво реализира инекция.

Ето пример за работата на алгоритъма. Дървото е първото от покриващите дървета от Фигура 2.75. В самото начало, $\mathbf{w}$ е празен.



$w = \langle\ \rangle$

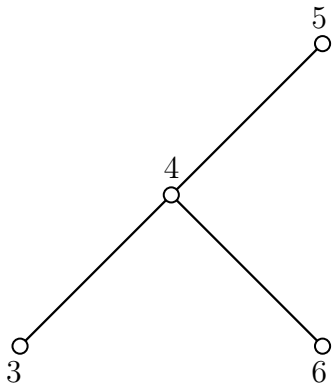
Висящите върхове са 2, 5 и 6. Минималният висящ връх[†] е 2. Неговият съсед е 1. Записваме 1 в $\mathbf{w}$ и изтриваме 2 от дървото. Дървото и кодът стават такива.



$w = \langle 1 \rangle$

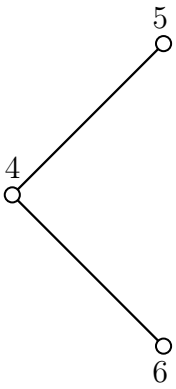
Висящите върхове са 1, 5 и 6. Минималният висящ връх е 1. Неговият съсед е 3. Записваме 3 в $\mathbf{w}$ и изтриваме 1 от дървото. Дървото и кодът стават такива.

[†]Върховете са числа, така че има смисъл да говорим за минимален връх.



$w = \langle 1, 3 \rangle$

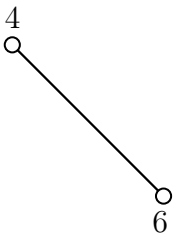
Висящите върхове са 3, 5 и 6. Минималният висящ връх е 3. Неговият съсед е 4. Записваме 4 в **w** и изтриваме 3 от дървото. Дървото и кодът стават такива.



$w = \langle 1, 3, 4 \rangle$

Висящите върхове са 5 и 6. Минималният висящ връх е 5. Неговият съсед е 4. Записваме 4 в **w** и изтриваме 5 от дървото. Дървото и кодът стават такива.

$w = \langle 1, 3, 4, 4 \rangle$



Изпълнението отново е на ред ❷, но сега вече реброто е само едно и алгоритъмът терминира, връщайки $\langle 1, 3, 4, 4 \rangle$. И така, $\langle 1, 3, 4, 4 \rangle$ е кодът на дървото от входа. Във втората част на доказателството ще видим алгоритъм, който от код строи съответното дърво, и в частност от този код построява именно дървото, с което започнахме.

Втора част на доказателството: генериране на дърво от код

Лема 14: Сюрективността на генерирането на код на Prüfer

За всяко $n \geqslant 2$, за всеки код C, който е редица с дължина $n-2$ над $\{1,\ldots,n\}$, съществува дърво T с върхове $\{1,\ldots,n\}$, такова че Код на Prüfer от дърво върху T връща C.

Доказателството на Лема 14 е по индукция по n. Базата е $n=2$. Наистина, има един единствен код, а именно празната редица $\langle\,\rangle$, с дължина $2-2=0$. От друга страна, алгоритъмът при вход дърво с върхове $\{1,2\}$ връща именно $\langle\,\rangle$, така че дърво с желаното свойство има. ✓

Да допуснем, че твърдението е вярно за някакво $n \geqslant 2$. Разглеждаме произволен код C с дължина $n-1$ над $\{1,\ldots,n+1\}$. Съгласно принципа на Dirichlet, поне едно число от $\{1,\ldots,n+1\}$ не се среща в C. Нека k е минималното число, което не се среща в C. Нека C' е кодът, който се образува от C чрез изтриване на първия елемент. Тогава C' е редица с дължина $n-2$ над $\{1,\ldots,n+1\}\backslash\{k\}$. Съгласно индуктивното предположение, съществува дърво T' с множество от върхове $\{1,\ldots,n+1\}\backslash\{k\}$, такова, че C' е изходът на Код на Prüfer от дърво върху T'.

Ключовото наблюдение е, че, съгласно **Факт2**, Код на Prüfer от дърво върху T връща кода, започващ с k и продължаващ с C'. Но това е точно C. Ерго, има дърво, върху което алгоритъмът връща C. □

Следствие 9

Алгоритъмът Код на Prüfer от дърво реализира сюрекция.

Това е и края на доказателството на Теорема 26. □

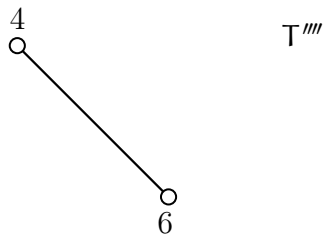
Лема 14 подсказва как да построим дърво от код. Ето илюстрация на работата на този алгоритъм. Нека кодът е $\langle 1,3,4,4\rangle$, който код вече се появи в примера за алгоритъма в обратната посока. Щом дължината е 4, то множеството от върховете е $\{1,2,3,4,5,6\}$. И така, ще генерираме уникалното дърво T с множество от върхове $\{1,2,3,4,5,6\}$, чийто код е $\langle 1,3,4,4\rangle$. Минималното число от $\{1,2,3,4,5,6\}$, което не се среща в $\langle 1,3,4,4\rangle$, е 2. Кодът след изтриване на първия елемент е $\langle 3,4,4\rangle$. И така, $\langle 3,4,4\rangle$ е кодът на някое дърво T' над $\{1,2,3,4,5,6\}\backslash\{2\}$, тоест, над $\{1,3,4,5,6\}$.

Разглеждаме кода $\langle 3,4,4\rangle$ на дърво с множество от върхове $\{1,3,4,5,6\}$. Минималното число от $\{1,3,4,5,6\}$, което не се среща в кода, е 1, а кодът след изтриването на първия елемент е $\langle 4,4\rangle$. И така, $\langle 4,4\rangle$ е кодът на някое дърво T'' над $\{1,2,3,4,5,6\}\backslash\{1,2\}$, тоест, над $\{3,4,5,6\}$.

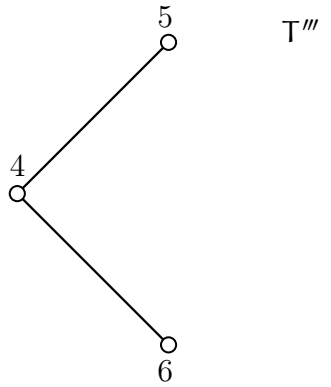
Разглеждаме кода $\langle 4,4\rangle$ на дърво с множество от върхове $\{3,4,5,6\}$. Минималното число от $\{3,4,5,6\}$, което не се среща, е 3, а кодът след изтриването на първия елемент е $\langle 4\rangle$. И така, $\langle 4\rangle$ е кодът на някое дърво T''' над $\{1,2,3,4,5,6\}\backslash\{1,2,3\}$, тоест, над $\{4,5,6\}$.

Разглеждаме кода $\langle 4\rangle$ на дърво с множество от върхове $\{4,5,6\}$. Минималното число от $\{4,5,6\}$, което не се среща, е 5, а кодът след изтриването на първия елемент е $\langle\,\rangle$. И така, $\langle\,\rangle$ е кодът на някое дърво T'''' над $\{1,2,3,4,5,6\}\backslash\{1,2,3,5\}$, тоест, над $\{4,6\}$.

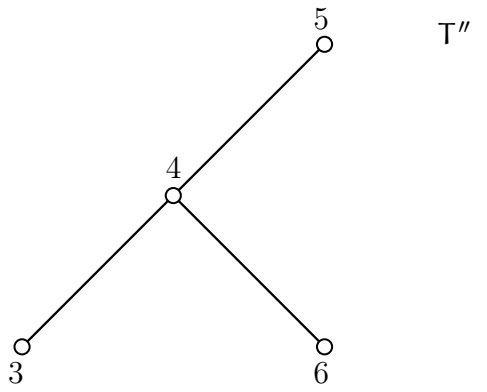
Разглеждаме кода $\langle\,\rangle$ на дърво с множество от върхове $\{4,6\}$. Това е спирачката на рекурсията, така че директно построяваме T'''':



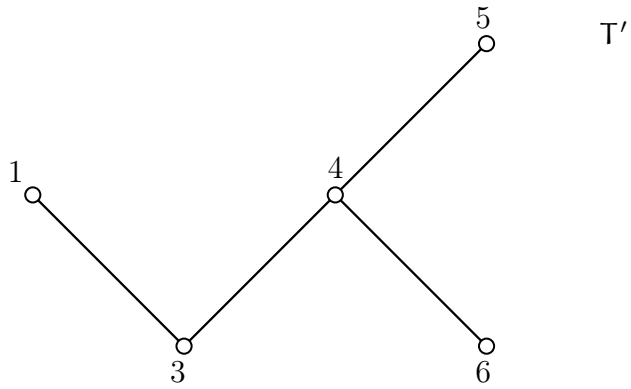
Тогава T''' се получава от T'''' чрез добавяне на връх 5 и реброто $(4, 5)$:



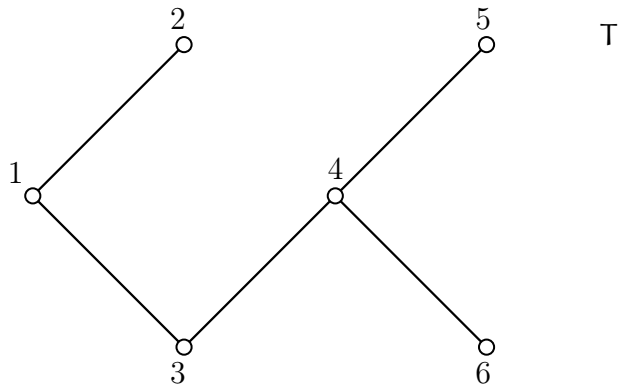
Тогава T'' се получава от T''' чрез добавяне на връх 3 и реброто $(3, 4)$:



Тогава T' се получава от T'' чрез добавяне на връх 1 и реброто $(1, 3)$:



Тогава T се получава от T' чрез добавяне на връх 2 и реброто $(1, 2)$:



Допълнение 15: Други извеждания на формулата на Cayley

Първи начин Броят на покриващите дървета на граф е равен на детерминантата на една матрица, която характеризира графа. Резултатът е в сила дори за мултиграфи (наличието на примки е без значение за покриващите дървета). Този нетривиален резултат е част от алгебричната теория на графите и е открит още през 19 век от великия физик Gustav Kirchhoff. Резултатът присъства имплицитно в негова статия [38] (превод на английски има в [37]). Подробно изследване на работата на Kirchhoff има в [36]. За подробно доказателство на резултата на Kirchhoff вижте [56, глава 9, стр. 135].

Определение 60: Матрица на Laplace на граф

Нека $\mathsf{G} = (\{1, \ldots, \mathfrak{n}\}, \mathsf{E})$ е граф. *Матрицата на Laplace* на граф G е квадратна, $\mathfrak{n} \times \mathfrak{n}$, симетрична матрица L, където

$$\mathsf{L}[\mathfrak{i}, \mathfrak{j}] = \begin{cases} \mathsf{d}(\mathfrak{i}), & \text{ако } \mathfrak{i} = \mathfrak{j}, \\ -1, & \text{ако } \mathfrak{i} \neq \mathfrak{j} \text{ и } (\mathfrak{i}, \mathfrak{j}) \in \mathsf{E}, \\ 0, & \text{в противен случай} \end{cases}$$

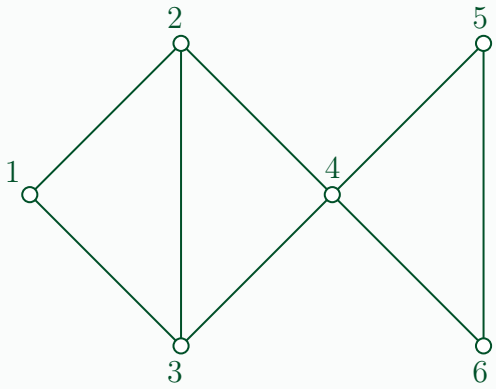
Теорема 27 е от [56, 9.8 Theorem, стр. 141].

Теорема 27: The Matrix-Tree Theorem (за броя на покриващите дървета)

Нека G е граф без примки с матрица на Laplace L. Нека L_0 означава L с последния ред и колона изтрити (или, по-общо казано, с i-ия ред и колона изтрити, за кое да е i). Тогава

$$\det(L_0) = \kappa(G)$$

Като пример да си припомним графа от Фигура 2.75, който, както се убедихме, има точно 24 покриващи дървета.



Матрицата на Laplace за този граф е

$$\begin{bmatrix} 2 & -1 & -1 & 0 & 0 & 0 \\ -1 & 3 & -1 & -1 & 0 & 0 \\ -1 & -1 & 3 & -1 & 0 & 0 \\ 0 & -1 & -1 & 4 & -1 & -1 \\ 0 & 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & -1 & -1 & 2 \end{bmatrix}$$

Съгласно Теорема 27, броят на покриващите го дървета е точно 24:

$$\kappa(G) = \det \begin{bmatrix} 2 & -1 & -1 & 0 & 0 \\ -1 & 3 & -1 & -1 & 0 \\ -1 & -1 & 3 & -1 & 0 \\ 0 & -1 & -1 & 4 & -1 \\ 0 & 0 & 0 & -1 & 2 \end{bmatrix} = 24$$

Следствие от Теорема 27 е следното ([56, Corollary 9.10-b, стр. 142]) твърдение.

Следствие 10: Броят на покриващите дървета и собствените ст-сти на матр. на съседство

Нека G е свързан, d-регулярен, граф без примки с матрица на съседство A. Нека собствените стойности на матрицата на съседство A са $\lambda_1, \ldots, \lambda_{n-1}, \lambda_n$, като $\lambda_n = d$. Тогава

$$\kappa(G) = \frac{(d-\lambda_1)(d-\lambda_2)\cdots(d-\lambda_{n-1})}{n}$$

Следното наблюдение е от [56, Proposition 1.5, стр. 4].

Наблюдение 24: Собств. стойности на матр. на съседство на пълния граф

Собствените стойности на матрицата на съседство за пълния граф K_n са тези:

- -1 с кратност $n-1$ и
- $n-1$ с кратност 1.

От Следствие 10 и Наблюдение 24 веднага следва формулата на Cayley.

Теорема 28: Формула на Cayley, алгебрично доказателство

Броят на покриващите дървета на именуван K_n е

$$\kappa(K_n) = \frac{(n-1-(-1))(n-1-(-1))\cdots(n-1-(-1))}{n} = \frac{n^{n-1}}{n} = n^{n-2}$$

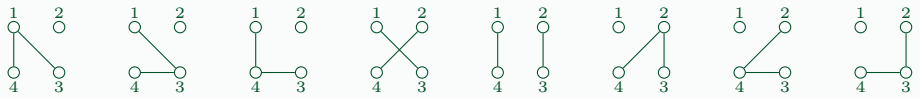
Втори начин Теорема 29 е от [2, стр. 204].

Теорема 29: Броят на именуваните гори по отн. на фиксирано подм-во от върхове

Разглеждаме множество от върхове $V = \{1, \ldots, n\}$. Фиксираме произволно k-елементно подмножество A на V. Тогава броят на именуваните гори над V, състоящи се от точно k дървета, като всеки връх от A е в различно дърво, е:

$$T_{n,k} = \begin{cases} \sum_{i=0}^{n-k} \binom{n-k}{i} T_{n-1,k-1+i}, & \text{ако } n \geqslant k > 1 \\ 0, & \text{ако } n \geqslant 1 \text{ и } k = 0 \\ 1, & \text{ако } n = 0 \text{ и } k = 0 \end{cases} \tag{2.14}$$

Първо да видим пример. Нека $n = 4$, $k = 2$ и $A = \{1, 2\}$.



Вижда се, че $T_{4,2} = 8$.

Доказателство: Началните условия са очевидни. Ще докажем рекурсивната част от формулата. Очевидно няма значение точно кои върхове са в A; значение има само кардиналността на A. БОО, нека $A = \{1, \ldots, k\}$. Фиксираме произволен връх от A. Нека това е връх 1. Индексната променлива i има смисъл на броя съседите на връх 1. Те може да са от 0 до $n-1-(k-1) = n-k$, което дава и границите на сумирането в (2.14).

Изтриваме връх 1 и получаваме гора с $n-1$ върха и с точно $k-1+i$ дървета, защото едно дърво изчезва, а се появяват i нови дървета. Различните гори с $n-1$ върха и $k-1+i$ дървета са $T_{n-1,k-1+i}$. Връщаме връх 1, като трябва да го направим съсед на точно i върха измежду върховете от $V \backslash A$. Но $|V \backslash A| = n-k$, следователно има точно

$\binom{n-k}{i}$ начина да изберем съседите на върнатия връх 1. Всяко свързване на връх 1 с i върха измежду върховете на някоя гора (общо $T_{n-1,k-1+i}$ горй) дава различна гора, което доказва и верността на (2.14).
Това, че $T_{n,n} = 1$ следва от (2.14). □

Резултатът е от [2, стр. 204].

Теорема 30: Пак за броя на именуваните гори по отн. на фиксирано подм-во от върхове

В сила е

$$T_{n,k} = kn^{n-k-1} \tag{2.15}$$

Доказателство: С индукция по (2.14). Базовите случаи $n \geqslant 1, k = 0$ и $n = k = 0$ са очевидни.
В индуктивната стъпка имаме:

$$\begin{aligned}
T_{n,k} &= \sum_{i=0}^{n-k} \binom{n-k}{i} T_{n-1,k-1+i} = \sum_{i=0}^{n-k} \binom{n-k}{i} (k-1+i)(n-1)^{n-1-(k-1+i)-1} \\
&= \sum_{0 \leqslant i \leqslant n-k} \binom{n-k}{i} (k-1+i)(n-1)^{n-k-i-1} \\
&= \sum_{0 \leqslant n-k-i \leqslant n-k} \binom{n-k}{n-k-(n-k-i)} (k-1+(n-k-i))(n-1)^{n-k-(n-k-i)-1} \\
&= \sum_{0 \leqslant i \leqslant n-k} \binom{n-k}{i} (n-i-1)(n-1)^{i-1} \\
&= \sum_{i=0}^{n-k} \binom{n-k}{i} (n-1)^i - \sum_{i=0}^{n-k} \binom{n-k}{i} i(n-1)^{i-1} \\
&= \sum_{i=0}^{n-k} \binom{n-k}{i} (n-1)^i 1^{n-k-i} - \sum_{i=1}^{n-k} \binom{n-k-1}{i-1} \frac{n-k}{i} i(n-1)^{i-1} \\
&= (n-1+1)^{n-k} - (n-k) \sum_{i=0}^{n-k-1} \binom{n-k-1}{i} (n-1)^i 1^{n-k-1-i} \\
&= n^{n-k} - (n-k)(n-1+1)^{n-k-1} = n^{n-k} - (n-k)n^{n-k-1} \\
&= n^{n-k} - n^{n-k} + kn^{n-k-1} = kn^{n-k-1}
\end{aligned} \tag{2.16}$$

□

Теорема 31: Формула на Cayley, комбинаторно доказателство

В контекста на Теорема 29, $T_{n,1}$ очевидно е броят на дърветата над V. Според Теорема 30, $T_{n,1} = 1n^{n-1-1} = n^{n-2}$. Тогава броят на дърветата е n^{n-2}.

2.12 Многосвързаност на графи

2.13 Планарност на графи

Изложението на материала в тази секция е близко до изложението в [27].

В тази секция ще разглеждаме както обикновени графи, така и мултиграфи, но винаги **свързани**. Наличието на паралелни ребра и примки няма значение за планарността, така че в ключовата Теорема на Kuratowski (Теорема 44)—необходимото и достатъчно условие граф да е планарен—графите, които ще разглеждаме, ще са обикновени. От друга страна, Ойлеровата характеризация на планарните графи (Теорема 32) е в сила за мултиграфи, а не само за обикновени графи, така че там ще разглеждаме мултиграфи. Преди да представим някакъв резултат ще уточняваме дали графите са обикновени или мултиграфи.

От друга страна, има смисъл да разглеждаме само свързани графи, независимо дали са обикновени или мултиграфи, защото, ако свързаните компоненти са повече от една, то те могат да се разглеждат една след друга. Нещо повече, някои от резултатите като Ойлеровата характеризация на планарните графи (Теорема 32) не са в сила при повече от една свързани компоненти. И така, разглеждаме само свързани графи или мултиграфи, освен ако изрично не кажем друго.

2.13.1 Определение

В Подсекция 2.13.1 разглеждаме неориентирани мултиграфи.

Подчертаваме, че следното определение дефинира **геометричен обект**. То ползва понятията *проста отворена крива* и *проста затворена крива*. Обикновено в учебниците по графи планарните ребра са само отворени криви. Тук въвеждаме и затворени криви, защото допускаме мултиграфите да имат примки. Ако не допускаме примки, няма причина да допускаме планарни ребра–затворени криви.

Проста отворена крива е, не особено формално казано, несамопресичаща се крива с два края като тази:  . Проста затворена крива е, още по-неформално казано, крива, която "тръгва" от някаква точка и, без да се пресича, се връща в същата точка, примерно така:  . Формалното определение на "проста отворена крива" е: подмножество на равнината, такова че между него и затворения интервал $[0,1]$ има хомеоморфизъм. Формалното определение на "проста отворена крива" е: подмножество на равнината, такова че между него и единичната окръжност има хомеомолфизъм. За определението на "хомеоморфизъм" в този смисъл вижте Определение 71 в Допълнение 22.

Определение 61: Планарно вписване на мултиграф.

Планарно вписване на мултиграф е всяка наредена двойка $\mathfrak{G} = (\mathfrak{V}, \mathfrak{E})$, където $\mathfrak{V} = \{\mathfrak{u}_1, \mathfrak{u}_2, \ldots, \mathfrak{u}_n\}$ е непразно множество от точки в равнината, наречени *планарни върхове*, а $\mathfrak{E} = \{\mathfrak{s}_1, \mathfrak{s}_2, \ldots, \mathfrak{s}_m\}$ е множество от прости отворени криви в равнината, наречени *планарни ребра*. За всяко планарно ребро, ако е отворена крива, то двата му края съвпадат с точно два от планарните върхове, а ако е затворена крива, точно една точка от него съвпада с някой планарен връх. Планарните ребра не се пресичат с изключение на това, че може да имат общи точки–планарни върхове.

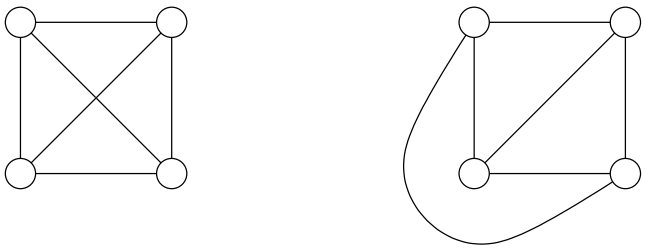
Определение 62 казва, че трябва да има изоморфизъм между планарния мултиграф и планарното му вписване; неслучайно Определение 62 прилича на Определение 40.

Определение 62: Планарен мултиграф.

Нека $G = (V, E)$ е мултиграф. G е *планарен* тогава и само тогава, когато съществува планарно вписване на мултиграф $\mathfrak{G} = (\mathfrak{V}, \mathfrak{E})$, такова че:

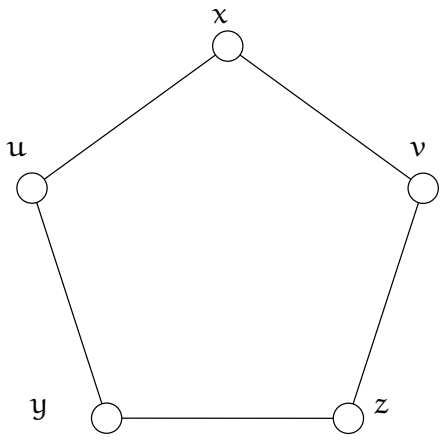
- Съществува биекция $\phi : V \to \mathfrak{V}$.
- Съществува биекция $\psi : E \to \mathfrak{E}$, такава че $\forall e \in E$:
  - ако реброто e не е примка и краищата му са x и y, то $\phi(x)$ и $\phi(y)$ са краищата на $\psi(e)$ в равнината.
  - ако реброто e е примка, инцидентна с върха x, то единственият планарен връх, принадлежащ на $\psi(e)$, е $\phi(x)$.

Както ще докажем след малко, не всеки граф има планарно вписване. Полуформално казано, граф е планарен, ако може да бъде нарисуван в равнината така, че кривите, съответстващи на ребрата, да не се пресичат (освен евентуално в общи краища). Това, че някой граф е нарисуван с пресичане на (планарните си) ребра не означава, че той няма планарно вписване. Например, ето две рисунки на K_4: в лявата има пресичане на ребра, следователно тя не показва планарно вписване, но дясната показва планарно вписване:

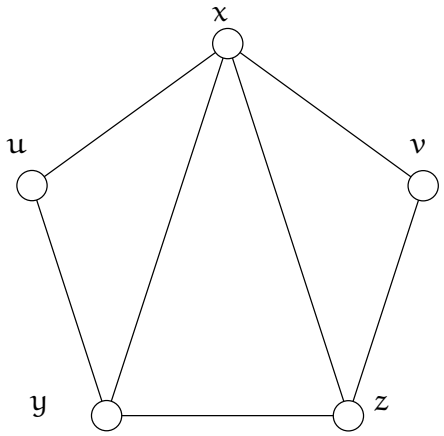


Задачата да се изчисли дали даден граф е планарен или не, тоест дали може да се нарисува в равнината без пресичане на ребра или не, не е тривиална. Следният пример показва това. Нека "$K_5 - e$" означава K_5, от който е изтрито произволно ребро. След малко ще докажем, че K_5 не е планарен, а сега ще покажем, че $K_5 - e$ е планарен.

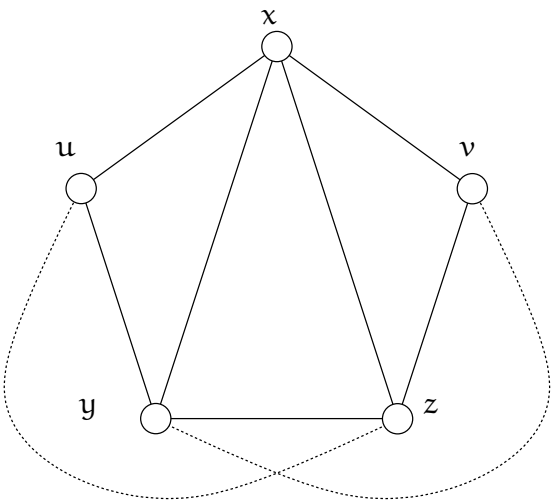
Нека върховете на $K_5 - e$ са x, y, u, v и z. Без ограничение на общността, нека липсващото ребро е между върховете u и v. Да се опитаме да нарисуваме $K_5 - e$ без пресичане на планарни ребра като първо нарисуваме върховете x, v, z, y и u в този ред по часовниковата стрелка като върхове на правилен петоъгълник, след това сложим периферия на петоъгълника от 5 планарни ребра и накрая сложим още 4 планарни ребра-диагонали. Ето разполагането на периферията:



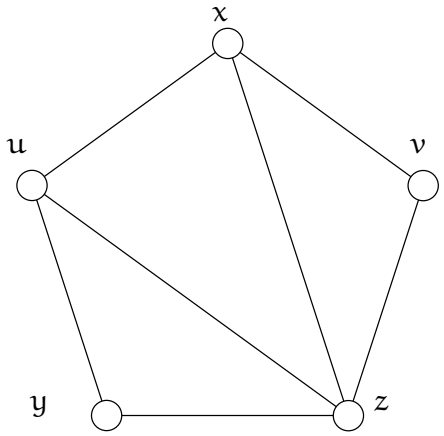
Ако първо сложим планарните ребра (x, y) и (x, z) като диагонали:



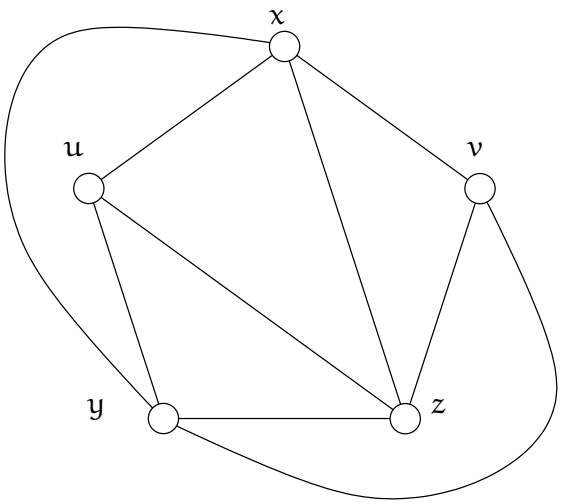
очевидно няма как да сложим останалите две планарни ребра (u, z) и (v, y) без да получим пресичане на планарни ребра:



Ако обаче започнем с (u, z) и (x, z) като диагонали:



можем да добавим и останалите (x, y) и (y, v) без пресичане и да се убедим, че $K_5 - e$ е планарен:



И така, виждаме, че ако разглеждаме планарното вписване като процес, в който планарните ребра се слагат в равнината едно след друго, конкретните им разполагания имат значение. Дори ако графът е планарен, може да не успеем да довършим планарното вписване при неподходящо разполагане на начално сложените планарни ребра. За да опишем коректно този факт обаче ни е необходимо ключовото понятие "лице на планарното вписване", което въвеждаме в Подсекция 2.13.2. Засега правим само следното наблюдение, което ползва понятието "алчен алгоритъм" от Допълнение 9.

Наблюдение 25

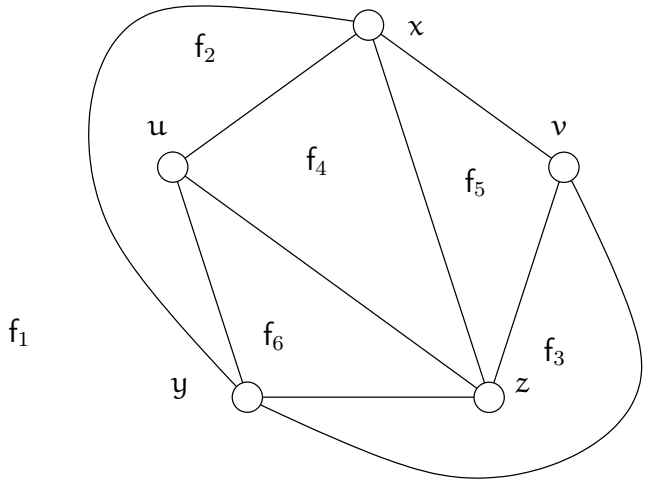
В общия случай не можем да решим задачата за намиране на планарно вписване на планарен граф чрез очевидния алчен алгоритъм: сложи първо планарните върхове по произволен начин (все пак, трябва да са две по две различни точки) и после слагай планарните ребра последователно в произволен ред, така че никое планарно ребро да не пресича вече сложени планарни ребра, докато всички планарни ребра не бъдат сложени.

2.13.2 Лица на планарните вписвания.

Определение 63: Лица на планарно вписване.

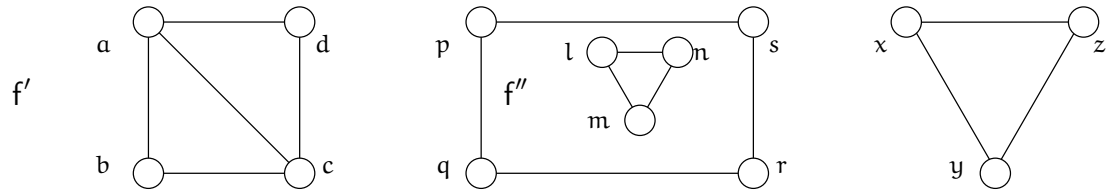
Нека G е планарен граф и $\mathfrak{G}$ е някое негово планарно вписване. Да махнем от равнината всички планарни върхове и ребра. Тази операция води до разпадането на равнината на свързани райони, които наричаме *лицата* на $\mathfrak{G}$. Точно едно от лицата е неограничено – това е *външното лице*, а останалите са *вътрешните лица*.

Например, следното планарно вписване на $K_5 - e$ има лица $f_1, \ldots, f_6$. Външното лице е f_1:



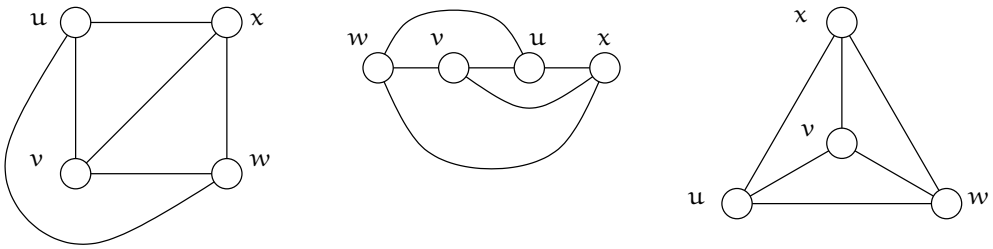
Очевидно е, че при вписванията на свързани планарни графи, всяко лице има точно един *ограждащ цикъл*. Ограждащият цикъл е цикълът от точно тези планарни ребра и върхове, които участват в границата на лицето. В последния пример, ограждащият цикъл на f_1 е цикълът x, y, v, x. Надолу ще видим, че ограждащият цикъл може да е прост или да не е прост.

Ако графът не е непременно свързан, едно лице може да има няколко ограждащи цикъла. Като пример да разгледаме следното планарно вписване на граф с четири свързани компоненти. Лицето f' има 3 ограждащи цикъла: единият е a, b, c, d, a, другият е p, q, r, s, p и третият е x, y, z, x. Лицето f'' има два ограждащи цикъла: единият е p, q, r, s, p и другият е l, m, n, l.



Но ние вече се разбрахме да разглеждаме само свързани графи. При свързаните графи наистина всяко лице има точно един ограждащ цикъл.

Дефиницията на планарно вписване говори за точки и криви в равнината, а това са геометрични понятия. Ние ще гледаме на планарните вписвания не на ниво геометрия, а на по-високо ниво[†]. Конкретното разположение на планарните върхове и конкретните форми на планарните ребра няма да ни интересуват. По причини, които ще станат ясни след малко, следните три планарни вписвания на K_4 ще считаме за еквивалентни, тоест едно и също вписване, нарисувано по три различни начина:



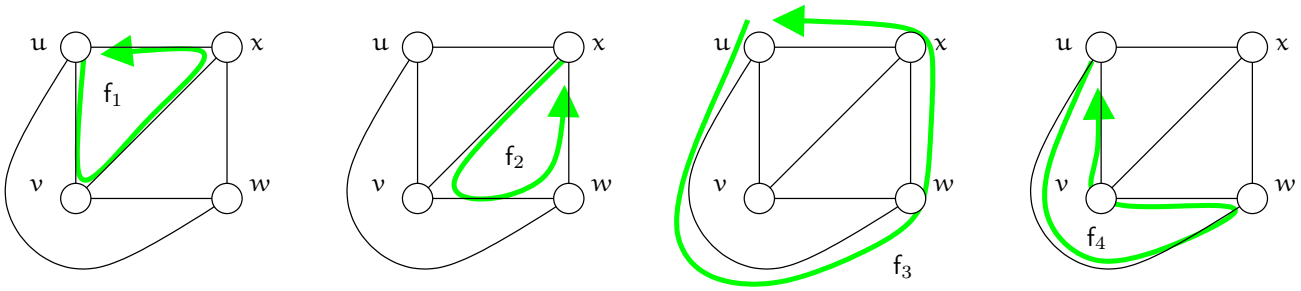
[†]По-високото ниво е топологията, а още по-високо ниво е теоретико-множественото, или още комбинаторното ниво. Топологическо вписване и комбинаторно вписване не е точно едно и също, но за целите на тази лекция ще игнорираме разликата. Тя е илюстрирана в [18, стр. 97].

На лицата няма да гледаме като на конкретни геометрични фигури, а по-общо, като на графови цикли, но цикли в дадена *посока*. Да изберем една посока на въртене в равнината, например обратната на часовниковата стрелка. Тогава всяко лице описваме чрез изреждане на върховете на ограждащия го цикъл във вече избраната посока[†]. Дали посоката е по или срещу часовниковата стрелка е без значение, важното е за всички лица посоката да е една и съща. Всяко планарно вписване ще считаме за определено, ако за всяко лице е казано кой е ограждащият го цикъл (описан в избраната посока). За простота ще считаме, че лице и неговият ограждащ цикъл са синоними. Такова описание на вписването не е геометрично, а е чисто *комбинаторно*. От комбинаторното вписване може да направим конкретна рисунка с конкретна геометрия на точките и линиите, но това вече е задача на изчислителната геометрия.

Да разгледаме пак планарно вписване на $K_4 = \big(\{u, v, w, x\}, \{(u, v), (u, w), (u, x), (v, w), (v, x), (w, x)\}\big)$. То се идентифицира чрез четирите си лица:

$$f_1 = u, v, x, u$$
$$f_2 = x, v, w, x$$
$$f_3 = u, w, x, u$$
$$f_4 = u, w, v, u$$

За да се убедим, че лицата имат такива описания, да ги разгледаме подробно едно по едно:



Дали в описанието на лицата ще записваме началния връх два пъти, както правим тук, или веднъж, например $f_1 = u, v, x$, не е съществено, а е въпрос на избор.

Заслужава да се отбележи, че по отношение на комбинаторното описание на вписванията, външното лице не се отличава от другите лица по нищо. Този факт има и друга интерпретация. Планарните графи са точно графите, които могат да бъдат вписани в сферата. Това се показва тривиално със *стереографска проекция* между равнината и сферата.

Допълнение 16: Стереографска проекция

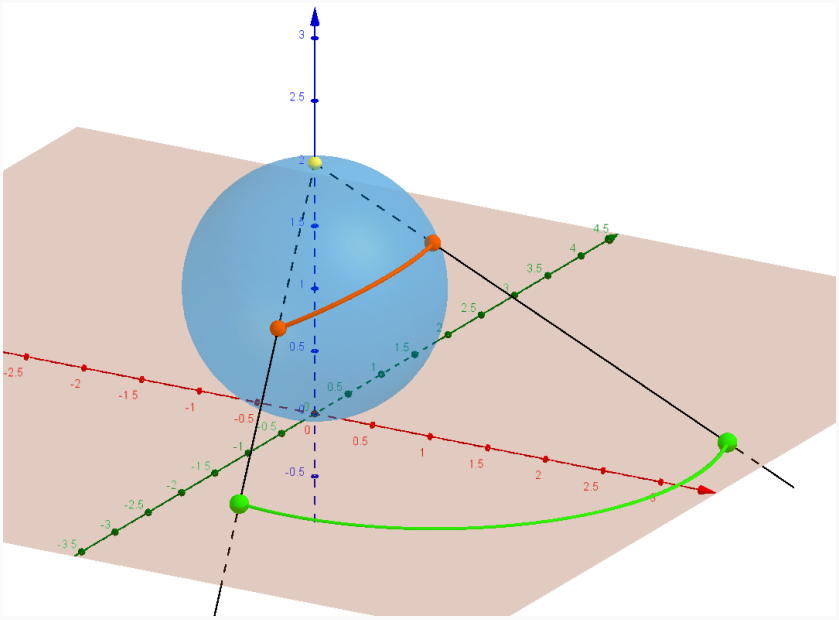
[†]Ако не използваме термина "цикъл", можем да кажем неформално, че за всяко лице описанието е последователността от върхове, които би видяло "двумерно същество" (живеещо в равнината), което обикаля систематично дадено лице по границата в избраната посока, докато не се върне там, откъдето е започнало обиколката.

Определение 64: Стереографска проекция

Да си представим тримерното Евклидово пространство $\mathbb{R}^3$ и в него, Евклидовата равнина $\mathcal{R} = \{(x, y, 0) \,|\, x, y \in \mathbb{R}\}$. Да си представим сферата $\mathcal{S} = \{x \in \mathbb{R}^3 \,|\, \text{dist}(x, (0, 0, 1)) = 1\}$. Очевидно $\mathcal{R} \cap \mathcal{S} = (0, 0, 0)$. *Стереографска проекция* е биекцията, която изобразява $\mathcal{S} \backslash \{(0, 0, 2)\}$ в $\mathcal{R}$ по следния начин: за всяка точка $X \in \mathcal{S} \backslash \{(0, 0, 2)\}$ построяваме лъча с начало точка $(0, 0, 2)$, който съдържа точка X. Този лъч пресича $\mathcal{R}$ в някаква точка Y, която е образът на X. И така, цялата сфера без $(0, 0, 2)$ се изобразява в равнината, а обратната функция изобразява равнината в сферата без $(0, 0, 2)$.

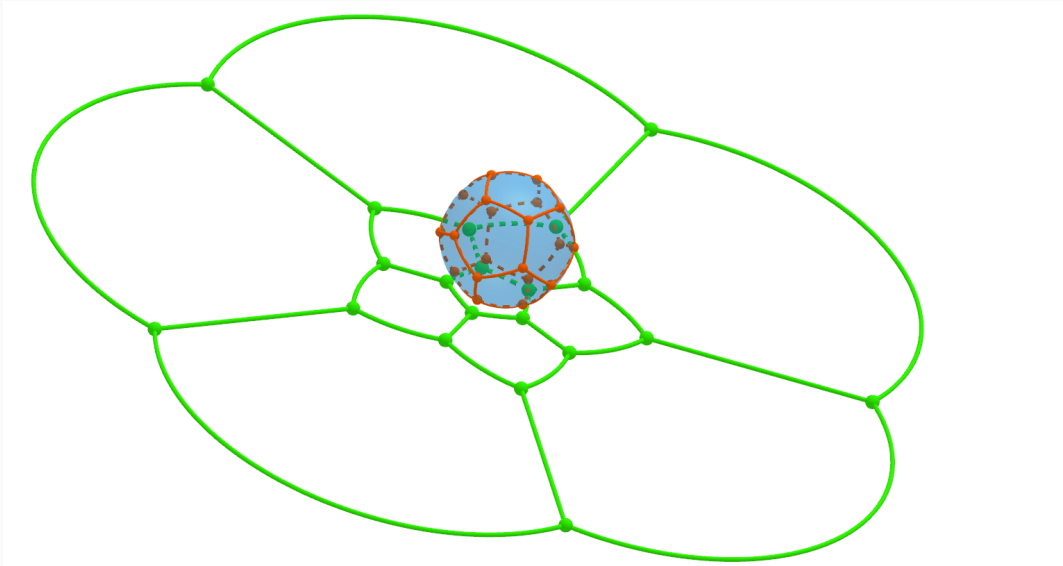
Фигура 2.76 илюстрира понятието стереографска проекция. На нея, "северният полюс" $(0, 0, 2)$ е жълтата точка, върху сферата е нарисувана оранжева дъга, а в равнината е нарисуван образът на дъгата в ярко зелено.

Фигура 2.76 : Стереографска проекция.



По-сложна стереографска проекция е показана на Фигура 2.77, която показва сфера със сферично вписване на графа-додекаедър и стереографска проекция, която изобразява сферичното вписване на додекаедъра (в оранжево) в равнинно вписване (в зелено). Графът-додекаедър очевидно отговаря на многостена-додекаедър, който е показан на Фигура 2.78 в Допълнение 18.

Фигура 2.77 : Стереографска проекция на додекаедъра.



Наблюдение 26: Има сферично вписване т.с.т.к. има планарно вписване

Всеки граф може да бъде нарисуван в равнината без пресичане на планарни ребра тогава и само тогава, когато може да бъде нарисуван върху сферата без пресичане на "сферичните ребра":

- на всяко планарно вписване очевидно съответства поне едно сферично вписване, а именно това, което се получава от обратната функция на стереографската проекция,
- а на всяко сферично вписване, в което "северният полюс" не се съдържа във вписването[a], съответства поне едно планарно вписване, а именно това, което се получава от стереографската проекция.

[a] Ако "северният полюс" е точка от сферичното вписване, винаги можем да завъртим вписването по такъв начин, че "северният полюс" да се окаже точка от някое лице на вписването.

Тъй като върху сферата за външно лице не може да се говори, очевидно външното лице в равнината не е съществено различно от другите лица и всяко лице от сферичното вписване може да бъде проектирано върху външно лице в равнината при подходяща стереографска проекция. Това е важно и заслужава да бъде казано отново.

Наблюдение 27: Условност на избора на външно лице

В комбинаторното описание на кое да е планарно вписване външно лице няма. Всяко геометрично планарно вписване има точно едно външно лице, но ако от комбинаторното описание правим геометрично вписване, можем да изберем което искаме комбинаторно лице за външно.

От изложеното дотук читателят може да остане с впечатлението, че:

① за всеки планарен граф, броят на лицата е един и същи за всяко вписване,

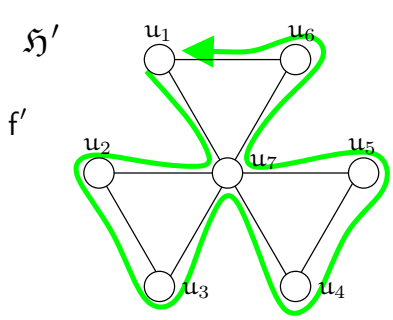
② за всеки планарен граф, лицата са едни и същи за всяко вписване,

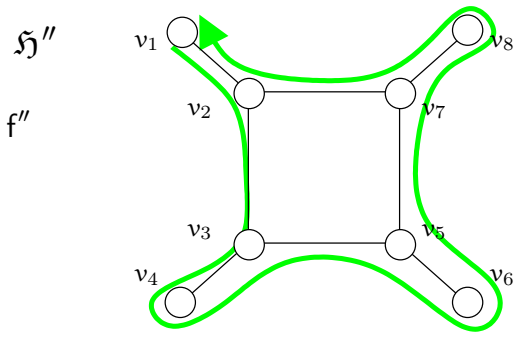
③ за всяко вписване, всеки две лица са различни в комбинаторния смисъл, и

④ лицата винаги имат прости ограждащи цикли.

Както ще видим, ① е вярно твърдение—което ще покажем в подсекция 2.13.3—а ②, ③ и ④ не са верни и сега ще ги опровергаем.

Да разгледаме твърдение ④. Всяко лице има *прост* ограждащ цикъл тогава и само тогава, когато графът има поне три върха и няма срязващи върхове. Ще оставим този факт без доказателство, като само ще дадем два примера за планарни графи H' и H'', всеки от които има поне един срязващ връх. Ще нарисуваме планарните им вписвания, $\mathfrak{H}'$ и $\mathfrak{H}''$ с външни лица съответно $\mathfrak{f}'$ и $\mathfrak{f}''$.



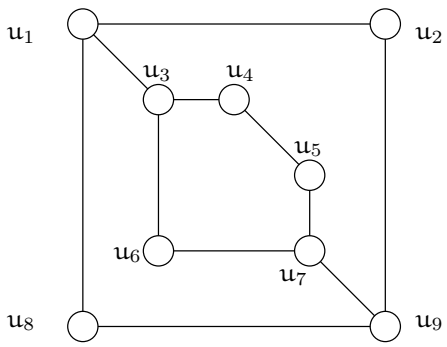


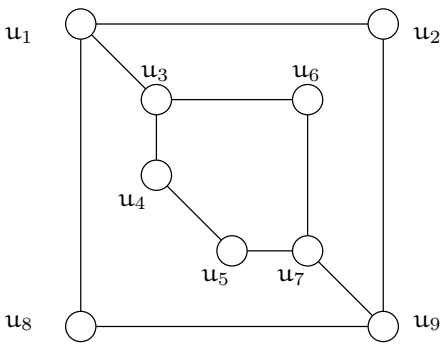
Очевидно

$$\mathfrak{f}' = u_1, u_7, u_2, u_3, u_7, u_4, u_5, u_7, u_6, u_1$$
$$\mathfrak{f}'' = v_1, v_2, v_3, v_4, v_3, v_5, v_6, v_5, v_7, v_8, v_7, v_2, v_1$$

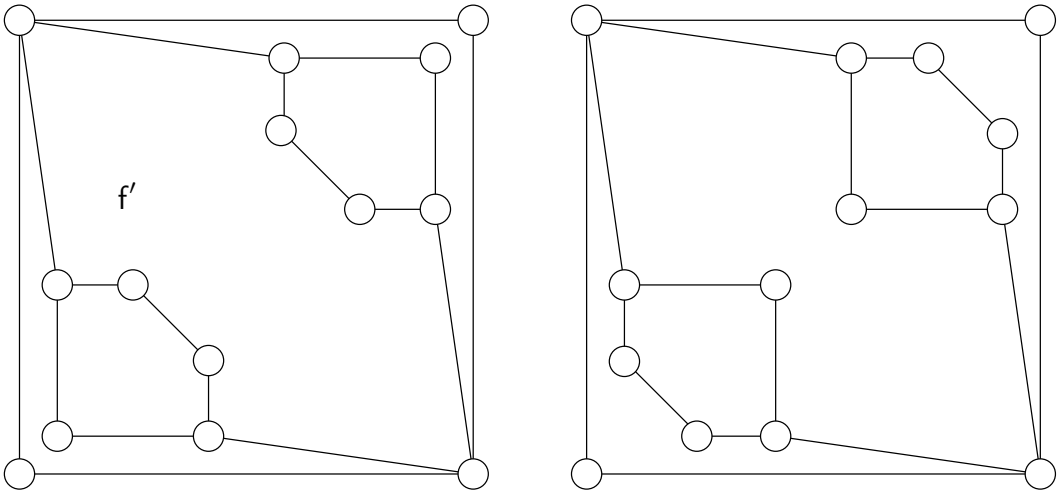
тоест ограждащите ги цикли не са прости.

Сега ще опровергаем твърдение ② като покажем, че на един и същи планарен граф може да съответстват различни планарни вписвания. При това, различни не като геометрия, а като комбинаторни описания на лицата. Да разгледаме следните две планарни вписвания на един и същи планарен граф:



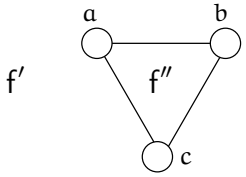


За да се убедим, че вписванията са различни, достатъчно е да забележим, че във вписването отляво има лице, в което присъстват u_8 и u_6, а във вписването отдясно няма такова лице. Последният пример показва различни вписвания на един и същи граф, но едното от тях може да бъде получено от другото чрез преименуване на върхове: ако разменим местата на имената u_2 и u_8 във вписването вдясно, ще получим вписването вляво. Може да се дадат примери за различни вписвания на един и същи граф, които не могат да бъдат получени едно от друго чрез просто преименуване, например:



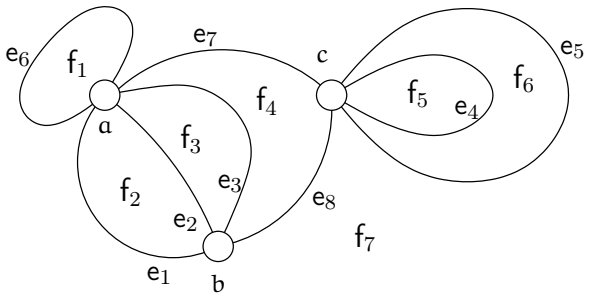
Графът е един и същ и в двете двете вписвания, и броят на лицата е 6, но вписването вляво има лице f' с 10 върха, а нито едно от шестте лица вдясно не е с 10 върха. Показахме, че лицата може да съответстват на цикли, които не са прости, както и че един и същи планарен граф може да има различни планарни вписвания.

И накрая ще опровергаем твърдение ③. Да разгледаме следното планарно вписване на K_3:



И f', и f'' имат описание a, c, b, a. Тъй като идентифицираме лице с неговото описание, излиза, че двете лица, в комбинаторния смисъл, не са различни.

Всички разгледани примери досега бяха на обикновени графи. Да разгледаме и един пример на планарно вписване на мултиграф с паралелни ребра и примки:



Мултиграфът има върхове a, b и c и ребра e_1, …, e_8, три от които са примки (e_6 към връх a и e_4 и e_5 към връх c), а други три са паралелни (e_1, e_2 и e_3 с краища a и b). Показаното планарното вписване има седем лица f_1, …, f_7, които описваме така:

$$
\begin{aligned}
f_1 &= a, e_6, a \\
f_2 &= a, e_1, b, e_2, a \\
f_3 &= a, e_2, b, e_3, a \\
f_4 &= a, e_3, b, e_8, c, e_7, a \\
f_5 &= c, e_4, c \\
f_6 &= c, e_5, c \\
f_7 &= a, e_1, b, e_8, c, e_5, c, e_7, a, e_6, a
\end{aligned}
$$

Тъй като става дума за мултиграф с паралелни ребра, в описанията на циклите участват и имената на ребрата (освен имената на върховете).

2.13.3 Характеристика на Euler. Следствия от нея.

Отново разглеждаме мултиграфи.

Теорема 32: Теорема на Euler

За всеки свързан планарен мултиграф G с n върха и m ребра е вярно, че всяко планарно вписване на G има един и същи брой лица, да речем f лица, като е изпълнено

$$n - m + f = 2 \tag{2.17}$$

Това равенство е известно като Ойлерова характеристика.

Казвайки "характеристика", имаме предвид характеристика на равнината. В Секция 2.14 разглеждаме други двумерни повърхнини, за вписванията на графи, в които (2.17) не е в сила.

Доказателство: Преписваме твърдението така:

$$f = m - n + 2 \tag{2.18}$$

Нека планарното вписване на G се казва $\mathfrak{G}$. Ще докажем (2.18) с индукция по f. Базовият случай е $f = 1$ (забележете, че $f = 0$ е невъзможно). $\mathfrak{G}$ да има точно едно лице и мултиграфът да е свързан очевидно влече, че G е дърво. Щом G е дърво, прилагаме Лема 11 и заключаваме, че $m = n - 1$. Заместваме f с 1 и m с $n - 1$ в (2.18) и получаваме $1 = (n - 1) - n + 2$. ✓

Индуктивното предположение е, че (2.18) е в сила за всички планарни вписвания с по-малко от f лица, за някакво $f > 1$. Разглеждаме граф G с m ребра и n върха, чието планарно вписване $\mathfrak{G}$ има точно f лица, където $f \geqslant 2$. Нека m' и n' са съответно броят на планарните ребра и планарните върхове на $\mathfrak{G}$. Очевидно, $m' = m$ и $n' = n$. Избираме произволно ребро e от G, което не е мост—такова ребро трябва да има, в противен случай $\mathfrak{G}$ не би имал повече от едно лице—и изтриваме от $\mathfrak{G}$ неговото съответно планарно ребро e'. Преди изтриването на e' от $\mathfrak{G}$, в $\mathfrak{G}$ е имало точно две различни лица, за които e' е било общо (това е очевидно). След изтриването на e' тези две лица се сливат в едно лице и по този начин броят на лицата намалява с единица. С други думи, $\mathfrak{G} - e'$ има точно $f - 1$ лица. Тогава индуктивното предположение е приложимо за $\mathfrak{G} - e'$. Броят на ребрата на $\mathfrak{G} - e'$ е $m' - 1$, а броят на върховете му е n'. От индуктивното предположение:

$$f - 1 = (m' - 1) - n' + 2$$

Но това е същото като $f = m' - n' + 2$. Припомняме си, че $m' = m$ и $n' = n$ и веднага получаваме (2.18). □

Допълнение 17: Върху доказателствата по индукция

Да разгледаме отново предложеното доказателство по индукция на Теорема 32. В него правим нещо като "стъпка назад": разглеждайки произволен граф, чието вписване има точно f лица, махаме ребро, с което сливаме две лица и броят на лицата става $f - 1$. Това е "стъпката назад". След нея вече индуктивното предположение е приложимо и довършването на доказателството е елементарно. Може да се запитаме: а дали не може да се мине без тази стъпка назад? Дали не може доказателството, пак по индукция по броя на лицата, да се направи по следния начин? Да наречем "**Д1**" следния опит за доказателство на Теорема 32.

Д1

- Базата е същата: $f = 1$ и пак става дума за дърво.
- Нека за някое f разгледаме произволен планарен мултиграф G, който има планарно вписване $\mathfrak{G}$ с точно f лица. Нека G има m ребра и n върха. Да допуснем, че е изпълнено $f = m - n + 2$. Разглеждаме произволно лице F на $\mathfrak{G}$ и произволни два планарни върха u и v от F. Щом са върхове от едно и също лице, можем да добавим едно **ново** планарно ребро e' към $\mathfrak{G}$, поставяйки e' изцяло във F, без e' да докосва или пресича нищо друго от $\mathfrak{G}$, освен че двата края на e' се идентифицират с u и v. По този начин e' "разсича" F на две нови лица F_1 и F_2. Тъй като самото F престава да съществува, броят на лицата нараства с единица. Новото планарно вписване има $f + 1$ лица, $m + 1$ планарни ребра и n планарни върха. Дали $f+1 = (m+1)-n+2$? Да, това следва директно от индуктивното предположение с добавяне на единица и от двете страни на равенството.

Д1 е формално некоректно, а да бъде изтъкната същината на тази формална некоректност е причината за написването на това допълнение. Това, което липсва на **Д1**, е едно друго доказателство. **Д1** неявно допуска, че **всеки** свързан планарен мултиграф с n върха и $m \geqslant n - 1$ ребра може да бъде генериран по следния начин: генерираме негово планарно вписване, започвайки от планарно дърво с n върха и слагайки едно по едно $m - n + 1 = m - (n - 1)$ планарни ребра, всяко от които се слага между два върха от едно и също лице (което гарантира, че можем да сложим новото планарно ребро без да пресичаме непозволено вече сложени планарни ребра), при които вписването остава планарно. Това неявно допускане трябва да се докаже, за да стане предложеното доказателство истинско. Лесно се вижда, че въпросното неявно допускане касае алтернативно, индуктивно определение на "свързано планарно вписване".

Определение 65: Индуктивно определение на свързано планарно вписване върху фиксирано множество от планарни върхове.

Нека е дадено непразно крайно множество $\mathfrak{V}$ от планарни върхове.

❶ **База** Всяко планарно дърво върху $\mathfrak{V}$ е свързано планарно вписване върху $\mathfrak{V}$.

❷ **Индуктивна стъпка** Нека $\mathfrak{D}$ е свързано планарно вписване върху $\mathfrak{V}$. Нека F е произволно лице спрямо $\mathfrak{D}$. Нека $\mathfrak{u}$ и $\mathfrak{v}$ са произволни, не непременно различни, планарни върхове от F. Слагаме **ново** планарно ребро $\mathfrak{e}'$ с краища $\mathfrak{u}$ и $\mathfrak{v}$, такова че единствените общи точки между $\mathfrak{e}'$ и планарните върхове и ребра от $\mathfrak{D}$ са именно $\mathfrak{u}$ и $\mathfrak{v}$.

❸ **Заключение** Нищо, което не може да бъде генерирано с краен брой прилагания на ❶ и ❷, не е свързано планарно вписване върху $\mathfrak{V}$.

Лема 15

Определение 65 е еквивалентно на Определение 61, ако в Определение 61 се ограничим до свързани планарни вписвания.

Доказателство: В едната посока доказателството е тривиално – очевидно процедурата от Определение 65 генерира само свързани планарни вписвания върху $\mathfrak{V}$. Това може да се докаже и по индукция, но едва ли има смисъл; наистина е очевидно. В другата посока трябва да докажем, че всяко свързано планарно вписване $\mathfrak{G}$ върху $\mathfrak{V}$ може да се генерира от тази процедура. Наистина, започвайки от произволно такова вписване $\mathfrak{G}$, изпълняваме обратния алгоритъм, който е много аналогичен на Алгоритъм 3:

- Проверяваме дали вписването е на дърво, тоест дали има цикли, тоест дали лицата са точно едно.
- Ако да, терминираме.
- Ако не, намираме планарно ребро, принадлежащо на две лица—такова трябва да има—и го изтриваме, сливайки двете лица в едно, след което пак отиваме на проверката.

Имайки "обратния алгоритъм", редуцираме даденото вписване $\mathfrak{G}$ до планарно дърво $\mathfrak{D}$, след което очевидно с краен брой прилагания на индуктивната стъпка от Определение 65 можем да възстановим $\mathfrak{G}$ от $\mathfrak{D}$ като просто слагаме ребрата в обратен ред на реда на махането им. □

И така, ако съчетаем **Д1** с Определение 65 и Лема 15, ще имаме валидно доказателство на Теорема 32. Но сложността на този подход надхвърля сложността на предложеното доказателство на теоремата. Тук излагаме този възможен подход само за да илюстрираме нещо важно за доказателствата по индукция – понякога "очевидното" просто доказателство е невалидно, защото се основава на индуктивна дефиниция, която не е изложена или най-малкото, за която не е доказано, че е еквивалентна на "нашата". В конкретния случай се оказа, че наистина имплицитната по отношение на **Д1** дефиниция е еквивалентна на "нашата". Но Хипотеза 1 на стр. 152 е пример за това, че в

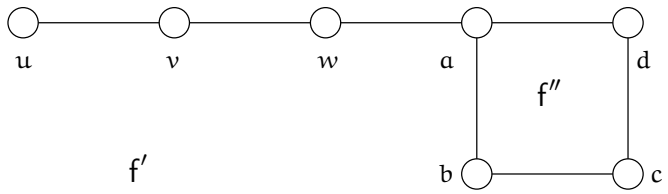
някои случаи имплицитната дефиниция не е еквивалентна на "нашата" и по този начин просто изглеждащото "доказателство" всъщност е невалидно.

И накрая, забележете, че предложеното доказателство на Теорема 32 по същество е като съчетаване на **Д1** с Определение 65 и Лема 15, но по икономичен и елегантен начин: докато Лема 15 изпълнява всички "стъпки назад", то доказателството на Теорема 32 прави само една "стъпка назад".

Определение 66: Степен на лице на планарно вписване.

За всяко планарно вписване на някой планарен мултиграф, за всяко лице t, *степента на* t е дължината на цикъла, ограждащ t. Степента на t бележим с $d(t)$.

Като пример ще разгледаме степените на двете лица f' и f'' на следното вписване:



Степента на f'' е очевидно 4. Степента на f' е 10, тъй като ограждащият цикъл $u, v, w, a, b, c, d, a, w, v, u$, който не е прост, има дължина 10.

Лема 16

За всеки планарен мултиграф G с m ребра, ако $\mathfrak{G}$ е произволно планарно вписване на G с f на брой лица $t_1, t_2, \ldots, t_f$, в сила е:

$$\sum_{i=1}^{f} d(t_i) = 2m$$

Доказателство: Всяко ребро се брои точно два пъти при сумирането на степените на лицата. □

Този резултат остава в сила дори когато G е тривиален граф: тогава той има един връх, нула ребра и планарното му вписване има едно лице (с нула ограждащи ребра). Наистина, $\sum_{i=1}^{1} 0 = 2 \times 0$.

Допълнение 18: Характеристиката на Euler и Платоновите тела

Теорема 32 има интересен геометричен аспект. Теоремата в този си вид е твърдение за мултиграфи графи с възможни примки. В частност, тя е в сила за обикновени графи. Но тогава тя е в сила и за многостени по следната причина. Многостените са геометрични обекти, но на всеки многостен M естествено съответства обикновен граф G_M: върховете на G_M съответстват на върховете на M, а ребрата на G_M съответстват ръбовете на многостена. G_M е планарен, понеже:

- многостенът може да бъде трансформиран в сфера чрез континуална трансфор-

мация, така че самият многостен естествено задава вписване на графа в сферата;

- както вече отбелязахме в Наблюдение 26, графите, които може да бъдат вписани в сферата са точно графите, които може да бъдат вписани в равнината.

Следователно, лицата на многостена съответстват на лицата на вписването.

Наблюдение 28: Ойлерова характеристика на многостените

Приложена за многостени, Теорема 32 казва, че за всеки многостен:

$$\text{броят на върховете} \;-\; \text{броят на ребрата} \;+\; \text{броят на лицата} \;=2$$

Числото 2 се нарича *Ойлерова характеристика на многостените.*

По-общо казано, Ойлеровата характеристика описва свойство на повърхнината, в която се вписват графите, съответстващи на многостените—тази повърхнина е сферата. Както ще видим в Секция 2.14, съществуват повърхнини, по-сложни от сферата, например тороидът, чиято характеристика не е 2. Оригиналната статия на Ойлер на латински, както и преводи на английски и немски, може да бъдат намерени *на този сайт* (вж. [23]).

От особен интерес са *регулярните многостени*, наречени още *Платонови тела*[a].

Определение 67: Регулярен многостен

Регулярен е всеки многостен $\mathfrak{M}$, такъв че:

- стените на $\mathfrak{M}$ са конгруентни регулярни многоъгълници и
- във всеки връх на $\mathfrak{M}$ се срещат един и същи брой стени.

Добре известен факт, който ще докажем след малко, използвайки теорията на графите, е, че съществуват точно пет Платонови тела:

1. регулярният тетраедър,
2. кубът, който може да наречем регулярен хексаедър,
3. регулярният октаедър,
4. регулярният додекаедър и
5. регулярният икосаедър.

Във всяко от тези имена, коренът е гръцката дума ἕδρα, която означава *сядане*, *седéне*, и оттам *лице на геометрично тяло*. Префиксът пред корена идва от броя на стените на съответното тяло:

1. *тетра-* идва от старогръцката дума τέτρας, която означава *четири*,
2. *хекса-* идна от старогръцката дума ἕξ, която означава *шест*,
3. *окта-* идва от старогръцката дума ὀκτω, която означава *осем*,

4. *додека-* идва от старогръцката дума δώδεκα, която означава *дванадесет*, и
5. *икоса-* идва от старогръцката дума εἴκοσι, която означава *двадесет*.

Фигура 2.78 показва петте Платонови тела. Фигурата е взета от *уикипедия*.

Фигура 2.78 : Петте платонови тела.

тетраедър хексаедър октаедър додекаедър икосаедър

Ще покажем, че няма други Платонови тела, разглеждайки ги като графи, вписани в сферата (тоест, планарни графи). За целта ще използваме Теорема 32. Да разглеждаме произволно Платоново тяло, като всъщност разглеждаме съответния му граф $G = (V, E)$, а също така разглеждаме и вписването на G в сферата. Нека F е множеството от лицата на това вписване. Тъй като всички върхове са от една и съща степен, с d_v означаваме степента на кой да е връх. Аналогично, тъй като всички лица на сферичното вписване са от една и съща степен, с d_f означаваме степента на кое да е лице. Нека n, m и f съответно означават броя на върховете, ребрата и лицата на вписването. От Лема 1 знаем, че $\sum_{u \in V} d(u) = 2m$. Тогава

$$nd_v = 2m \tag{2.19}$$

От Лема 16 знаем, че $\sum_{t \in F} d(t) = 2m$. Тогава

$$fd_f = 2m \tag{2.20}$$

От Теорема 32 знаем, че

$$n - m + f = 2 \tag{2.21}$$

Замествайки n и f в (2.21) съгласно (2.19) и (2.20), съответно, получаваме

$$\frac{2m}{d_v} - m + \frac{2m}{d_f} = 2 \leftrightarrow \frac{1}{d_v} + \frac{1}{d_f} = \frac{1}{2} + \frac{1}{m} \tag{2.22}$$

Но $m > 0$. Тогава

$$\frac{1}{d_v} + \frac{1}{d_f} > \frac{1}{2} \tag{2.23}$$

От неравенство (2.23) и факта, че $d_v \geqslant 3$ и $d_f \geqslant 3$ лесно извеждаме, че единствените пет възможности за стойностите на d_v и d_f са:

$$d_v = 3, d_f = 3 \tag{2.24}$$
$$d_v = 3, d_f = 4 \tag{2.25}$$
$$d_v = 3, d_f = 5 \tag{2.26}$$
$$d_v = 4, d_f = 3 \tag{2.27}$$
$$d_v = 5, d_f = 3 \tag{2.28}$$

Първата възможност (2.24) съответства на тетраедъра, втората възможност (2.25) съответства на хексаедъра (куба), третата възможност (2.26) съответства на додекаедъра, четвъртата възможност (2.27) съответства на октаедъра и петата възможност (2.28) съответства на икосаедъра.

[a] На английски, *Platonic solids*.

Теорема 33: $m \leqslant 3n - 6$ **при планарните обикновени графи**

За всеки планарен обикновен граф G, не непременно свързан, с поне две ребра, ако n е броят на върховете и m е броят на ребрата, то

$$m \leqslant 3n - 6$$

Доказателство: Нека G е свързан – ако не е свързан, доказваме неравенството за свързаните компоненти и сумираме по компонентите. И така, G е свързан планарен граф. Нека лицата са f на брой, наречени $t_1, t_2, \ldots, t_f$. Имаме:

$$\begin{aligned}
n - m + f &= 2 && \text{съгласно Теорема 32}\\
n - 2 &= m - f && \text{очевидно}\\
3n - 6 &= 3m - 3f && \text{очевидно}\\
3n - 6 &= m + 2m - 3f && \text{очевидно}\\
3n - 6 &= m + \sum_{i=1}^{f} d(t_i) - 3f && \text{Съгласно Лема 16}
\end{aligned}$$

Забележете, че $\sum_{i=1}^{f} d(t_i) - 3f$ е неотрицателно количество за всеки планарен **обикновен** граф с поне две ребра, защото всяко лице има ограждащ цикъл с дължина поне 3, тоест $d(t_i) \geqslant 3$ за всяко i. Щом $\sum_{i=1}^{f} d(t_i) - 3f \geqslant 0$, следва, че $3n - 6 \geqslant m$. □

Следствие 11

K_5 не е планарен граф.

Доказателство: Директно от Теорема 33: K_5 има 10 ребра, 5 върха, и $10 \nleqslant 3 \times 5 - 6$. □

Следствие 12

За всеки планарен обикновен граф, $m < 3n - 6$ тогава и само тогава, когато всяко негово планарно вписване има поне едно лице от степен поне 4. □

И така, горната граница $3n - 6$ за броя на ребрата е достижима, и то точно тогава, когато всички лица са от степен точно 3. Лесно се вижда, че тази горна граница е в сила само за обикновени планарни графи – при планарните мултиграфи няма горна граница за броя на ребрата при фиксиран брой на върховете.

Определение 68: Триангулация.

Всяко планарно вписване, в което всяко лице е от степен 3, ще наричаме *триангулация*.

За удобство и по-кратко изразяване може да игнорираме разликата между триангулация като геометрично понятие, от една страна, и графа, на който е тази триангулация, от друга страна. Тогава терминът "триангулация" ще бъде приложим и за графи; а именно за графите, чието планарно вписване е триангулация. Следният очевидно коректен алгоритъм трансформира произволно свързано планарно вписване на граф в триангулация.

Алгоритъм 6: Триангулиране

Вход: Планарно вписване $\mathfrak{G}$.
Изход: Триангулация.

❶ Ако всички лица на $\mathfrak{G}$ са от степен 3, върни $\mathfrak{G}$.

❷ В противен случай, нека S е произволно лице на $\mathfrak{G}$ от степен $\geqslant 4$. Нека u и v са два различни произволни планарни върха от S, между които няма планарно ребро. Добави към $\mathfrak{G}$ планарно ребро между u и v, така че новото ребро да не пресича други планарни ребра във вътрешни точки. Отиди на ❶.

Съгласно Следствие 12, Триангулиране връща планарно вписване на граф, за който $m = 3n - 6$. Той "раздробява" лицата от степен $\geqslant 4$ на лица от степен точно 3. Ние вече видяхме пример за работата на Триангулиране, без да сме използвали в явен вид името на алгоритъма. На фигурата на стр. 135 е показан граф-цикъл с дължина 5, който не е триангулация (съответно не е вярно, че $m = 3n - 6$, защото $5 \neq 3 \times 5 - 6$). После добавяме ребрата (u, z) и (x, z), но това все още не е триангулация (съответно не е вярно, че $m = 3n - 6$, защото $7 \neq 3 \times 5 - 6$). И накрая добавяме ребрата (x, y) и (y, v) (вж. фигурата на стр. 136), с което полученият граф, който наричаме $K_5 - e$, е триангулация (съответно $m = 3n - 6$, защото $9 = 3 \times 5 - 6$).

Допълнение 19: За индуктивно дефинираните триангулации

Ще разгледаме индуктивно определение на понятието "триангулация". То се оказва некоректно в смисъл, че **не е** еквивалентно на Определение 68. Важният извод тук е, че не всяко индуктивно определение на вече дефинирано понятие, което индуктивно определение звучи смислено, е непременно еквивалентно на първоначалното не-индуктивно определение.

Всяко индуктивно определение е алгоритъм, който генерира елементите на някакво множество, започвайки от **база**, която се състои от един или повече елемента на множеството, и прилагайки **индуктивна стъпка**, която—ако **вече** са генерирани някакви елементи на множеството—генерира нови елементи от досега генерираните, използвайки една или повече **присъединителни операции**. В общия случай, индуктивната стъпка се прилага неограничен брой пъти, защото по правило индуктивно генерираните множества са безкрайни, но всеки отделен елемент на множеството се генерира чрез краен брой стъпки. Това ни позволява да казваме "алгоритъм", а не "процедура": алгоритмите задължително трябва да завършват след краен брой стъпки.

Алгоритъм 7: Индуктивно Триангулиране

Вход: Равнината без нищо вписано в нея.
Изход: Триангулация.

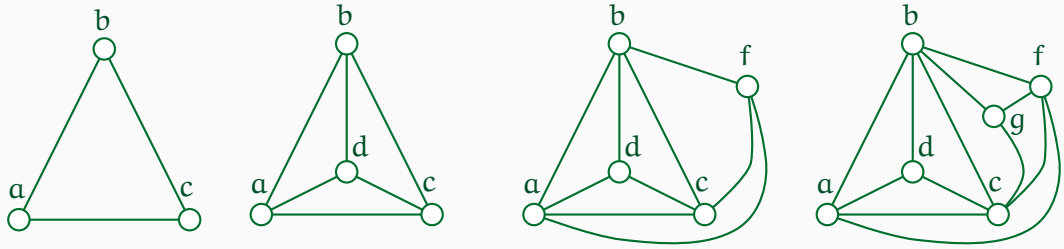
❶ (База) Ако в равнината не е вписано нищо, сложи планарно вписване на K_3.

❷ (Индуктивна стъпка) Ако вече е вписана някаква триангулация $\mathfrak{G}$, вземи произволно нейно лице S, сложи в негова вътрешна точка един нов планарен връх x и свържи x с трите върха на S чрез планарни ребра, които не се пресичат, и така получи нова триангулация $\mathfrak{G}'$.

Присъединителната операция в случая е добавяне на нов връх и нови ребра, инцидентни с него.

Фигура 2.79 илюстрира работата на този алгоритъм.

Фигура 2.79 : Алгоритъм, строящ триангулации.



Фиг.2.79(а): започваме с 3-клика.

Фиг.2.79(б): слагаме нов връх d и три ребра.

Фиг.2.79(в): слагаме нов връх f и три ребра.

Фиг.2.79(г): слагаме нов връх g и три ребра.

Читателят може да се изкуши да допусне, че Индуктивно Триангулиране може да генерира всяка възможна триангулация, аналогично на Триангулиране (вж. Алгоритъм 6).

Хипотеза 1

Алгоритъм Триангулиране може да генерира всяка възможна триангулация.

Ако Хипотеза 1 беше вярна, то доказателството на теоремата за четирите цвята (вж. Теорема 7) би било абсолютно тривиално. Тривиално упражнение е да се докаже със структурна индукция, че обектите, генерирани от Индуктивно Триангулиране, са оцветими с не повече от четири цвята:

- В базата на индуктивната дефиниция, очевидно K_3 е 4-оцветим;
- в индуктивната стъпка, ако допуснем, че триангулацията $\mathfrak{G}$ е 4-оцветима, то планарните върхове на лицето S ползват не повече от 3 от тях и имаме "свободен" цвят, в който да оцветим новодобавения връх x, така че $\mathfrak{G}'$ получава оцветяване с не повече от четири цвята, такова че на всяко планарно ребро двата края са различни цветове.

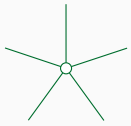
Очевидно оцветяването на планарното вписване с не повече от четири цвята определя 4-оцветяване на съответния планарен граф-триангулация. И така, графите-триангулации, които Триангулиране генерира, са 4-оцветими. Съгласно Хипотеза 1, това са всички триангулации. Очевидно всеки планарен граф може да се получи от някакъв граф-триангулация чрез изтриване на нула или повече ребра. Но изтриването на ребра не може да промени 4-оцветимостта. Следователно, всеки планарен граф е 4-оцветим.

Ясно е, че доказателството на теоремата за четирите цвята не може да е толкова просто. Невъзможно е водещи математици в света да се опитват да докажат теоремата с десетилетия и да не забележат, че има доказателство със сложността на задача за домашно. Хипотеза 1 не може да е вярна. Сега ще опровергаем Хипотеза 1 с контрапример—триангулация, която не може да бъде генерирана от Индуктивно Триангулиране. Ще започнем от факта, че всеки планарен граф има връх от степен $\leqslant 5$ (вж. Теорема 35, която е надолу в тази секция). Първо да се запитаме, може ли да има планарен граф, в който **всеки** връх е от степен точно 5? От най-общи съображения, такъв граф трябва да има поне 6 върха, а именно някакъв връх от степен 5 и неговите 5 съседа. В действителност, такъв граф $G = (V, E)$ трябва да има поне 12 върха по следните причини:

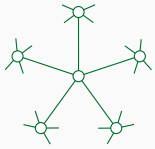
- Както вече знаем, $m \leqslant 3n - 6$.
- От друга страна, знаем, че $\sum_{v \in V} d(v) = 2m$, а $\sum_{v \in V} d(v)$ трябва да е $5n$, така че $m = \frac{5n}{2}$.

От $m \leqslant 3n-6$ и $m = \frac{5n}{2}$ веднага следва $n \geqslant 12$. А дали тази долна граница е достижима? Тоест, дали има 5-регулярен планарен граф с 12 върха? Фигура 2.80 показва, че такъв граф има.

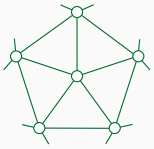
Фигура 2.80 : Конструиране на планарен 5-регулярен граф с 12 върха.



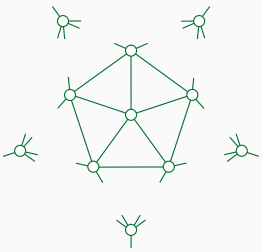
Фиг.2.80(а): започваме с връх от степен 5.



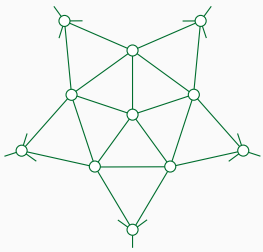
Фиг.2.80(б): слагаме пет съседа от степен 5.



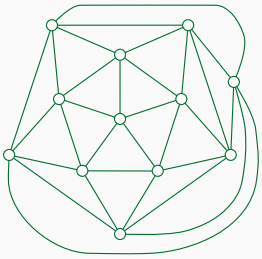
Фиг.2.80(в): свързваме съседите в цикъл.



Фиг.2.80(г): слагаме още пет върха от степен 5.



Фиг.2.80(д): свързваме ги със свободните "валентности" на предните пет върха.



Фиг.2.80(е): свързваме ги в цикъл, после нов връх от степен 5 към последните свободни "валентности".

Графът на Фигура 2.80—който, както лесно се вижда, е изоморфен на графа-икосаедър, който на свой ред съответства на многостена-икосаедър, показан на Фигура 2.78—е контрапример за Хипотеза 1. Той представлява триангулация, която не може да бъде генерирана от Индуктивно Триангулиране. За да се убедим в това, достатъчно е да съобразим, че Индуктивно Триангулиране генерира триангулации (и съответно, графи), които имат връх от степен 2 (базата) или 3 (индуктивната стъпка). А графът на Фигура 2.80 няма такива върхове, тъй като е 5-регулярен.

Теорема 34: $m \leqslant 2n - 4$ при планарните обикновени двуделни графи

За всеки планарен обикновен двуделен граф G с поне две ребра, ако n е броят на върховете и m е броят на ребрата, то

$$m \leqslant 2n - 4$$

Доказателство: Нека G е свързан – ако не е свързан, доказваме неравенството за свърза-

ните компоненти и сумираме по компонентите. И така, G е свързан планарен двуделен граф. Нека лицата са f на брой, наречени t_1, t_2, ..., t_f. Имаме:

$$\begin{aligned}
n - m + f &= 2 && \text{съгласно Теорема 32}\\
n - 2 &= m - f && \text{очевидно}\\
2n - 4 &= 2m - 2f && \text{очевидно}\\
2n - 4 &= m + \frac{2m - 4f}{2} && \text{очевидно}\\
2n - 4 &= m + \frac{\sum_{i=1}^{f} d(t_i) - 4f}{2} && \text{Съгласно Лема 16}
\end{aligned}$$

Забележете, че $\sum_{i=1}^{f} d(t_i) - 4f$ е неотрицателно количество за всеки планарен двуделен граф с поне две ребра, защото всяко лице има ограждащ цикъл с дължина поне 4, тоест $d(t_i) \geqslant 4$ за всяко i. Щом $\sum_{i=1}^{f} d(t_i) - 4f \geqslant 0$, следва, че $2n - 4 \geqslant m$. □

Следствие 13

$K_{3,3}$ не е планарен граф.

Доказателство: Директно от Теорема 34: $K_{3,3}$ има 9 ребра, 6 върха, и $9 \nleqslant 2 \times 6 - 4$. □

Теорема 35: Във всеки планарен граф има връх от степен $\leqslant 5$.

Във всеки планарен граф има връх от степен, не по-голяма от 5.

Доказателство: Да допуснем противното – съществува планарен граф $G = (V, E)$, такъв че $\forall v \in V(d(v) \geqslant 6)$. Тогава:

$$\sum_{v \in V} d(v) \geqslant 6n \quad \leftrightarrow \quad \sum_{v \in V} d(v) = 6n + k \text{ за някое неотрицателно } k$$

От Лема 1 знаем, че $\sum_{v \in V} d(v) = 2m$. Следователно,

$$2m = 6n + k \quad \leftrightarrow \quad m = 3n + \frac{k}{2}$$

Но тъй като $\frac{k}{2}$ е неотрицателно, този извод противоречи на Теорема 33 на стр. 150. □

Допълнение 20: Теоремата за шестте цвята. Теоремата за петте цвята.

Теоремата за четирите цвята (Теорема 7) е изключително сложен теоретичен резултат. В това допълнение ще докажем два по-слаби резултата. Напълно естествено, техните доказателства са много по-прости и лесно разбираеми. Както и може да се очаква, по-слабата от тези теореми, а именно за шестте цвята, е по-лесната от двете.

Теорема 36: Теоремата за шестте цвята

Всеки планарен граф е 6-оцветим.

Доказателство: Ще докажем теоремата с индукция по броя на върховете. За база ще вземем планарните графи с не повече от 6 върха. Те са очевидно 6-оцветими. Да допуснем, че всеки планарен граф с $\leqslant n$ върха е 6-оцветим. Да разгледаме произволен планарен граф $G = (V, E)$ с $n + 1$ върха. Съгласно Теорема 35, $\exists u \in V : d(u) \leqslant 5$. Очевидно $G - u$ също е планарен, освен това $G - u$ има n върха. Съгласно индуктивното предположение, $G - u$ е 6-оцветим. Нека f е произволно оцветяване на $G - u$ в не повече от 6 цвята. По конструкция, $|N(u)| \leqslant 5$. Тогава върховете от $N(u)$ "ползват" не повече от пет цвята и задължително има един цвят, да кажем белият цвят, такъв че никой връх от $N(u)$ не е оцветен в него. Тогава оцветяването f' на G, което дава бял цвят на u, а на всеки друг връх v дава цвета $f(v)$, е легитимно оцветяване с не повече от 6 цвята. □

Теорема 37: Теоремата за петте цвята

Всеки планарен граф е 5-оцветим.

Доказателство: Нека цветовете са жълт, зелен, син, червен и кафяв. С цел по-кратко изложение няма да правим строга разлика между планарен граф, неговите върхове и ребра, от една страна, и планарното му вписване със своите планарни върхове и ребра, от друга страна.
Ще докажем теоремата с индукция по броя на върховете. За база ще вземем планарните графи с не повече от 5 върха. Те са очевидно 5-оцветими. Да допуснем, че всеки планарен граф с $\leqslant n$ върха е 5-оцветим. Да разгледаме произволен планарен граф $G = (V, E)$ с $n + 1$ върха. Съгласно Теорема 35, $\exists u \in V : d(u) \leqslant 5$. Очевидно $G - u$ също е планарен, освен това $G - u$ има n върха. Съгласно индуктивното предположение, $G - u$ е 5-оцветим. По конструкция, $|N(u)| \leqslant 5$.
Дотук доказателството повтаря доказателството на Теорема 36. Но сега се налага да продължим по друг начин. Сега разполагаме само с пет цвята. Ако $N(u) = 5$, оцветяването на $G - u$ с $\leqslant 5$ цвята може да е такова, че върховете от $N(u)$ са оцветени в пет различни цветове и нямаме "свободен" цвят, който да дадем на u. Може да се наложи да "пребоядисаме" върхове в $G - u$ по такъв начин, че хем да "освободим" един цвят за u, хем оцветяването да остане валидно (да няма ребро, чиито краища са в един и същи цвят).

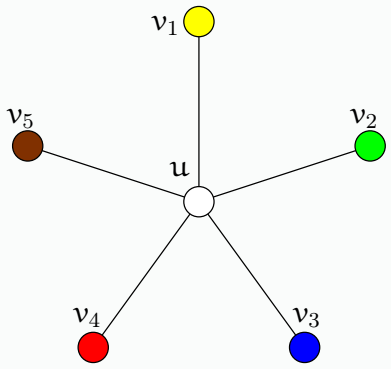
Ако $|N(u)| \leqslant 4$, то "пребоядисване" не се налага; очевидно съществува поне един цвят измежду петте, без ограничение на общността нека това е зеленият цвят, който не се използва от никой връх в $N(u)$. Тогава оцветяваме u в зелено и сме готови.

Нека $|N(u)| = 5$. Ако има поне един цвят измежду петте, който не се използва от никой връх в $N(u)$, доказателството се извършва аналогично.

Нека всичките пет цвята се ползват от петте върха на $N(u)$. Ще "пребоядисваме" върхове в $G - u$, за да "освободим" един цвят за u.
Нека $N(u) = \{v_1, v_2, v_3, v_4, v_5\}$. Без ограничение на общността, нека в планарното вписване на G, тези върхове са наредени по този начин:

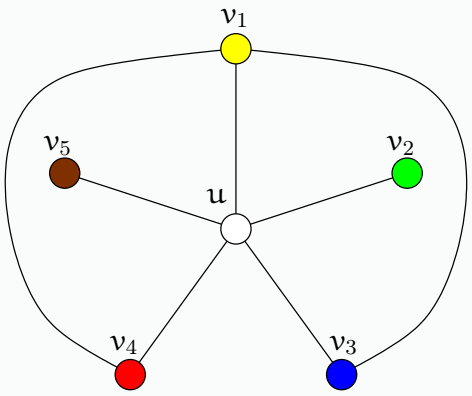
$$(v_1, v_2, v_3, v_4, v_5)$$

ако обикаляме около u в дадена посока, да речем, по часовниковата стрелка. Нека цветовете им са, както е показано на следната рисунка:



Между всеки два от върховете $\{v_1, v_2, v_3, v_4, v_5\}$ може да има или да няма ребро, но тези възможности не са независими. Да разгледаме десетте ненаредени двойки $\{v_1, v_2\}$, $\{v_1, v_3\}$, $\{v_1, v_4\}$, $\{v_1, v_5\}$, $\{v_2, v_3\}$, $\{v_2, v_4\}$, $\{v_2, v_5\}$, $\{v_3, v_4\}$, $\{v_3, v_5\}$, $\{v_4, v_5\}$. Да наречем тези двойки, *потенциалните ребра*. От тях, $\{v_1, v_2\}$, $\{v_2, v_3\}$, $\{v_3, v_4\}$, $\{v_4, v_5\}$, $\{v_1, v_5\}$ са *късите* потенциални ребра, а останалите, *дългите*. Подчертаваме, че това разграничение между къси и дълги потенциални ребра е възможно само след указването на разположението на върховете в равнината; ако разсъждавахме на ниво граф (а не на ниво планарно вписване) без да уточним някаква кръгова наредба между $\{v_1, v_2, v_3, v_4, v_5\}$, такова категоризиране на потенциалните ребра би било безсмислено. Още дефинираме, че кое да е потенциално ребро *присъства* в G, ако в G има ребро с краища съответните два върха.

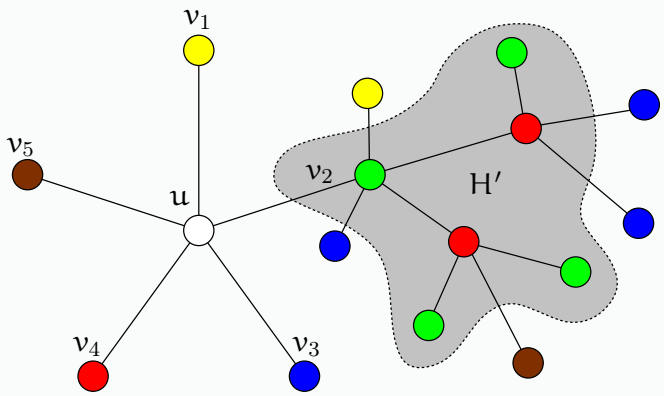
Тривиално е да се покаже, че поне едно от потенциалните ребра не присъства в G. Ако допуснем обратното, в G би имало подграф, изоморфен на K_5. Става дума за индуцирания от $\{v_1, v_2, v_3, v_4, v_5\}$ подграф. Сега ще покажем по-силно твърдение: поне три от дългите потенциални ребра не присъстват в G, независимо от това дали и колко къси потенциални ребра присъстват. Доказателството се основава на наличието на петта ребра (u, v_1), $\ldots$, (u, v_5) в (планарното вписване на) G. Лесно се вижда, че най-много две дълги потенциални ребра може да се разположат в равнината, така че да няма пресичане нито на дълги ребра, нито на дълго ребро с някое от (u, v_i) за $1 \leqslant i \leqslant 5$. Следната рисунка онагледява възможността да има две дълги ребра, инцидентни с v_1:



Следователно, за поне два върха от $\{v_1, v_2, v_3, v_4, v_5\}$, без ограничение на общността нека това са v_2 и v_4, е вярно, че те не са съседи в G. Останалата част на доказателството използва допускането, че между v_2 и v_4 няма ребро. Подчертаваме, че никъде надолу *няма да използваме допускане*, че има ребра (v_1, v_3) и (v_1, v_4). Последната рисунка имаше за цел да покаже, че не може да има повече от две дълги ребра. Дълги ребра може изобщо да няма.

Цветовете на v_2 и v_4 са зелено и червено. Нека H е подграфът на G, индуциран от зелените и червените върхове.

Случай I: v_2 и v_4 се намират в различни свързани компоненти на H. Нека H' е свързаната компонента на H, която съдържа v_2. Например:



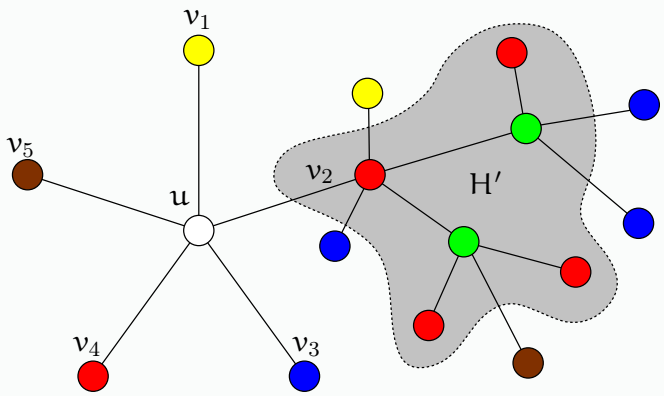
Правим следната смяна на цветове на върхове:

- всеки връх на H', който е бил зелен преди смяната на цветове, става червен,
- всеки връх на H', който е бил червен преди смяната на цветове, става зелен,
- цветът на всеки връх в G, който не е в H', не се променя.

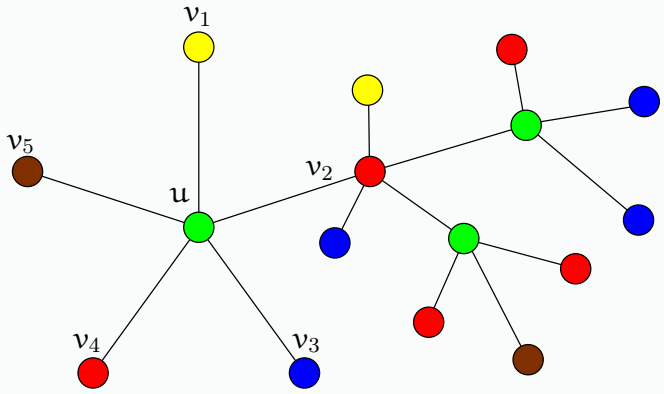
Ще докажем, че тази смяна на цветове представлява легално оцветяване на върховете на графа. Тоест, че не може да се получи ребро, двата края на което са оцветени в един цвят.

1. ребрата, нито един връх на които не е от H', очевидно не могат да имат два края от един цвят след смяната;
2. да разгледаме произволно ребро (x, y), точно единият край на който, да речем x, е в H'. Преди смяната на цветовете, x е бил червен или зелен, y е бил жълт, син или кафяв. След смяната, x е пак в един от цветовете червен или зелен (обратния на предния цвят), а y си остава в цвят, който не е нито червен, нито зелен. Следователно, двата края на (x, y) не са оцветени в един цвят след смяната.
3. да разгледаме произволно ребро (x, y) от H'. Без ограничение на общността нека x е бил червен, а y, зелен преди смяната. Тогава след смяната x е зелен, а y, червен. Следователно, двата края на (x, y) не са оцветени в един цвят след смяната.

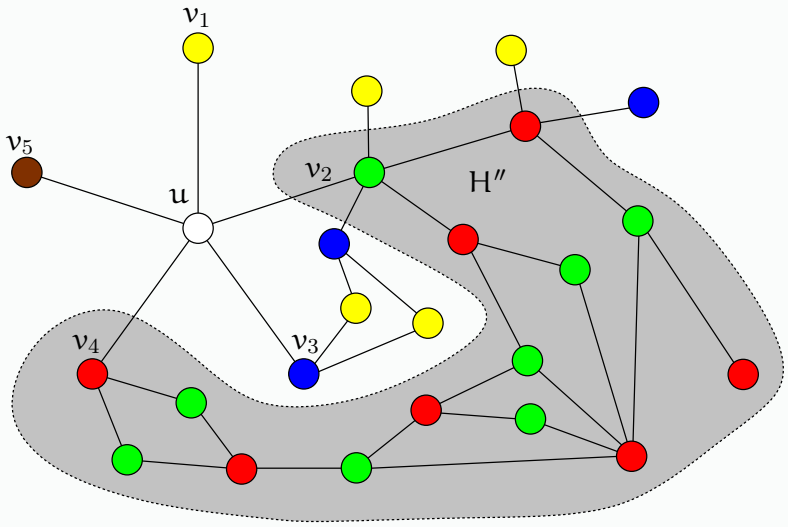
Доказахме твърдението. Ето как изглежда горният пример след смяната на цветовете на H':



След смяната на цветовете, цветът, в който е бил оцветен v_2 преди смяната, а именно зеления, е "свободен" в смисъл, че нито един съсед на u не го ползва. Оцветяваме u в него и получаваме 5-оцветяване на G:

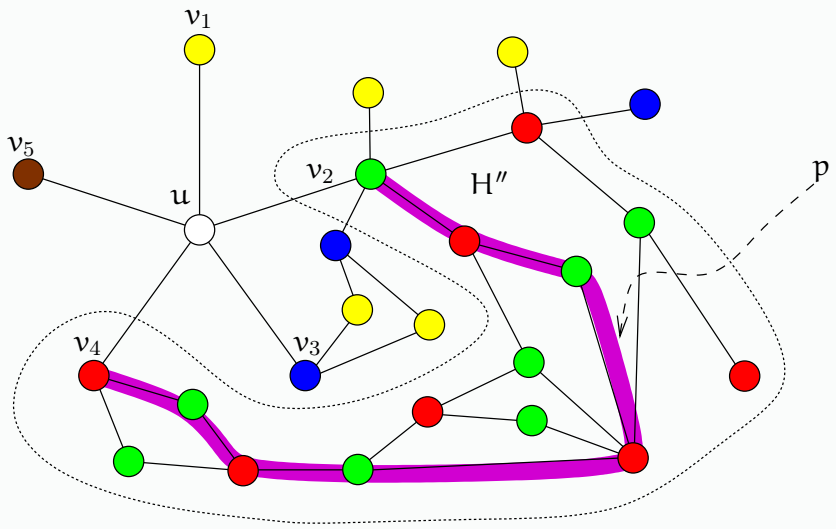


Случай II: v_2 и v_4 се намират в една и съща свързана компонента на H. Нека тази свързаната компонента на H бъде наречена H''. Например:

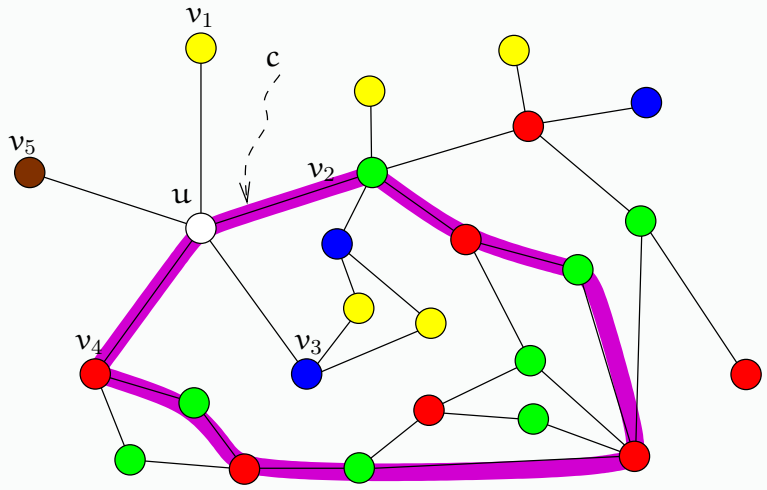


В този случай е безсмислено да сменяме оцветяването на върховете от H'' като в **Случай I**—червено в зелено и обратно—защото няма да "освободим" нито червения, нито зеления цвят за u; u би имал и червен, и зелен съсед след смяната. Но забелязваме, че v_1 и v_3 в този случай са, в някакъв смисъл, изолирани един от друг. Формално, в G не

съществува алтернииращ жълто-син път с краища v_1 и v_3. Сега ще докажем този факт. Тъй като H'' е свързан, съществува алтернииращ зелено-червен прост път между v_1 и v_3 в H''. На следната рисунка е показан такъв примерен път p, очертан в пурпурно:

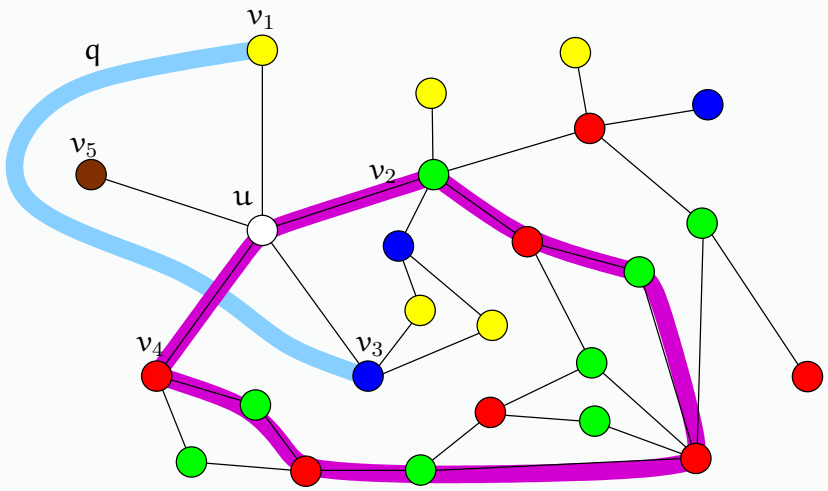


Заедно с ребрата (u, v_4) и (u, v_2) и връх u, пътят p образува прост цикъл c в G:

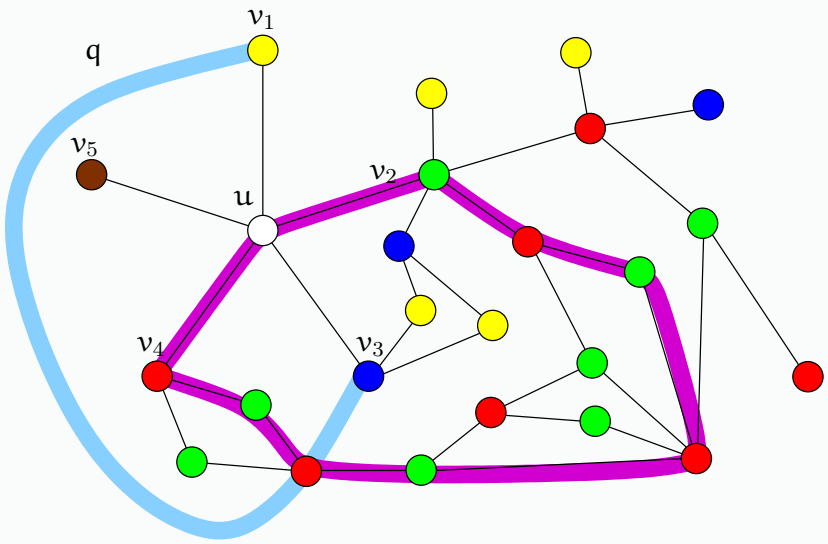


Спрямо цикъла c има точно два района на равнината и очевидно всеки върховете v_1 и v_3 се намира в един от тези райони. Следователно:

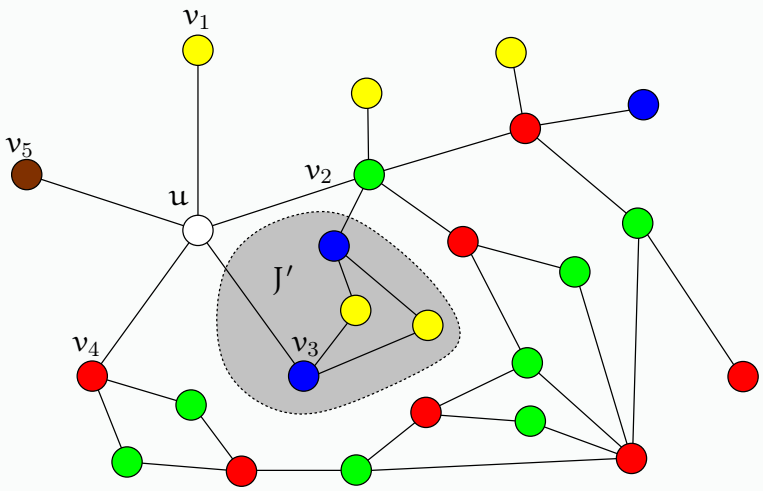
- нито може да има ребро с краища v_1 и v_3 в G, защото такова ребро би пресичало ребро от c, а по конструкция G е планарен;
- нито може да има алтернииращ жълто-син път с краища v_1 и v_3. Ако допуснем, че има такъв път q и той няма общ връх с c, би имало непозволено пресичане на ребра и G не би бил планарен (q е очертан в светло синьо):



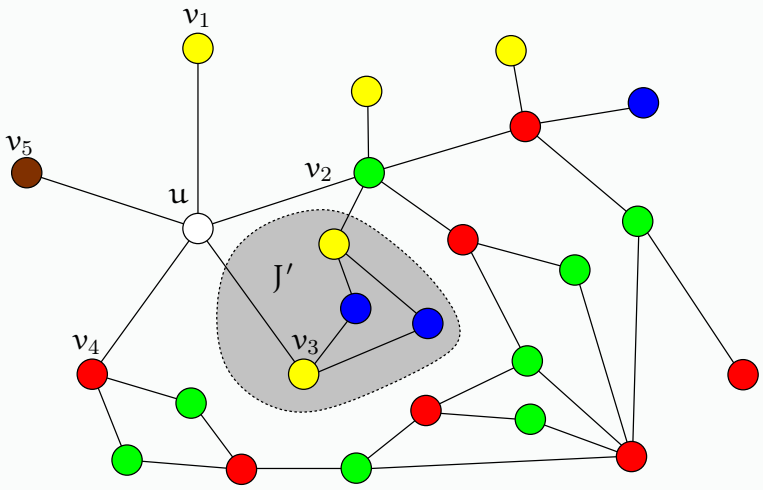
Да допуснем, че хипотетичният път q има поне един общ връх с c. Върховете на q са оцветени в жълто и синьо. Върховете на c са оцветени в три цвята: зелен, червен и бял (това е цветът на u в момента, u още не е оцветен в нито един от петте цвята, така че можем да считаме, че е оцветен в някакъв шести временен цвят, например бял). Получаваме противоречие – общият връх трябва да е от една страна син или жълт, а от друга, зелен, червен или бял.



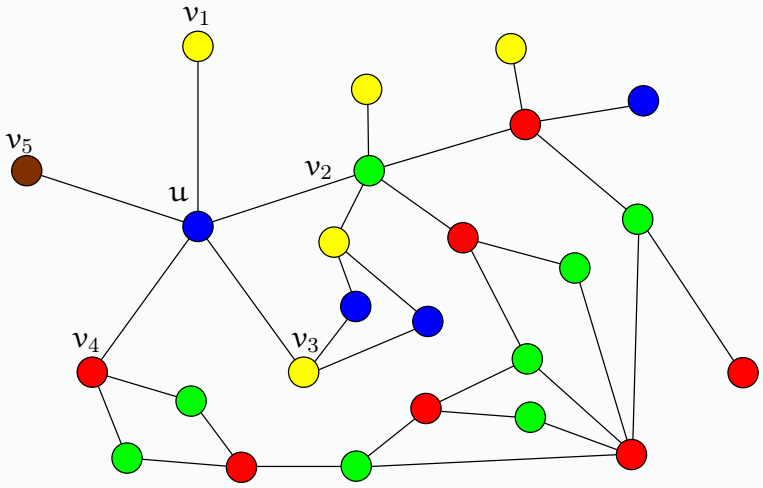
Доказахме, че при текущите допускания не съществува алтернираща жълто-синя път между v_1 и v_3. Нека J е подграфът на G, индуциран от жълтите и сините върхове. Щом в G няма алтернираща жълто-синя път между v_1 и v_3, то v_1 и v_3 са в различни свързани компоненти на J. Нека v_3 се намира в свързаната компонента J′ на J:



Извършваме смяна на цветовете жълто в синьо и синьо в жълто върху J', оставяйки другите върхове на G в същите цветове като преди.



По начин напълно аналогичен на **Случай I** доказваме, че след тази смяна на цветовете не може да получим ребро, чиито два края са оцветени в един и същи цвят. Но тогава връх v_3 се оказва жълт. По този начин си "освобождаваме" синия цвят и оцветяваме u в синьо.

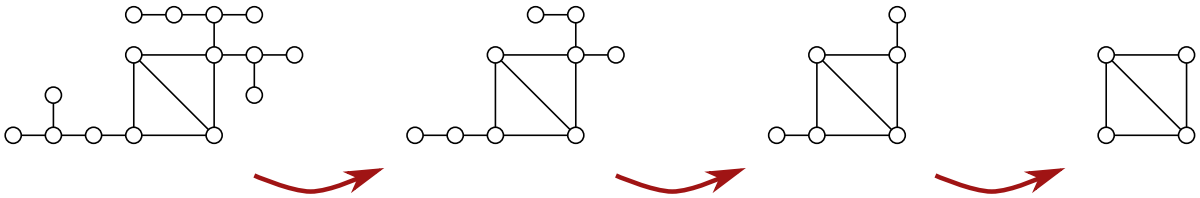


□

2.13.4 Хомеоморфизъм на графи.

Разглеждаме свързани мултиграфи. Нека G е свързан мултиграф. Ще започнем с наблюдението, че по отношение на планарността, висящите върхове нямат значение. Ако има висящи върхове, можем спокойно да ги игнорираме. По-прецизно казано, ако има висящи върхове в G, ние ги трием, докато не се получи граф без висящи върхове. G е планарен тогава и само тогава, когато полученият граф е планарен.

Изтриване на висящите върхове може да се наложи да става става итеративно, защото след "първия пас" на изтриване на висящи върхове може са стане така, че други върхове, които преди не са били висящи, да станат от висящи, и така нататък. Тази ситуация възниква, ако G е или дърво, или не е дърво, но има "израстъци-дървета". Следната фигура илюстрира втората възможност: графът G е $K_4 - e$ с три "израстъка-дървета"; трием висящите върхове на три итерации, въпросните "израстъци" изчезват и остава само $K_4 - e$.



Не е трудно да се съобрази, че ако G е дърво, то итеративното изтриване на висящите върхове ще редуцира това дърво до един единствен връх.

Следващото наблюдение е, че върховете от степен две също нямат значение за планарността, но в различен смисъл. Тях няма да ги трием, понеже просто изтриване на върхове от степен две може да направи планарен граф от непланарен граф, а ще ги елиминираме по друг начин. Без ограничение на общността, нека G е нетривиален и няма висящи върхове. Ако в G няма върхове от степен, по-голяма от две, то всички върхове са точно от степен две и G е граф-цикъл, а всеки граф-цикъл задължително е планарен и в този смисъл, не е интересен от гледна точка на задачата дали даден граф е планарен или не.

Да допуснем, G че има поне един връх от степен, по-голяма от две, както и поне един връх от степен две. Да разгледаме всички максимални по включване пътища $p_1, \ldots, p_t$ в G, чиито вътрешни върхове са от степен две (краищата им са върхове от степен, по-голяма от две). Нека p_i има краища u_i и v_i, за $1 \leqslant i \leqslant t$; очевидно върховете $u_1, \ldots, u_t, v_1, \ldots, v_t$ не са непременно два по два различни. Ако $u_i = v_i$ за някое i, то очевидно p_i е цикъл, в който всички върхове са от степен две с изключение на $u_i = v_i$, който е от степен, по-голяма от две. За всяко $i \in \{1, \ldots, t\}$, изтриваме вътрешните върхове на p_i (които са от степен две) и слагаме едно **ново** ребро e'_i, което има краища u_i и v_i. Иначе казано, заменяме "вътрешността" на p_i с едно ребро.[†]

Тези идеи са илюстрирани на Фигура 2.81. Различните пътища с вътрешни върхове от степен две и крайни върхове от степен, по-голяма от две, са оцветени в различни цветове за по-лесното проследяване на резултата от заменянето им с ребра. Графът-резултат G' е мултиграф с примка. Лесно се вижда, че ако превърнем G' в обикновен граф, заменяйки сноповете от паралелни ребра с по едно ребро и изтривайки примките, и после продължим

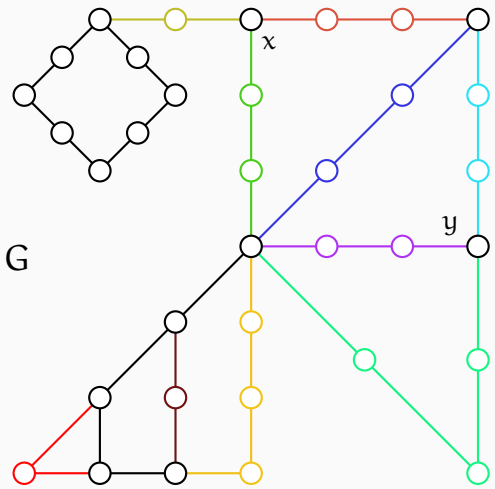
[†]При тези операции може да стане така, че започвайки от обикновен граф да получим мултиграф. Ето защо. Паралелни ребра може да се получат по два начина. Първо, ако преди замяната на p_i с ново ребро e' е имало ребро e с краища u_i и v_i, то след замяната e и e' ще се окажат паралелни ребра. Второ, ако два различни пътя p_i и p_j преди замяната им с (различни) ребра са имали едни и същи краища, то след замяната им с ребрата e' и e'', те (ребрата e' и e'') ще се окажат паралелни ребра. А примка ще се получи от замяната на всеки p_i, който е (прост) цикъл, с ребро.

да прилагаме изложените правила за елиминиране на върхове от степени едно и две и паралелните ребра, докато можем, ще колабираме целия граф до един единствен връх[†]. Но това не е целта на изложението тук. Целта е да илюстрираме факта, че върховете от степен две нямат значение за планарността – което вече направихме убедително.

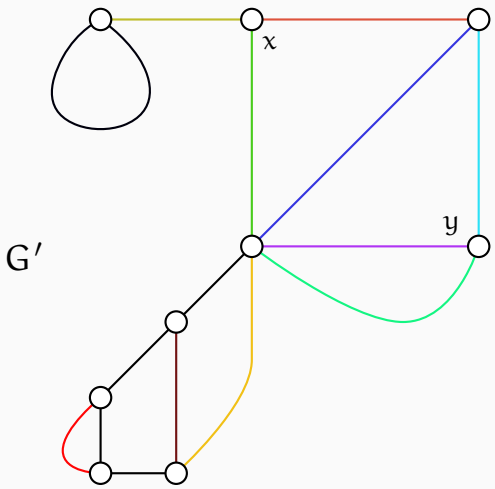
Наблюдение 29

Върховете от степен две нямат значение за планарността на графите в следния смисъл: ако всички максимални по включване, не непременно прости, пътища с вътрешни върхове от степен две бъдат заменени всеки с по едно ребро, полученият граф (който може да е мултиграф с примки; това е без значение) ще е планарен тогава и само тогава, когато оригиналният граф е бил планарен.
Планарността на даден граф се определя единствено от върховете от степен поне три и това, кой от тях с кой друг от тях е свързан. Дали тези свързвания стават с ребра или с пътища, чиито вътрешни върхове са от степен две, няма значение.

Фигура 2.81 : Върховете от степен две нямат значение за планарността.



Оригиналният граф G с върхове от степен две.



Полученият граф G′ без върхове от степен две.

[†]Не всеки граф може да бъде колабиран до един единствен връх по този начин. Това е възможно тогава и само тогава, когато графът има параметър *treewidth*, по-малък от 3. Даденият G има treewidth 2 и затова може да бъде сведен до един връх по този начин. За да увеличим параметъра treewidth на G на 3, достатъчно е да сложим едно ребро между x и y. Повече информация за параметъра treewidth на графи има в [18].

Определение 69: Подразделяне на ребра и серийно редуциране на ребра

Нека $G = (V, E)$ е произволен мултиграф и $e = (v, w)$ е произволно ребро в G. *Подразделянето на* e[а] е трансформирането на G в

$$G' = \big(V \cup \{u\}, (E \backslash \{e\}) \cup \{(v, u), (u, w)\}\big)$$

където u е връх, такъв че $u \notin V$. Ако e е примка (което означава, че $v = w$), то (v, u) и (u, w) са две паралелни ребра.
Обратната операция на подразделянето е *серийното редуциране на ребра*[б]. Нека x е произволен връх от в G, инцидентен с точно две различни ребра $e_1 = (y, x)$ и $e_2 = (x, z)$, нито едно от които не е примка. Серийното редуциране на e_1 и e_2 е трансформирането на G в

$$G'' = \big(V \backslash \{x\}, (E \backslash (\{e_1\} \cup \{e_2\})) \cup \{e_3\}\big)$$

където e_3 е ребро с краища y и z, такова че $e_3 \notin E$.

[а] На английски често се ползва терминът е *egde subdivision*, но той не е универсален. Gibbons [27, стр. 76] казва "insertion of a vertex of degree 2". Diestel [18, стр. 20] казва "replace ... edges with independent paths". В [58, стр. 491] се казва "edge subdivision".

[б] На английски се използа терминът *series reduction*, но и той не е универсален. Gibbons [27, стр. 76] казва "the reverse of this process", имайки предвид подразделянето на ребра. Diestel [18] успява да изгради изложението си без да ползва това понятие. В [58, стр. 491] се казва "series reduction".

Забележете, че новопоявилото се ребро e_3 в серийното редуциране може да е примка. Никоя примка обаче не може да изчезне в резултат на серийно редуциране.

Определение 70: Хомеоморфизъм на графи.

Два графа G и H са хомеоморфни, ако съществува граф J, такъв че J се получава от G чрез крайна серия (може и празна) от подразделяния и серийни редуцирания на ребра и J е изоморфен на H.

Забелязваме, че релацията "са хомеоморфни" е симетрична, така че G и H са "равностойни участници" в Определение 70. Освен това, релацията е рефлексивна и транзитивна, така че тя е релация на еквивалентност. Като пример за хомеоморфизъм може да послужат графите G и G' на Фигура 2.81. Те са хомеоморфни. Нека се убедим, че е така. Графът G' се получава от G чрез серийни редуцирания на ребра и той е изоморфен на себе си.

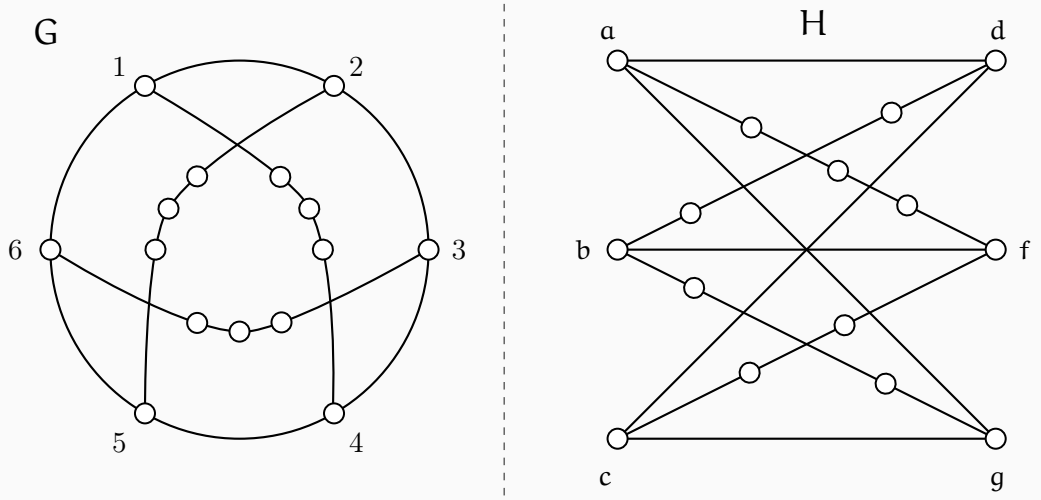
Изоморфизмът присъства в Определение 70 по една причина: графите G и H, бидейки произволни графи, може да имат нямат общи имена на върхове; дори да имат общи имена на върхове, тези общи имена може да са върху "несъответстващи" си върхове, в смисъла на изоморфизма. Ако ограничим вниманието си върху Фигура 2.81, няма нужда да мислим за изоморфизъм, понеже G' се получава от G, по отношение на върховете, чрез и само чрез изтриване на върхове, така че останалите върхове, а именно тези в G', точно съответстват на върхове от G в смисъла на изоморфизма.

Лесно се вижда, че можехме да въведем "хомеоморфизъм на графи" само чрез едната операция от Определение 70, а именно подразделянето. Можехме да кажем, че G и H са хомеоморфни тогава и само тогава, когато съществува G', който се получава от G чрез подразделяния на ребра и съществува H', който се получава от H чрез подразделяния на ребра, такива че G' и H' са изоморфни. Причината да въвеждаме и обратната на подразделянето операция на серийно редуциране на ребра е по-лесната мислена визуализация: ако човек се

опитва да види дали два графа са хомеоморфни, съвсем естестествено е мислено да извършва както подразделяния, така и серийните редуцирания на ребра.

Друг пример за хомеоморфни графи е показан на Фигура 2.82. Само "съществените" върхове, тези от степен, по-голяма от две, са именувани. Проверете сами, че единият граф се получава от другия чрез серия от подразделяния и серийни редуцирания на ребра, съгласно Определение 70; а също така, че има изоморфни графи G' и H', които се получават съответно от G и H само с подразделяния на ребра, и че има изоморфни графи G'' и H'', които се получават съответно от G и H само със серийно редуциране на ребра.

Фигура 2.82 : G и H са хомеоморфни графи.



Наблюдение 30

Със серийни редукции на ребра не може да намалим броя на паралелните ребра в мултиграф; няма как с такива редукции да сведем сноп от $k \geqslant 2$ ребра паралелни ребра до едно единствено ребро, освен в случая $k = 2$, в който поне единият от двата върха е не-инцидентен с други ребра (но тогава серийната редукция дава примка). Също така със серийни редукции на ребра не можем да изтрием примка. Следователно, със серийни редукции на ребра не може да направим обикновен граф от "истински" мултиграф.

Допълнение 21: Още за хомеоморфизма

Етимологията на "хомеоморфизъм" е следната: на гръцки ὅμοιος означава "подобен", а μορφή означава "форма". Следното определение е взето от топологията. То е значително по-старо от Определение 70.

Определение 71: Хомеоморфизъм между пространства

Две топологически пространства са хомеоморфни, ако между тях има биекция, такава че и тя, и обратната ѝ функция–биекция са непрекъснати.

Неформално и не съвсем точно казано, едното пространство може да се получи от

другото без разкъсвания и по чисто "еластичен" начин. Поначало "хомеоморфизъм" е възникнало именно в топологията и е пренесено в теорията на графите по-късно. Пренасянето му в теорията на графите става много естествено, ако на графите се гледа не теоретико-множествено, а топологически, като на вид топологически пространства. Подробно въведение в историята на понятието "хомеоморфизъм" и използването му има в [44].

Трябва да не бъркаме "хомеоморфизъм" с "хомоморфизъм"! Това са съвсем различни понятия, въпреки че звучат много подобно.

Възможен е и друг подход към дефинирането на графи, които в се различават само по върховете от степен 2, например, в чудесната книга "Graph Theory" на Diestel [18]. Нека "подразделяне на ребра" е дефинирано като в Определение 69. За даден граф G, *подразделяне на* G е всеки граф, който се получава от G чрез подразделяния на ребра.

Определение 72: Топологичен минор, [18, стр. 20]

За всеки два графа G_1 и G_2 казваме, че G_1 *топологичен минор на* G_2, ако G_2 съдържа подграф G', такъв че G_1 е изоморфен на подразделяне на G'.

Diestel използва "хомеоморфизъм" само в чисто топологическия смисъл, никога за графи. Неговото изложението на теоремата на Kuratowski се базира на понятията "топологичен минор", което понятие ще въведем по-долу, и "минор" от Допълнение 22. Първо да видим разликата между подхода на Diestel и подхода, възприет в тези лекционни записки, който е взет от Gibbons [27].

Подходът на Gibbons [27] е, първо да се дефинира хомеоморфизъм и после в централния резултат—теоремата на Kuratowski на страница 77 (Теорема 44 в тези записки)—да каже "подграф, хомеоморфен на". При Diestel [18] аналогичният резултат е Лема 4.4.3 на страница 97. Там се говори за топологични минори[a]. Сега ще видим, че "граф G_1 е хомеоморфен на подграф на граф G_2" и "G_1 е топологичен минор на G_2" **не казват едно и също нещо**, въпреки че сходство има. Релацията на хомеоморфизъм е симетрична, докато релацията "е топологичен минор на" е антисиметрична, ако гледаме на графите като на неименувани графи (понятие, въведено в Подсекция 2.8.2). Ако за два неименувани графа е вярно, че първият е топологичен минор на втория и вторият е топологичен минор на първия, то това е един и същи (неименуван) граф. Ако обаче неименуваните графи са различни, то или нито един не е топологичен минор на другия, или точно единият от двата е топологичен минор на другия. Следователно става дума за частична наредба, защото релацията "е топологичен минор на", освен антисиметрична, е рефлексивна и транзитивна.
Ако G_1 е топологичен минор на G_2, то G_1 е "по-малък" от G_2 в някакъв смисъл. По-прецизно казано, G_1 предхожда G_2 във въпросната частична наредба. Хомеоморфизмът, от друга страна, дефинира релация на еквивалентност. Между два различни графа, които са хомеоморфни, не е задължително единият да предхожда другия в никакъв смисъл. Можем да свържем понятията "хомеоморфизъм" и "топологичен минор" така: графите G_1 и G_2 са хомеоморфни тогава и само тогава, когато съществува трети граф G_3, такъв че и G_1, и G_2 са топологични минори на G_3, тоест, и G_1, и G_2 са "по-малки" от G_3. Да разгледаме G и H от Фигура 2.82. Както вече отбелязахме, те са хомеоморфни. От друга страна, нито единият от тях не е топологичен минор на другия. Нека се убедим в това. Очевидно и G, и H са хомеоморфни на $K_{3,3}$ – и двата се получават

от $K_{3,3}$ чрез подразделяне на ребра. Обаче H има четири пътя, които имат вътрешни върхове от степен две, а G има само три такива пътя, следователно H не е "по-малък" от G и няма как H да е топологичен минор на G. От друга страна, измежду пътищата в H, имащи вътрешни върхове от степен две, само един има три вътрешни върха, а останалите имат по два вътрешни върха; а и трите пътя в G, имащи вътрешни върхове от степен две, имат по три вътрешни върха. Следователно, няма как G да е топологичен минор на H. По отношение на частичната наредба "е топологичен минор на", нито един от двата не е по-малък от другия; иначе казано, H и G са несравними.
В резюме: ако G_1 е хомеоморфен на подграф на G_2, то не е задължително G_1 да предхожда G_2, докато, ако G_1 е топологичен минор на G_2, то G_1 задължително предхожда G_2.

[a]Всъщност, Diestel говори за минори, не непременно топологични, но преди това в Лема 4.4.2 показва, че по отношение на K_5 или $K_{3,3}$ е все едно дали ще се ползва "минор" или "топологичен минор".

Допълнение 22: Минори в графи

Разглеждаме прости графи, тъй като паралелните ребра и примките са без значение по отношение на минорите.

Определение 73: Контракция на ребро

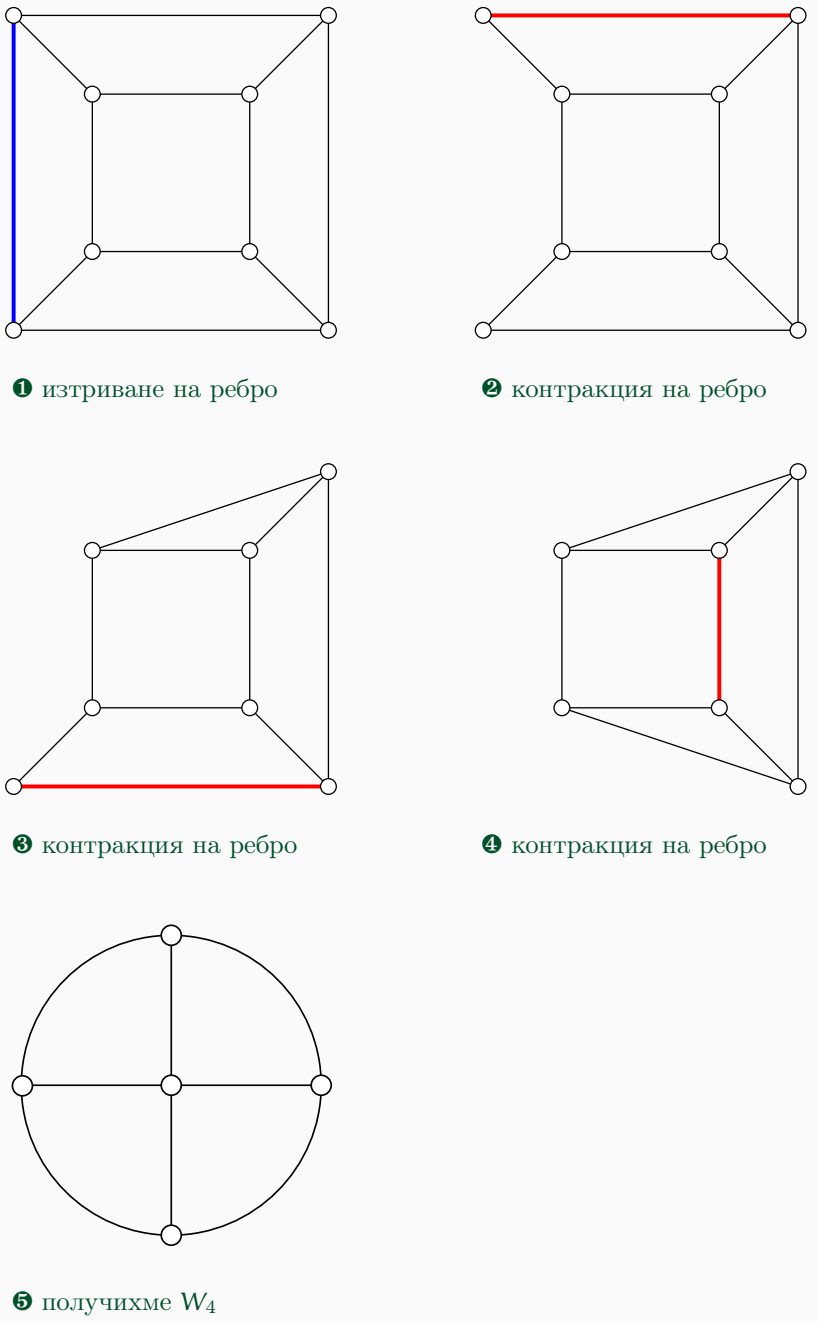
Нека са даден граф $G = (V, E)$ и нека $e \in E$. Нека $e = (u, v)$. Нека $V_1 = N(u) \setminus \{v\}$ и $V_2 = N(v) \setminus \{u\}$. Нека z е нов връх. Нека $G' = G - u - v$. *Контракция на реброто* e е операция, която трансформира G в графа $(V(G') \cup \{z\}, E(G') \cup E_z)$, където $E_z = \{(z, x) \mid x \in V_1 \cup V_2\}$.

Неформално казано, контракцията на ребро е идентифицирането на двата му края, при което реброто се превръща в примка и после изчезва, а ако се появят снопове от (по две) паралелни ребра, всеки сноп се заменя с едно ребро. Крайният резултат е обикновен граф. Фигура 2.83 илюстрира контракции на ребра.

Определение 74: Минор на граф

Нека са дадени два графа H и G. Казваме, че H е *минор* на G, ако граф, изоморфен на H, се получава от G чрез серия от изтривания на върхове, изтривания на ребра и контракции на ребра.

Фигура 2.83 : W_4 е минор на Q_3.



Фигура 2.83 показва, че графът W_4[а] е минор на тримерния хиперкуб Q_3[б].

Лесно се вижда, че "подграф, изоморфен на" и "топологичен минор" (Определение 72) са частни случаи на "минор".

- Ако H е изоморфен на подграф на G, то този подграф може да бъде получен със серия от изтривания на върхове и ребра. Следователно, H е минор на G.
- Ако H е топологичен минор на G, то в G съществува подграф G', такъв че H се явява подразделение на G. Можем да получим G' от G само с изтривания

> на върхове и ребра. За да получим H от G', използваме третата операция от Определение 74, а именно контракцията на ребра. С контракции на ребра може да редуцираме всеки път, чиито вътрешни върхове са от степен 2, до едно ребро. Следователно H е топологичен минор на G.

Конверсното обаче не е вярно. Ако H е минор на G, то е възможно H да не е изоморфен на нито един подграф на G и дори да не е хомеоморфен на нито един подграф на G. За да се убедим в това, да разгледаме пак Фигура 2.83. W_4 нито е подграф на Q_3, нито е хомеоморфен на подграф на Q_3, защото $\Delta(W_4) = 4$, а $\Delta(\mathsf{Q}_3) = 3$. И така, понятието "минор" е по-общо от "топологичен минор", което понятие на свой ред е по-общо от "подграф, изоморфен на". Заключаваме, че в Определение 74 няма излишък – и трите изредени операции са съществени за понятието, което се дефинира.

Следното необходимо и достатъчно условие за планарност на графи е доказано през 1937 г. от Klaus Wagner [59]. Тази теорема прилича твърде много на теоремата на Kuratowski на стр. 182, само че е формулирана чрез минори, а не чрез хомеоморфни подграфи.

Теорема 38: Теорема на Wagner

Граф е планарен тогава и само тогава, когато не съдържа нито K_5, нито $\mathsf{K}_{3,3}$ като минори.

Този резултат е интересен сам по себе си, но не е епохален, имайки предвид, че се появява седем години след теоремата на Kuratowski, казваща нещо доста сходно. Важността му е друга. Wagner, гледайки на задачата за планарността на графи от гледната точка на минорите, е формулирал една много по-обща хипотеза, доказването на която е отнело десетилетия и е довело до най-значителните и дълбоки резултати в съвременната теория на графите.

Хипотеза 2: Хипотеза на Wagner [60]

Във всяко безкрайно множество от неименувани графи съществуват два графа, единият от които е минор на другия.

Според Diestel [18, стр. 373]:

> Wagner did indeed discuss this problem in the 1960s with his then students, Halin and Mader, and it is not unthinkable that one of them conjectured a positive solution. Wagner himself always insisted that he did not—even after the graph minor theorem had been proved.

Тази хипотеза вече е доказана от Robertson и Seymour, в резултат на което е станала теорема, известна като The Graph Minor Theorem (теорема за минорите).

Теорема 39: Теоремата за минорите (The Graph Minor Theorem)

Във всяко безкрайно множество от неименувани графи съществуват два графа, единият от които е минор на другия.

Доказателството се простира върху няколкостотин страници в двадесет (20) статии, започвайки с [52] и завършвайки с [53]. Резюме на най-важното от този епохален труд има както в [18], така и в свободно достъпната статия на Lovász [41].

Ще дадем еквивалентна формулировка на Теорема 39, но първо ще дадем едно определение.

Определение 75: Квазинаредба

Бинарна релация е *квазинаредба*[а], ако е рефлексивна и транзитивна.
Квазинаредба $\preccurlyeq \subseteq A \times A$ е *добра квазинаредба*[б], ако $\preccurlyeq$ няма нито безкрайни антивериги[в], нито безкрайни *спускащи се надолу* вериги. Спускаща се надолу верига[г] е множество $\{a_1, a_2, a_3, \ldots\} \subseteq A$, такова че $\ldots \preccurlyeq a_3 \preccurlyeq a_2 \preccurlyeq a_1$.

[а] На английски терминът е *quasi-order*. *Preorder* е синоним.
[б] На английски е *well-quasi-order*, или WQO накратко.
[в] Антиверига е подмножество на A, такова че $\forall a, b \in A : a \neq b \rightarrow a \not\preccurlyeq b \wedge b \not\preccurlyeq a$.
[г] На английски терминът е *descending chain*.

Теорема 40: Теоремата за минорите, алтернативна формулировка [18, глава 12.2]

Крайните графи са добре квазинаредени от минорната наредба $\preceq$.

Еквивалентността на двете формулировки (Теорема 39 и Теорема 40) се вижда лесно.

- Това, че няма безкрайни спускащи се надолу антивериги е очевидно, имайки предвид, че графите са крайни, а всяка от операциите изтриване на връх, изтриване на ребро и контракция на ребро прави графа по-малък.
- Това, че няма безкрайни антивериги е точно каквото казва Теорема 39.

Нотацията "$\preceq$" се използва за означаване на минори: $H \preceq G$ означава, че H е минор на G (или е изоморфен на минор на G, ако говорим за именувани графи).
Пример за минорно затворено множество е множеството от всички гори: всеки минор на ацикличен граф очевидно е ацикличен, защото операциите от Определение 74 не могат да образуват цикли, така че всеки минор на гора също е гора. Друг такъв пример е множеството от планарните графи, защото всеки минор на планарен граф очевидно също е планарен. Трети пример е множеството от графите, които имат върхово покриване не повече от k; очевидно вземането на минор не може да увеличи числото на покриването. Контрапример за минорно затворено множество е множеството от свързаните графи, защото изтриването на ребро или връх може да наруши свързаността; в частност дърветата не са минорно затворено множество. Друг контрапример са 2-оцветимите графи – ако е даден четен цикъл (както знаем, той е 2-оцветим) и направим контракция на едно ребро, ще получим цикъл с дължина с единица по-малка, тоест нечетен цикъл, за който знаем, че не е 2-оцветим. Трети контрапример е множеството от графите, които имат доминиращо число не повече от k; ако в G има връх от степен $n-1$, то очевидно $\gamma(G) = 1$, но изтриването на този връх може да увеличи числото на доминирането.

Определение 76: Идеал и обструкции

Нека е дадена квазинаредба $\leqslant \subseteq A \times A$. *Идеал по отношение на* $\leqslant$, или просто *идеал*, е всяко $\mathcal{I} \subseteq A$, което е затворено надолу по отношение на $\leqslant$. Иначе казано, $\mathcal{I} \subseteq A$ е идеал, ако

$$\forall a, b \in A : (a \in \mathcal{I} \wedge b \leqslant a \rightarrow b \in \mathcal{I})$$

Нека $\mathcal{I} \subseteq A$ е идеал. *Множество от обструкции за* $\mathcal{I}$ е всяко $\mathcal{O} \subseteq A$, такова че $\forall a \in A : a \in \mathcal{I} \leftrightarrow (\forall b \in \mathcal{O} : b \nleqslant a)$.

Нека квазинаредбата, която разглеждаме, е минорната наредба върху неименуваните графи. Ето примери за идеали по отношения на нея.

- Множеството от всички гори е идеал, защото, ако G е ацикличен и $H \preceq G$, то и H е ацикличен.
- Множеството от планарните графи е идеал, защото всеки минор на планарен граф очевидно също е планарен.
- Множеството от графите, които имат върхово покриване не повече от k, е идеал, защото ако $\tau(G) \leqslant k$ и $H \preceq G$, то очевидно $\tau(H) \leqslant k$.

Ето контрапримери за идеали по отношение на нея.

- Множеството от свързаните графи не е идеал, защото изтриването на ребро или връх може да наруши свързаността. В частност, дърветата не са идеал.
- Множеството от 2-оцветимите графи не са идеал – ако е даден четен цикъл (както знаем, той е 2-оцветим) и направим контракция на едно ребро, ще получим нечетен цикъл, за който знаем, че не е 2-оцветим.
- Множеството от графите, които имат доминиращо число не повече от k, не е идеал – ако в G има връх от степен $n - 1$, то очевидно $\gamma(G) = 1$, но изтриването на този връх може да увеличи числото на доминирането.

Ето примери за множества от обструкции.

- Нека идеалът е множеството от ацикличните графи. Множество от обструкции е, например, множеството $\{C_3\}$, където C_3 означава 3-цикъла.
- Нека идеалът е множеството от планарните графи. Множество от обструкции е, например, $\{K_5, K_{3,3}\}$.

Следната теорема е взета от [21]. Авторите Downey и Fellows казват, че произходът ѝ не може да бъде установен и че тя е математически фолклор.

Теорема 41: WQO principle

Нека $\leqslant \subseteq A \times A$ е произволна добра квазинаредба. За всеки идеал $\mathcal{I}$ по отношение на $\leqslant$ съществува **крайно** множество от обструкции.

Следствие 14

Ако за всеки два елемента $x, y \in A$ можем да тестваме в полиномиално време дали $x \leqslant y$, то съществува с полиномиален алгоритъм за тестване дали произволен $z \in A$ принадлежи на идеала $\mathcal{I}$, или не.

Доказателство: Да допуснем, че за всяка обструкция $o \in \mathcal{O}$ бихме могли в полиномиално време да решим дали $o \leqslant z$. Но от Теорема 41 знаем, че обструкциите са само краен брой. Очевидно е, че въпросът дали $z \in \mathcal{I}$ има утвърдителен отговор тогава и само тогава, когато за всяка обструкция o не е вярно, че $o \leqslant z$. И така, тестването за принадлежност към идеала се свежда до тестване за краен брой обструкции. □

Сега вече разглеждаме не коя да е квазинаредба, а именно минорната квазинаредба. Теорема 42 е изключително труден за доказване резултат на Robertson и Seymour от [51]. Забележете, че теоремата **не твърди**, че можем да тестваме в кубично време дали за два произволни графа, единият е минор на другият! Задачата дали за два произволни графа, единият е минор на другия, съдържа като подзадача NP-пълната задачата дали за два произволни графа, единият е подграф на другия; ако подзадачата е NP-пълна, то самата задача е NP-трудна, а както знаем, засега не е известна NP-трудна задача, решима в полиномиално време. Това, което **се твърди**, е, че за предварително фиксиран граф H съществува кубичен алгоритъм, с който можем да тестваме дали $H \preceq G$, за всеки G.

Теорема 42: Тестване за минори в кубично време [51]

За всеки **фиксиран** граф H, за всеки граф G, тестването дали $H \preceq G$ може да се извърши във $O(|V(G)|^3)$[a].

[a] Нотацията $O(x)$ означава всяка функция, която е ограничена от горе от cx, за някаква константа c.

Теорема 42 заедно със Следствие 14 дава един изключителен резултат – ако дадено графово свойство се запазва при вземане на минори, то съществува кубичен алгоритъм за неговото проверяване! И това е в сила за всяко графово свойство, което се запазва при вземане на минори; иначе казано, за всеки идеал в минорната наредба, защото идеал в минорната наредба е именно графово свойство, което се запазва при вземане на минор. Това заслужава да бъде записано като самостоятелна теорема.

Теорема 43: Тестване в кубично време за принадлежност към идеал

Ако дадено графово свойство се запазва при вземане на минори, то съществува кубичен алгоритъм за неговото проверяване.

Да си припомним примерите за идеали в минорната наредба – ацикличните графи, планарните графи, графите с число на върховото покриване не повече от k. Добре известно е, че ацикличността може да се тества в линейно време, също и планарността, така че за тези свойства не ни трябва дълбокия резултат на Robertson и Seymour. Обаче за графите с число на върховото покриване не повече от k, тази теорема е първият резултат в историята, който дава основание да твърдим, че може свойството да се

тества в кубично време! От общи съображения, дали числото на върхово покриване е не повече от k може да тестваме, непрецизно казано, във време $\Theta(n^k)$, но не по-добро. Иначе казано, от общи съображения асимптотичната долна граница е n^k. Това, че има по-добра асимптотична долна граница, а именно кубичната, изобщо не е очевидно.

Теорема 43 е приложима в огромен брой случаи. Достатъчно е да покажем, че някое графово свойство се запазва при вземане на минори, за да твърдим, че съществува полиномиален и дори кубичен алгоритъм за това свойство. Вижте статията на Lovasz [41] за нетривиални приложения на Теорема 43, например за задачата Linkless Embedding – дали даден граф може да се нарисува в тримерното пространство по такъв начин, че никои два цикъла без общи върхове да не са разположени като "обхващащи се взаимно пръстени". За тази задача от общи съображения дори не можем да твърдим, че принадлежи на класа на сложност NP. А Теорема 43 ни казва, че не само принадлежи на NP, а дори принадлежи на P, и дори нещо повече, има кубичен алгоритъм за нея. Поради простата причина, че свойството linkless embedding на граф се запазва върху минори.

За съжаление, Теорема 43 е неконструктивен резултат, което означава, че тя доказва само съществуването на кубичен алгоритъм, без да ни казва абсолютно нищо за това, какъв е самият алгоритъм. Има далечна аналогия с теоремата на Weierstrass в анализа[в], която гарантира съществуване на минимум и максимум, ако са изпълнени определени условия, но не ни казва нищо за това, как да намерим минимум или максимум. Въпреки това, Теорема 43 е резултат с фантастична стойност, както чисто теоретично, така и от практическа гледна точка – доказателството за съществуване на кубичен алгоритъм понякога вдъхновява научни изследвания, които в крайна сметка намират бърз практичен алгоритъм за задача, която дотогава е изглеждала безнадеждна. За повече подробности вижте *тази публикация* в блог, поддържан от Lance Fortnow и Bill Gasarch.

[а] "W_4" идва от *wheel*. Графът се получава от добавянето на нов връх към 4-цикъл и свързване на новия връх към и само към всеки връх от 4-цикъла.

[б] "Хиперкуб" е дефиниран в Секция 2.15.

[в] Ако реална функция е непрекъсната и ограничена в затворен интервал, то тя достига както минимум, така и максимум в него.

2.13.5 Теорема на Kuratowski.

Оригиналната статия на Kuratowski [39] (на френски) е свободно достъпна. Има превод на английски от Jan Jaworowski [40], но той не е свободно достъпен.

Очевидно е, че граф е планарен тогава и само тогава, когато всяка негова свързана компонента е планарна. Поради това, в контекста на планарните графи имаме право да допускаме без ограничение на общността, че графът, който разглеждаме, е свързан. Да си припомним "блок" е максимален по включване подграф, който е свързан и няма срязващи върхове (вж. Определение 29). Очевидно е, че граф е планарен тогава и само тогава, когато всеки негов блок е планарен. Поради това, в контекста на планарните графи имаме право да допускаме без ограничение на общността, че графът, който разглеждаме, е блок.

Започваме с едно определение, взето от [27, стр. 75].

Определение 77: Парче и съединение.

Нека $G = (V, E)$ е свързан граф. Нека $G_1 = (V_1, E_1)$ е подграф на G и нека $G_2 = G - V_1$. *Парче на* G *спрямо* G_1 е всеки от следните графи.

- За всяка свързана компонента $H = (V_H, E_H)$ на G_2: графът

$$\widehat{H} = \big(V_H \cup (N(V_H) \cap V_1), E_H \cup (\mathcal{I}(V_H) \cap \mathcal{I}(V_1))\big)$$

- За всяко ребро $(x, y) \in E \backslash E_1$, такова че $x, y \in V_1$: самото ребро (x, y).

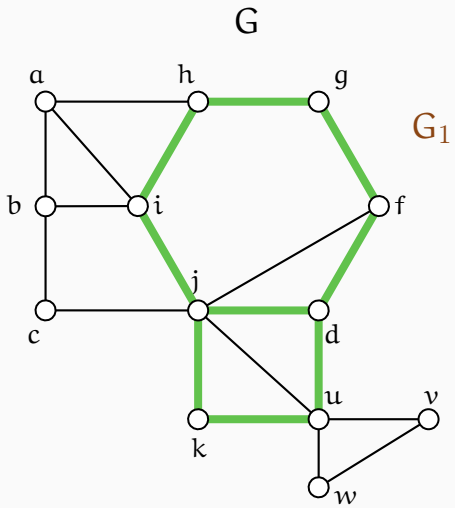
Общите върхове на кое да е парче H с G_1 се наричат *точки на захващане на* H *към* G_1, или просто *точки на захващане на* H, ако G_1 се подразбира. Останалите върхове на H са *интерналните върхове на* H[a]. Множеството от интерналните върхове на H се бележи с $\text{inter}(H)$.
Ако дадено парче има повече от една точки на захващане, то се нарича *съединение*[б].

[a] "Интернални върхове" звучи грозно, но не можем да ползваме термина "вътрешни върхове", защото той вече е дефиниран в контекста на пътищата

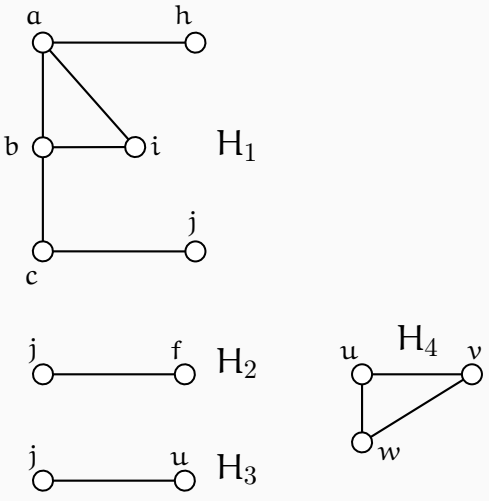
[б] В [27, стр. 75] терминът е *bridge*. Тъй като вече сме дефинирали "мост" по друг начин (Определение 26), налага се да използваме друг термин на български.

Фигура 2.84 илюстрира Определение 77. Вляво е показан граф G с подграф G_1 (нарисуван в зелено и с удебелени ребра), а вдясно са четирите парчета H_1, H_2, H_3 и H_4 на G спрямо G_1. Парчетата H_2 и H_3 са ребра и, въпреки че имат общ връх в G, като парчета на G те са отделни подграфи. Съединения са H_1, H_2 и H_3, а H_4 не е съединение, понеже има само една точка на захващане. Ако G е блок, то всяко негово парче, спрямо кой да е подграф, е съединение – обратното би значело, че G има срязващ връх.

Фигура 2.84 : Парчета спрямо подграф.



Граф G с подграф G_1.



Парчетата на G спрямо G_1 са $H_1, \ldots, H_4$.

Забележете, че ребрата (j, f) и (j, u) са две отделни съединения, наречени H_2 и H_3, а не едно съединение.

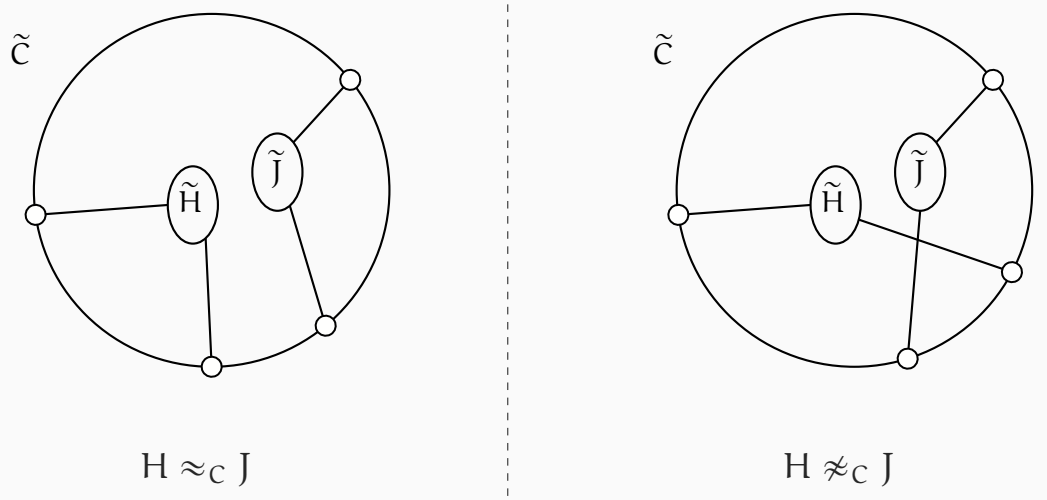
Може би вече се вижда какво е значението на съединенията за планарността: ако подграфът G_1 е планарен, то това дали целият граф G е планарен се определя еднозначно от парчетата

спрямо G_1. Нещо повече. Ако поне едно от тези съединения не е планарно, то очевидно и G не е планарен, но конверсното не е вярно – може всички съединения да са планарни и въпреки това G да не е планарен, защото някои от съединенията са захванати към G_1 по такъв начин, че не може едновременно да бъдат сложени в равнината. С други думи, разполагането на всяко от тях в равнината е зависимо от разполагането на другото. За кратък запис на тази идея ще въведем следната нотация от [27, стр. 76], но само в случай, че подграфът, спрямо който разглеждаме съединения, е цикъл.

Нотация 4: $\approx$ и $\not\approx$

Нека G е граф и C е цикъл в него. Нека H и J са две съединения спрямо C. Нека те са планарни графи със съответни планарни образи $\widetilde{H}$ и $\widetilde{J}$. Казваме, че H и J са *съвместими спрямо* C, или просто *съвместими*, ако C се подразбира, ако в равнината може да разположим $\widetilde{H}$ и $\widetilde{J}$ в едно и също лице на равнината спрямо $\widetilde{C}$, където $\widetilde{C}$ е планарен образ на C. Ако H и J са съвместими, пишем $H \approx_C J$, в противен случай пишем $H \not\approx_C J$. Ако C се подразбира може да пишем просто $H \approx J$ или $H \not\approx J$, съответно.

Фигура 2.85 : Съвместими и несъвместими съединения спрямо цикъл.



Фигура 2.85 илюстрира Нотация 4.

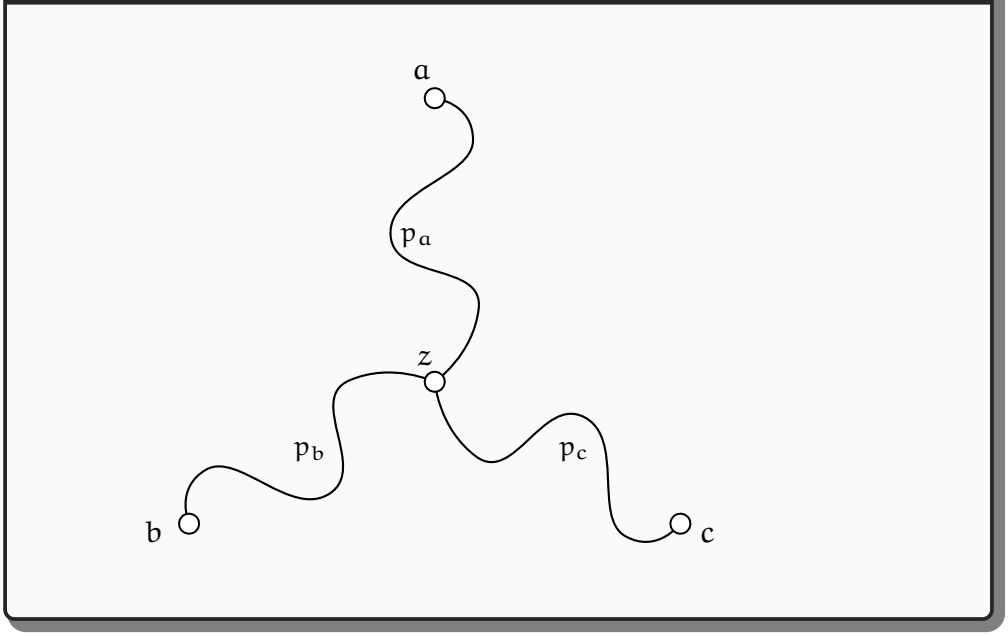
Лема 17 ползва Определение 17.

Лема 17

Нека G е свързан граф и H е произволно съединение спрямо произволен подграф на G. Нека H има поне три точки на захващане a, b и c. Тогава съществува връх $z \in \mathrm{inter}(H)$, такъв че в H има z-a път p_a, z-b път p_b и z-c път p_c, които три пътя са независими.

Доказателство: Образно казано, Лема 17 твърди, че в H има подграф, който прилича на звезда с три лъча. Центърът на звездата е z, а трите ѝ лъча са p_a, p_b и p_c. Вижте Фигура 2.86.

Фигура 2.86 : "Звезда с 3 лъча" при 3 точки на захващане.



Очевидно съединението H не е само едно ребро, така че $\mathrm{inter}(H) \neq \varnothing$. Да разгледаме произволен $w \in \mathrm{inter}(H)$. Тъй като H е свързан граф, в него има w-a път q_a, w-b път q_b и w-c път q_c. Започвайки с w, q_a, q_b и q_c, ще построим желаните z, p_q, p_b и p_c. Ако q_a, q_b и q_c са независими, преминаваме към **Стъпка 3**. В противен случай, преминаваме към **Стъпка 1**.

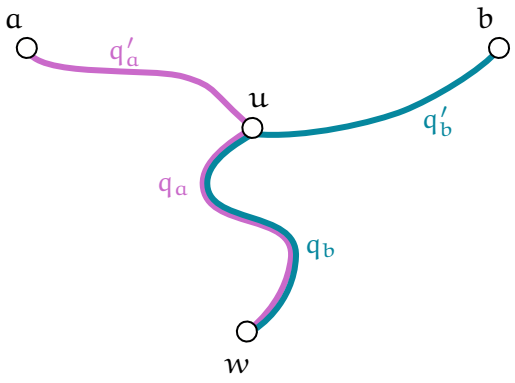
Стъпка 1: Ако q_a и q_b са независими пътища, преминаваме към **Стъпка 2**. Нека q_a и q_b не са независими. Да разгледаме върха $u \in V(q_a) \cap V(q_b)$, който е максимално отдалечен от w както в q_a, така и в q_b. Ясно е, че $u \neq w$.

Без ограничение на общността можем да допуснем, че $u \neq a$ и $u \neq b$, защото:

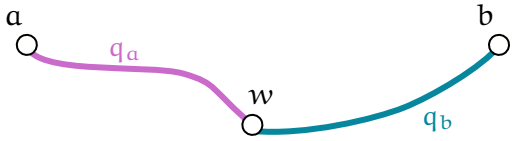
- Ако $u = a$, то задължително съществува друг w-a път, в който максимално отдалеченият от w общ връх с q_b не е връх a. Ако нямаше такъв друг w-a път, то a би се намирал върху всеки w-b път и тогава a и b не биха били върхове от едно и също съединение. И така, можем да изберем като q_a този друг път.

- Напълно аналогично, винаги можем да изберем като q_b такъв w-b път, в който максимално отдалеченият от w общ връх с q_a не е връх b.

Нека q'_a е подпътят на q_a от u до a и нека q'_b е подпътят на q_b от u до b. Както видяхме, можем да мислим за q_a и q_b като за пътища с общ край w, които се "разделят завинаги" след u на q'_a и q'_b (преди u, q_a и q_b може да "вървят заедно" като на фигурата, може да се "събират и разделят", това няма значение):



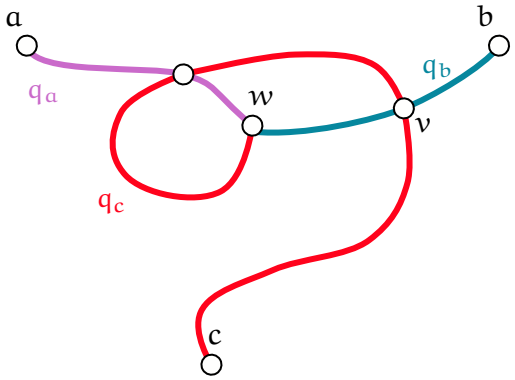
Преименуваме u на w, q'_a на q_a и q'_b на q_b. В някакъв смисъл, "изрязваме" общата част, оставяйки независими пътища:



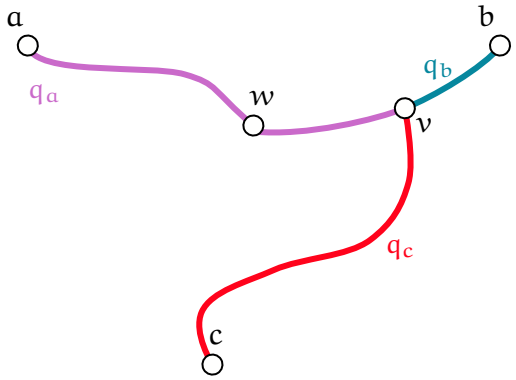
Преминаваме към **Стъпка 2**.

Стъпка 2: Нека q_c е произволен w-c път спрямо новата стойност на w. Ако трите пътя q_a, q_b и q_c са независими, преминаваме към **Стъпка 3**. В противен случай, също както в **Стъпка 1**, допускаме без ограничение на общността, че c не се намира нито в q_a, нито в q_b, и освен това q_c не съдържа нито a, нито b. С други думи, поне единият от q_a и q_b има общи **вътрешни** върхове с q_c. Ще разгледаме случаят, в който и q_a, и q_b имат общи вътрешни върхове с q_c. Читателят лесно може да съобрази какво да правим, ако точно единият от q_a и q_b има общи вътрешни върхове с q_c.

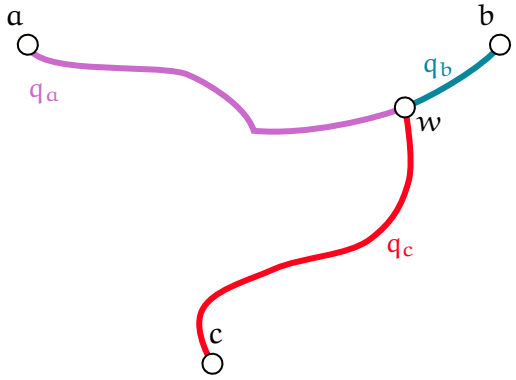
И така, и q_a, и q_b имат общи вътрешни върхове с q_c. Дефинираме v като върха от $V(q_c) \cap (V(q_a) \cup V(q_b))$, който най-отдалечен от w в q_c. Без ограничение на общността, нека $v \in V(q_b)$. Следната фигура илюстрира текущите допускания:



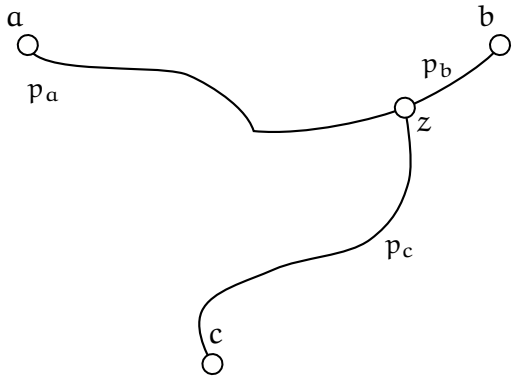
Наричаме "q_a" пътят от v до a в $q_a \cup q_b$, наричаме "q_b" пътят от v до b в q_b, и наричаме "q_c" пътят от v до c в q_c. Очевидно новите q_a, q_b и q_c са независими:



Преименуваме v на w и преминаваме към **Стъпка 3**:



Стъпка 3: Преименуваме w на z, q_a на p_a, q_b на p_b и q_c на p_c:



Доказахме, че в H съществува структурата, показана на Фигура 2.86. □

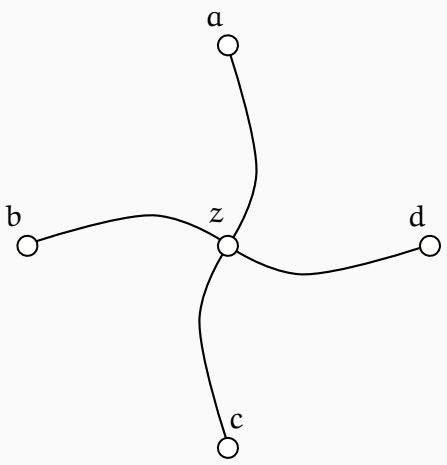
Лема 18

Нека G е свързан граф и J е произволно съединение спрямо произволен подграф на G. Нека J има поне четири точки на захващане a, b, c и d. Тогава поне едно от следните две е вярно.

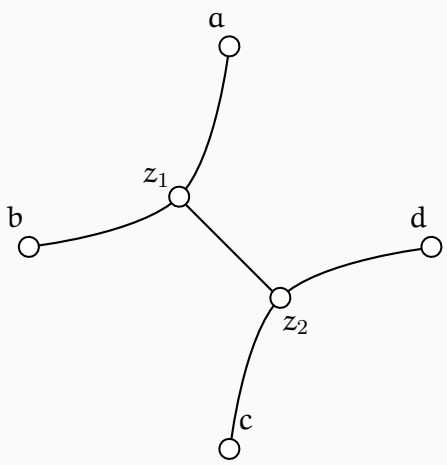
- Съществува връх $z \in \mathrm{inter}(J)$, такъв че в J има z-a път, z-b път z-c път и z-d път, които четири пътя са независими. В такъв случай казваме, че *в* J *има звезда с четири лъча*. Накратко пишем *в* J *има* ✦.
- Спрямо някакво разбиване на $\{a, b, c, d\}$ на две двуелементни подмножества, да кажем $\{a, b\}$ и $\{c, d\}$, съществуват $z_1, z_2 \in \mathrm{inter}(J)$, такива че в J има z_1-a път, z_1-b път, z_2-c път и z_2-d път, както и z_1-z_2 път, които пет пътя са независими. В такъв случай казваме че *в* J *има буква H*. Накратко пишем *в* J *има* ⋓.

Доказателство: Образно казано, Лема 18 твърди, че в H има подграф, който прилича на звезда с четири лъча или на знака ⋓. Вижте Фигура 2.87.

Фигура 2.87 : "Звезда с 4 лъча" (✦) или "буква H" (⋓) при 4 точки на захващане.



Звезда с 4 лъча. Ползваме знак ✦.

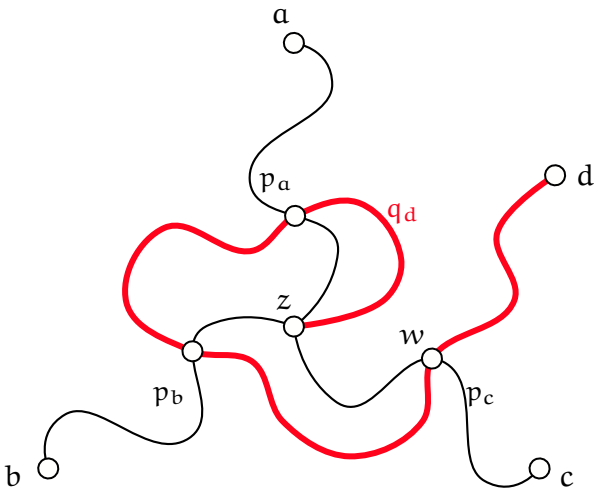


Буква H. Ползваме знак ⋓.

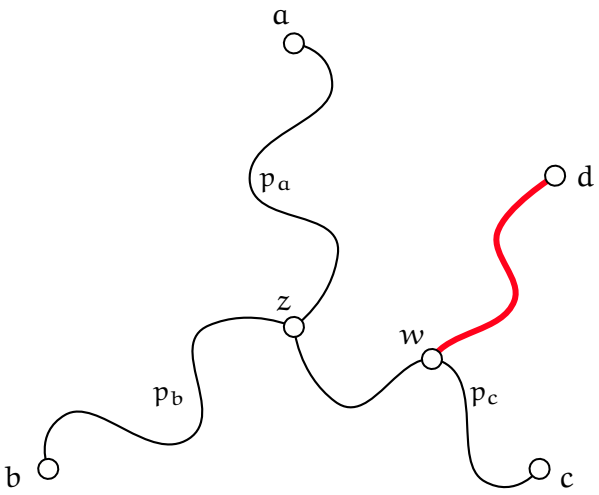
Да игнорираме за момент връх d. Лема 17 е приложима за a, b и c. Съгласно нея, в H съществува подграф, който изглежда като този на Фигура 2.86. Сега да разгледаме и d.

Ако съществува z-d път, който е независим с всеки от p_a, p_b и p_c, то очевидно има ✦.

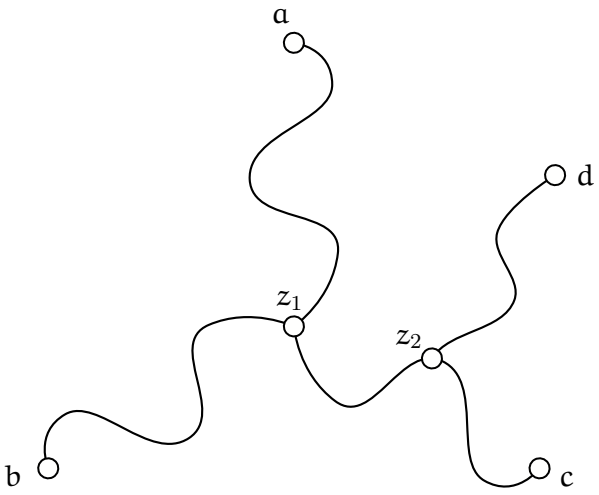
В противен случай, да разгледаме произволен z-d път q_d. Без ограничение на общността, нека най-отдалеченият от z връх в q_d, общ с p_a или p_b или p_c да е някой връх w от p_c. С разсъждения, аналогични на разсъжденията в доказателството на Лема 17 показваме, че w е вътрешен връх както за p_c, така и за q_d. Нещата изглеждат, примерно, така:



Да разгледаме само подпътя на q_d от w до d включително:



Сега вече е ясно, че ако преименуваме z на z_1 и w на z_2, има ⋉:



□

Теоремата на Kuratowski е нетривиален резултат, чието доказателство, ако се прави подробно, е доста дълго. Краят на доказателството, изложено тук, е на стр. 202.

Теорема 44: Теорема на Kuratowski

За всеки граф, той е планарен тогава и само тогава, когато не съдържа подграф, хомеоморфен на K_5 или $K_{3,3}$.

Доказателство: В едната посока, ако графът е планарен, то той не съдържа подграф, хомеоморфен на K_5 или $K_{3,3}$. Ще докажем еквивалентното съждение: ако графът съдържа подграф, хомеоморфен на K_5 или $K_{3,3}$, то той не е планарен. Но от Следствие 11 и Следствие 13 знаем, че нито K_5 е планарен, нито $K_{3,3}$ е планарен. Тогава очевидно всеки граф, хомеоморфен на K_5 или $K_{3,3}$, не е планарен. Тогава всеки граф, който съдържа подграф, хомеоморфен на K_5 или $K_{3,3}$, не е планарен. ✓

В другата посока, ако граф не съдържа подграф, хомеоморфен на K_5 или $K_{3,3}$, то той е планарен. Доказателството[†] е по индукция по броя на ребрата.

База: Очевидно твърдението е вярно за всеки граф с едно или две ребра. ✓

Индуктивно предположение: Нека твърдението е вярно за всички графи с по-малко от M ребра.

Индуктивна стъпка: Разглеждаме произволен граф $G = (V, E)$ с M ребра. Да допуснем, че G не съдържа подграф, хомеоморфен на K_5 или $K_{3,3}$, и G не е планарен[‡]. Допускането, че G не е планарен, влече следното:

1. G е свързан. В противен случай, G има поне две свързани компоненти. Ако повече от една от тях имат поне едно ребро, то всяка от тях има по-малко от M ребра, поради което всяка от тях е планарна от индуктивното предположение, което влече, че целият граф е планарен. В противен случай точно една от тях съдържа всички M ребра, а останалите са изолирани върхове, така че без ограничение на общността разглеждаме само тази с ребрата.

2. G няма срязващи върхове. Другояче казано, G има точно един блок, а именно самият G е блок. В противен случай, G има поне два блока, всеки от които има по-малко от M ребра и е планарен съгласно индуктивното предположение, а щом блоковете са планарни, то очевидно и самият G е планарен.

3. Това е по-сложно и неочевидно твърдение и ще го обособим като лема.

Лема 19

За всяко ребро $e = (x, y)$ от E е вярно, че $G - e$ съдържа цикъл C, такъв че $x \in V(C)$ и $y \in V(C)$.

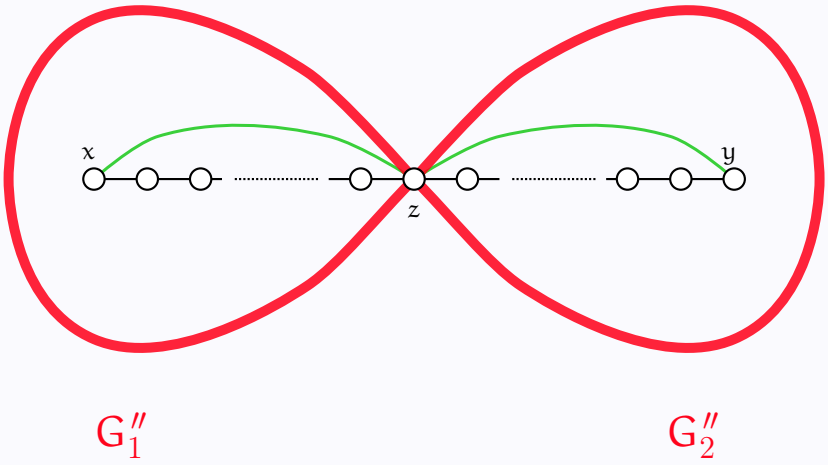
[†]По учебника на Gibbons [27, стр. 77–80, Theorem 3.5].

[‡]Това е доказателство с допускане на противното. Нека p е съждението "G съдържа подграф, хомеоморфен на K_5 или на $K_{3,3}$". Нека q е съждението "G е планарен". Теоремата на Kuratowski казва, в съждителна логика, $\neg p \leftrightarrow q$. За простота игнорираме факта, че всъщност теоремата се изразява чрез предикат. Първо доказахме $q \rightarrow \neg p$. В момента доказваме $\neg p \rightarrow q$. Това е еквивалентно на $p \vee q$. Допускайки противното, ние допускаме $\neg(p \vee q) \equiv \neg p \wedge \neg q$. Но това е точно "$G$ не съдържа подграф, хомеоморфен на K_5 или $K_{3,3}$, и G не е планарен".

Доказателство: Нека $G - e$ бъде наречен G'. Първо забелязваме, че G' е свързан, понеже G няма срязващи върхове. Да допуснем, че такъв цикъл (който съдържа и x, и y) в G' няма. Тогава съществува връх z, такъв че всеки път между x и y съдържа връх z – а поне един път между x и y има, понеже G' е свързан. Тогава z е срязващ връх в G' и освен това, x и y се намират в различни компоненти на $G \ominus z$. Да речем, че компонентата на $G \ominus z$, която съдържа x, се казва G'_1, а компонентата, която съдържа y, се казва G'_2.

Ако G'_1 не съдържа реброто (x, z), добавяме реброто (x, z) към него. Независимо от това дали сме добавили ребро или не, казваме на този граф G''_1. Аналогично, ако G'_2 не съдържа реброто (y, z), добавяме реброто (y, z) към него и независимо от това дали сме добавили ребро или не, казваме на този граф G''_2. Фигура 2.88 показва графа, който сме конструирали до момента.

Фигура 2.88 : Графът G' плюс (x, z) и (y, z).



Графът след изтриването на реброто (x, y) и добавянето на (x, z) и (y, z).

Ще докажем, че G''_1 не съдържа подграф, хомеоморфен на K_5 или $K_{3,3}$.

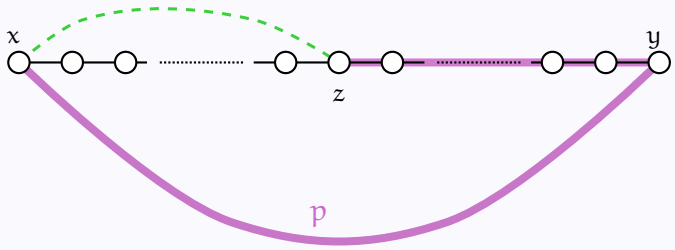
Да допуснем, че G''_1 съдържа подграф, хомеоморфен на K_5 или $K_{3,3}$. Забелязваме, че тогава G съдържа подграф, хомеоморфен на G''_1, защото:

- или G''_1 съвпада с подграфа на G, индуциран от $V(G''_1)$, или G''_1 се състои от този подграф плюс още едно ребро (а именно, (x, z)). Неформално казано, единственото нещо, което може би се намира в G''_1, но не се намира в G, е реброто (x, z).
- дори G''_1 да съдържа реброто (x, z), а то да не е в G, пътят $p = x, y, \ldots, z$ в G може да изиграе ролята на реброто (x, z) (вж. Фигура 2.89).

И така, оригиналният G съдържа подграф, хомеоморфен на G''_1. Щом G съдържа подграф, хомеоморфен на G''_1 и G''_1 съдържа подграф, хомеоморфен на K_5 или $K_{3,3}$, то и

оригиналният G съдържа подграф, хомеоморфен на K_5 или $K_{3,3}$. Това обаче противоречи на по-рано направеното допускане, че G не съдържа подграф, хомеоморфен на K_5 или $K_{3,3}$. Следователно, G_1'' не съдържа подграф, хомеоморфен на K_5 или $K_{3,3}$. Напълно аналогично доказваме, че G_2'' не съдържа подграф, хомеоморфен на K_5 или $K_{3,3}$.

Фигура 2.89 : В оригиналния граф (може да) няма ребро (x,z), но съществува пътят p.



Щом нито G_1'', нито G_2'' съдържа подграф, хомеоморфен на K_5 или $K_{3,3}$, то и G_1'', и G_2'' са планарни от индуктивното предположение (и двата има по-малко от M ребра). Тогава можем да конструираме планарни вписвания на G_1'' и G_2'', които ще наречем съответно $\widetilde{G_1''}$ и $\widetilde{G_2''}$. Нещо повече, съгласно Наблюдение 27, според което всяко лице може да бъде външно, винаги можем да направим такова планарно вписване, че всяко предварително избрано ребро да е върху външното лице. И така, строим $\widetilde{G_1''}$ по такъв начин, че (x,z) да е върху външното лице, и $\widetilde{G_2''}$ по такъв начин, че (y,z) да е върху външното лице. Тогава очевидно е възможно да "сглобим" едно планарно вписване от $\widetilde{G_1''}$ и $\widetilde{G_2''}$, "слепвайки" ги в общия връх z. После, ако в оригиналния G е нямало ребро (x,z), то изтриваме планарния му образ, и ако в оригиналния G е нямало ребро (y,z), то изтриваме неговия планарен образ. И после поставяме планарно ребро, съответстващо на (x,y). С това построяваме планарно вписване на G, в противоречие с допускането, че G не е планарен.

Важно е да се разбере защо добавяме ребра (x,z) и (y,z) в това доказателство. Ние искаме да построим планарно вписване на G, слепвайки вписвания на два подграфа с общ връх z и после поставяйки планарно ребро, съответстващо на (x,y). Но ако вземем произволни планарни вписвания на тези подграфи е възможно да не можем да сложим планарно ребро между планарните x и y, защото планарните x и y са "затворени" в някакви вътрешни лица и нямат "директна видимост". За да сме сигурни, че планарните вписвани на подграфите са такива, че и планарният x, и планарният y са върху едно и също лице (нека е външното лице), достатъчно е планарният x да е крайна точка на планарно ребро $\widetilde{e_1}$ от външното лице и планарният y да е крайна точка на планарно ребро $\widetilde{e_2}$ от външното лице. Нещо повече, тъй като двете планарни вписвания биват "слепени" в планарния връх z, трябва планарният z да е крайна точка както на $\widetilde{e_1}$, така и на $\widetilde{e_2}$. Излиза, че $\widetilde{e_1}$ е планарният образ на (x,z), а $\widetilde{e_2}$ е планарният образ на (y,z). Следователно, за да работи конструкцията със "слепването", необходимо е да има ребра (x,z) и (y,z) в блоковете, съдържащи съответно x и y.

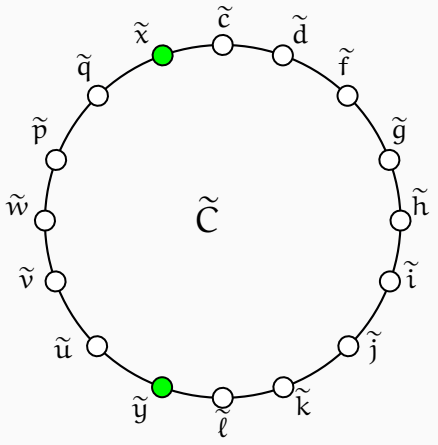
С което доказахме Лема 19. ✓

Разглеждаме произволно ребро $e = (x, y)$ в G. Вече знаем, че $G' = G - e$ е свързан, че в G' има цикъл C, такъв че x и y са върхове в този цикъл, и че G' няма подграфи, хомеоморфни на K_5 или $K_{3,3}$. Освен това G' има $M - 1$ ребра, така че съгласно индуктивното предположение, G' е планарен граф. Нека $\widetilde{G'}$ е негово планарно вписване. Нека $\tilde{x}$, $\tilde{y}$, $\tilde{e}$ и $\widetilde{C}$ са планарните образи съответно на x, y, e и C. Очевидно $\widetilde{C}$ е проста затворена крива в равнината, тъй като C е прост цикъл. В общия случай $\widetilde{C}$ не огражда едно лице, а множество лица на вписването.

Без ограничение на общността допускаме, че измежду всички планарни цикли, минаващи през $\tilde{x}$ и $\tilde{y}$, $\widetilde{C}$ е такъв планарен цикъл, който **огражда максимален брой лица на вписването**. Очевидно $\widetilde{C}$ разделя равнината на вътрешен и външен район. За всяко съединение H на G' спрямо C, казваме, че H е *вътрешно съединение*, ако планарният му образ, да го наречем $\widetilde{H}$, се намира във вътрешния район спрямо $\widetilde{C}$. Ако H не е вътрешно, казваме, че H е *външно*.

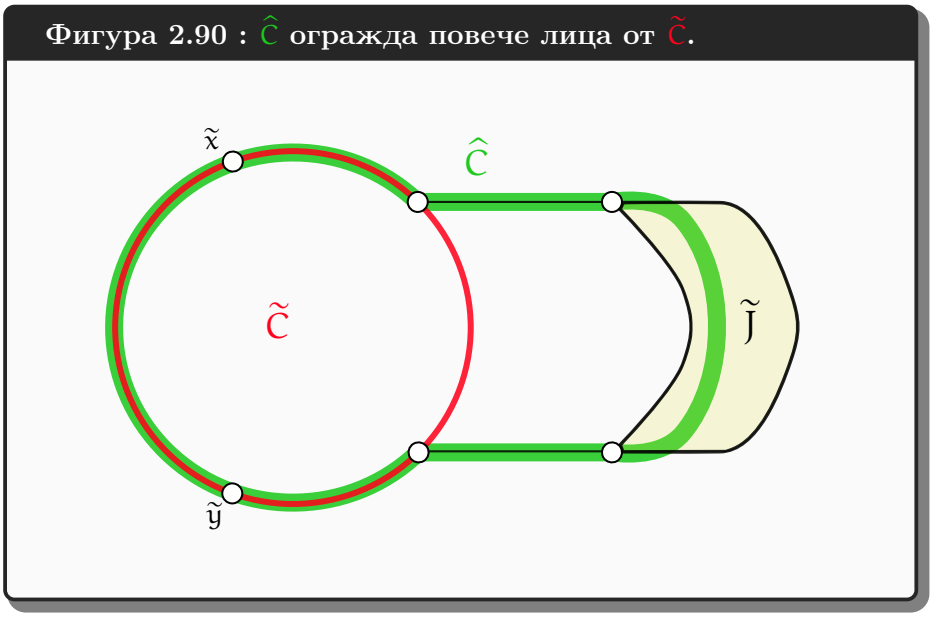
Нотация 5: $x\rightsquigarrow y$ и $y\rightsquigarrow x$

Да изберем някаква произволна посока върху $\widetilde{C}$. Нека тази посока е по часовниковата стрелка. Нека $\widetilde{C}_{x,y}$ означава тази дъга на $\widetilde{C}\backslash\{\tilde{x}, \tilde{y}\}$, която лежи в посока по часовниковата стрелка след $\tilde{x}$ нататък, а $\widetilde{C}_{y,x}$ означава тази дъга на $\widetilde{C}\backslash\{\tilde{x}, \tilde{y}\}$, която лежи в посока по часовниковата стрелка след $\tilde{y}$. Тогава нотацията "$x\rightsquigarrow y$ " означава множеството от всички върхове на C, чиито планарни образи се намират в $\widetilde{C}_{x,y}$, а "$y\rightsquigarrow x$ " означава множеството от всички върхове на C, чиито планарни образи се намират в $\widetilde{C}_{y,x}$. Очевидно, $x, y \notin x\rightsquigarrow y$ и $x, y \notin y\rightsquigarrow x$. Забележете, че $x\rightsquigarrow y$ или $y\rightsquigarrow x$ —но не и двете—може да е празното множество.
Следната фигура илюстрира тази нотация. $x\rightsquigarrow y = \{c, d, f, g, h, i, j, k, \ell\}$ и $y\rightsquigarrow x = \{u, v, w, p, q\}$.



Съществено наблюдение е, че нито едно външно съединение спрямо $\widetilde{C}$ не може да има повече от един общ връх с $(x\rightsquigarrow y) \cup \{x, y\}$ и не може да има повече от един общ връх с $(y\rightsquigarrow x) \cup \{x, y\}$. Да допуснем обратното. Нека някое външно съединение J има поне два общи върха с $(x\rightsquigarrow y) \cup \{x, y\}$. Нека неговият планарен образ е $\widetilde{J}$. Веднага следва, че има планарен цикъл $\widehat{C}$, различен от $\widetilde{C}$, минаващ през $\tilde{x}$ и $\tilde{y}$, а също така и през $\widetilde{J}$, и ограждащ повече лица, отколкото огражда $\widetilde{C}$. Последното е в противоречие с направеното по-рано допускане, че измежду всички такива планарни цикли, $\widetilde{C}$ е такъв, който огражда максимален брой лица на вписването. На Фигура 2.90 съединението $\widetilde{J}$ е външно спрямо планарния цикъл $\widetilde{C}$ и освен това J има два общи върха с $x\rightsquigarrow y$, поради което съществува планарен цикъл $\widehat{C}$, който огражда повече лица от $\widetilde{C}$.

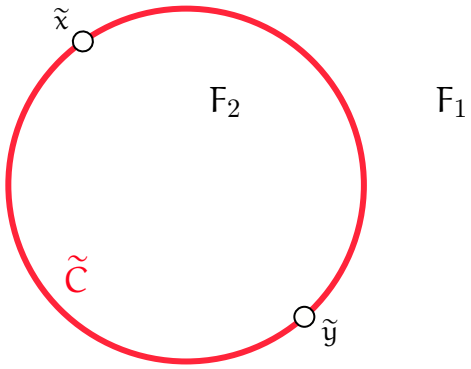
Фигура 2.90 : $\widehat{C}$ **ограждa повече лица от** $\widetilde{C}$**.**



От друга страна, вътрешните съединения спрямо $\widetilde{C}$ може да има повече от един общ връх с $x \rightsquigarrow y \cup (\{x, y\})$ и може да има повече от един общ връх с $y \rightsquigarrow x \cup (\{x, y\})$.

Да си припомним, че не-планарният граф G се получава от планарния граф G' с добавяне на реброто $e = (x, y)$. Ние вече разглеждаме някакво планарно вписване на G', което нарекохме $\widetilde{G'}$. Ключово наблюдение за доказателството е, че нещо трябва да "пречи" на поставянето на планарен образ на e в $\widetilde{G'}$, инак G би бил планарен. По-точно казано, трябва $\widetilde{x}$ и $\widetilde{y}$ **да не са върху едно и също лице** на $\widetilde{G'}$, защото, ако има такова лице, че $\widetilde{x}$ и $\widetilde{y}$ са планарни върхове от ограждащата го крива, то можем да сложим планарно ребро $\widetilde{e} = (\widetilde{x}, \widetilde{y})$ в това лице, без да нарушим планарността, и по този начин да получим планарно вписване на G, в противоречие с допускането, че G не е планарен.

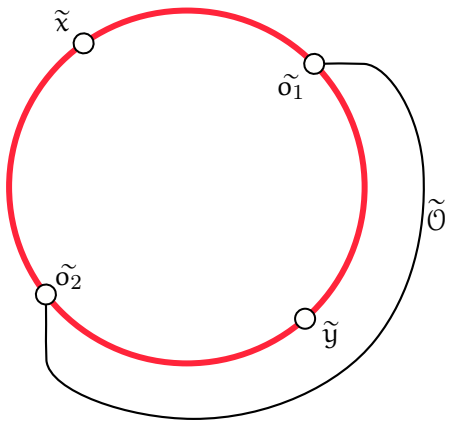
Да разгледаме **само** $\widetilde{C}$, игнорирайки останалото от $\widetilde{G'}$. Спрямо $\widetilde{C}$ равнината има две лица. Да речем, външно лице F_1 и вътрешно лице F_2, като $\widetilde{x}$ и $\widetilde{y}$ са и на F_1, и на F_2:



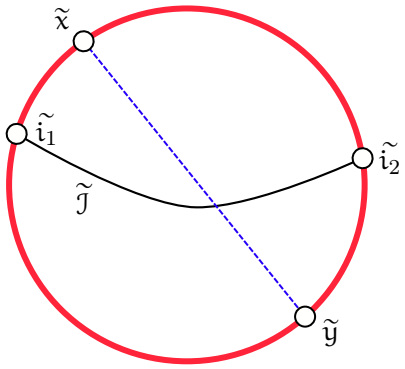
Забелязваме, че нещо, което да "пречи" на поставянето на $\widetilde{e} = (\widetilde{x}, \widetilde{y})$ трябва да има както в F_1, така и в F_2.

- По отношение на $\widetilde{C}$ трябва има поне едно съединение $\mathcal{O}$, такова че неговият планарен образ, да го наречем $\widetilde{\mathcal{O}}$, да се намира в F_1 и пречи да бъде поставено $\widetilde{e}$ в F_1. За да може $\widetilde{\mathcal{O}}$ да пречи на поставянето планарно ребро между $\widetilde{x}$ и $\widetilde{y}$, трябва $\mathcal{O}$ да има общ връх o_1

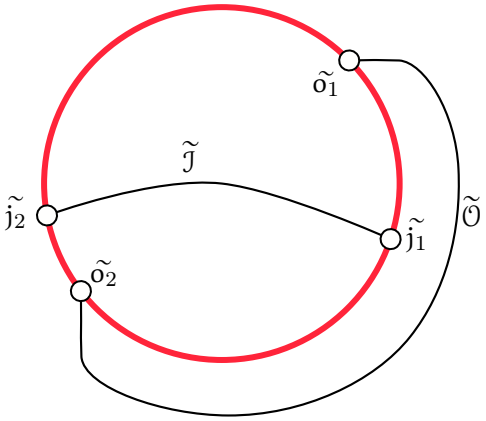
с $x \curvearrowright y$ и общ връх o_2 с $y \curvearrowright x$. Както вече видяхме, $\mathcal{O}$ не може да има повече от един общ връх нито с $x \curvearrowright y$, нито с $y \curvearrowright x$. Нека $\widetilde{o_1}$ и $\widetilde{o_2}$ са планарните образи съответно на o_1 и o_2. За простота, ще си представяме $\widetilde{\mathcal{O}}$ като едно планарно ребро:



- Аналогично, трябва има поне едно съединение, да го наречем $\mathcal{I}$, в F_2, чийто планарен образ $\widetilde{\mathcal{I}}$ пречи на поставяне на $\widetilde{e}$ в F_2. Тъй като сега няма твърдо ограничение за точно един общ връх с $x \curvearrowright y$ или точно един общ връх с $y \curvearrowright x$, ще смятаме, че конфигурацията, показана на предната фигура ($\widetilde{\mathcal{C}}$ и $\widetilde{\mathcal{O}}$) е фиксирана и **спрямо нея** разглеждаме различните възможности за $\widetilde{\mathcal{I}}$. От една страна, $\widetilde{\mathcal{I}}$ трябва да се захваща за два върха i_1 и i_2 съответно в $x \curvearrowright y$ и $y \curvearrowright x$, чиито планарни образи са съответно $\widetilde{i_1}$ и $\widetilde{i_2}$, така че $\widetilde{\mathcal{I}}$ да пречи на поставянето на $\widetilde{e}$ (показано със синя пунктирана линия):



От друга страна обаче, $\widetilde{\mathcal{I}}$ трябва да се захваща за два върха j_1 и j_2 съответно в $o_1 \curvearrowright o_2$ и $o_2 \curvearrowright o_1$, чиито планарни образи са съответно $\widetilde{j_1}$ и $\widetilde{j_2}$, така че $\widetilde{\mathcal{I}}$ и $\widetilde{\mathcal{O}}$ да не може да бъдат поставени в едно и също лице спрямо $\widetilde{\mathcal{C}}$ – нито във F_1, нито във F_2:



До края на доказателството разглеждаме само $\widetilde{C}$, $\widetilde{\mathcal{O}}$, $\widetilde{J}$, $\widetilde{x}$, $\widetilde{y}$, $\widetilde{o_1}$, $\widetilde{o_2}$, $\widetilde{i_1}$, $\widetilde{i_2}$, $\widetilde{j_1}$ и $\widetilde{j_2}$. Забележете, че x, y, o_1, o_2, i_1, i_2, j_1 и j_2 **не са непременно** осем различни върха. Това, което знаем за задължителните различия между тях е:

- x, y, o_1 и o_2 са два по два различни,
- $\{i_1, i_2\} \cap \{x, y\} = \varnothing$ и
- $\{j_1, j_2\} \cap \{o_1, o_2\} = \varnothing$.

В този момент от доказателството, в учебника на Gibbons [27, стр. 79] се разглеждат само пет конфигурации, наречени "essentially different", като във всяка от тях се открива подграф, хомеоморфен на K_5 или $K_{3,3}$. За съжаление, Gibbons не обяснява дори в най-общи линии защо точно тези конфигурации са достатъчни за разглеждане. Тези лекционни записки са предназначени за читателска аудитория, която за пръв път се сблъсква с подобни разсъждения. Поради това ще направим детайлно разбиване на случаи и подслучаи. Единственото опростяване на доказателството ни ще бъде в установяване на изоморфизъм—винаги когато е възможно—между някои от подграфите, които разглеждаме. Очевидно няма смисъл да се разглеждат поотделно изоморфни графи, за да се покаже, че във всеки от тях има подграф, хомеоморфен на K_5 или $K_{3,3}$. Установяване на изоморфизъм между два графа с лист и молив може да е трудоемко и досадно занимание. Установяване на липса на изоморфизъм с лист и молив е значително по-трудоемко (и досадно). Поради това ще използваме софтуер, за да установяваме изоморфизъм или липса на изоморфизъм.

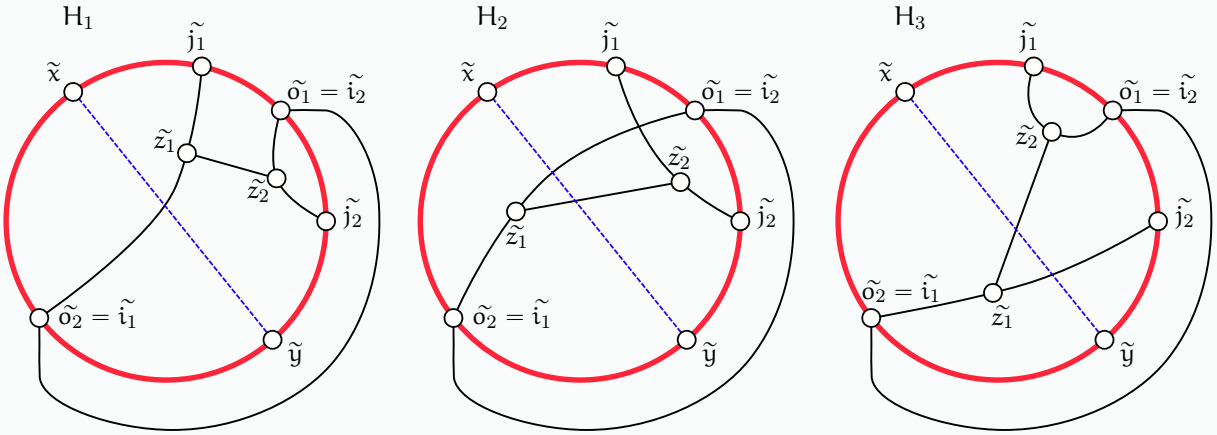
Допълнение 23: nauty: програма за намиране изоморфизми на графи

Най-известната и, в общия случай, най-бързата компютърна програма за установяване дали два графа са изоморфни или не, е nauty. Нейният автор е *професор Brendan McKay* от Australian National University. Първата версия на nauty се появява малко след 1980 г. Следват много подобрени версии, като около 2008 г. към пакета на nauty се добавя програмата Traces на *професор Adolfo Piperno* от Sapienza Universita di Roma. Страници на nauty с Traces има *тук* и *тук*. Упътване за потребителя на nauty, версия 2.6, има *тук*. Теоретичната обосновка на целия софтуерен пакет се съдържа в статията "Practical graph isomorphism, II" на McKay и Piperno [42].

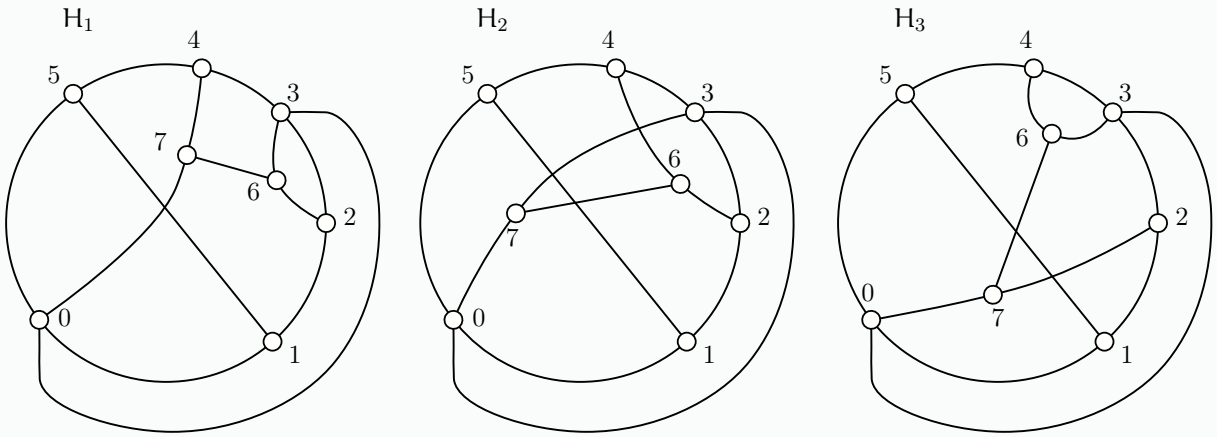
Името "nauty" не е правописна грешка[a]. Идва от **n**o **aut**omorphisms. *Автоморфизъм*

е изоморфизъм на граф в себе си. Много общо казано, основната идея на nauty е да пресметне автоморфизмите на двата графа и да ги сравни. Пакетът nauty и Traces има огромен брой приложения, като установяването на изоморфизъм или липса на изоморфизъм е само една от тях. Пакетът е написан на C и се предлага като сорс код с отворен лиценз. След компилиране се генерират множество обектни файлове, които потребителят може да използва, за да свърже собствена програма на C с тях и така да използва функциите, които те съдържат. API-то е описано подробно в упътването, посочено горе.

Пакетът предлага и проста интерактивна програма `dreadnaut`, чрез която потребителят може да използва основните възможности на пакета. Графите може да бъдат въведени директно в `dreadnaut`, но може да бъдат описани в отделни файлове, които да се зареждат от `dreadnaut`. Описанието на граф е чрез списъци на съседство, като при неориентирани графи е допустимо всяко ребро да се описва един път, а не два пъти. За пълнота, в примера за използване на `dreadnaut`, който ще дадем, списъците на съседство са "истински", тоест всяко ребро се появява точно два пъти. Форматът на описанието на граф е прост текст, в който първият ред е `n=z g`, където `z` е броят на върховете, а на следващите `z` реда съдържат списъците на съседство. Идентификаторите на върховете трябва да бъдат числа; а именно, числата $0, 1, \ldots, z-1$. Тази особеност е леко досадна, ако в описанието на графа, което сме си съставили, етикетите на върховете са някакви стрингове – тогава се налага да транслираме тези стрингове в числата $0, 1, \ldots, z$. Като пример ще разгледаме трите графа H_1, H_2 и H_3 на стр. 197. Ето ги и тук:



Да им преименуваме върховете, така че да бъдат с етикети, подходящи за nauty:



Да направим три файла h1.txt, h2.txt и h3.txt, съдържащи описанието (тоест, списъците на съседство) на, съответно, H_1, H_2 и H_3:

h1.txt	h2.txt	h3.txt
`n=8 g`	`n=8 g`	`n=8 g`
`0: 5 7 1 3;`	`0: 5 7 1 3;`	`0: 7 5 3 1;`
`1: 5 0 2;`	`1: 2 5 0;`	`1: 0 2 5;`
`2: 1 6 3;`	`2: 1 6 3;`	`2: 7 3 1;`
`3: 4 6 2 0;`	`3: 4 7 2 0;`	`3: 4 6 2 0;`
`4: 5 7 3;`	`4: 5 3 6;`	`4: 5 6 3;`
`5: 4 1 0;`	`5: 4 1 0;`	`5: 1 4 0;`
`6: 3 2 7;`	`6: 4 7 2;`	`6: 7 3 4;`
`7: 4 0 6.`	`7: 3 6 0.`	`7: 2 6 0.`

Извикването на `dreadnaut` от командния ред върху тези файлове става по следния начин. Първо да видим дали H_1 и H_2 са изоморфни.

```
$ dreadnaut
Dreadnaut version 2.7 (64 bits).
> c
> < h1.txt
> x @
(0 3)(1 2)(4 7)(5 6)
level 1:  2 cells; 4 orbits; 1 fixed; index 2/6
4 orbits; grpsize=2; 1 gen; 5 nodes; maxlev=2
canupdates=3; cpu time = 0.00 seconds
> < h2.txt
> x
(1 5)(2 4)
level 1:  6 orbits; 2 fixed; index 2
6 orbits; grpsize=2; 1 gen; 3 nodes; maxlev=2
canupdates=1; cpu time = 0.00 seconds
> ##
h and h' are different.
> q
```

`dreadnaut` е интерактивна програма. Промптът ѝ е `>` . Командата `c` кара `dreadnaut` да прави определено оцветяване на графите, върху които ще работи; това трябва да се направи, ако търсим изоморфизми. Командата `< h1.txt` кара `dreadnaut` да прочете файла `h1.txt`. `x @` са две команди, като `x` вика самата nauty, а `@` съхранява резултата от нейната работа в друг работен граф, наречен $\mathtt{h}'$ (дефолт работният граф е `h`). После `< h2.txt` чете съдържанието на `h2.txt`. После `x` пак вика nauty, но сега върху последния прочетен граф. И накрая `##` сравнява двата работни графа `h` и $\mathtt{h}'$ и, ако са изоморфни, дава явно изоморфизъм, а ако не са, съобщава това на потребителя. В случая H_1 и H_2 не са изоморфни. Командата `q` прекратява работата на `dreadnaut`.

Сега да видим работата на `dreadnaut` върху изоморфни графи, каквито са H_1 и H_3.

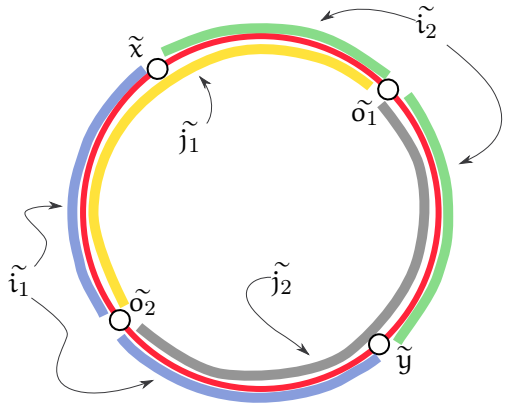
```
$ dreadnaut
Dreadnaut version 2.7 (64 bits).
> c
> < h1.txt
> x @
(0 3)(1 2)(4 7)(5 6)
level 1:  2 cells; 4 orbits; 1 fixed; index 2/6
4 orbits; grpsize=2; 1 gen; 5 nodes; maxlev=2
canupdates=3; cpu time = 0.00 seconds
> < h3.txt
> x
(0 3)(1 6)(2 7)(4 5)
level 1:  2 cells; 4 orbits; 1 fixed; index 2/6
4 orbits; grpsize=2; 1 gen; 6 nodes (3 bad leaves); maxlev=2
canupdates=1; cpu time = 0.00 seconds
> ##
h and h' are identical.
 0-0 1-5 2-4 3-3 4-2 5-1 6-6 7-7
> q
```

Работният граф `h` в края е H_3, чието описание е заредено последно, а $\mathtt{h}'$ е H_1. Показаното съответствие се чете така: във всяка двойка върхове, този вляво е от H_3, а този вдясно, от H_1. Нека читателят се убеди сам, че програмата наистина е намерила изоморфизъм.

[a] Човек може да помисли, че това е неправилно написано “naughty”.

Следните случаи са изчерпателни. Във всеки от тях откриваме, че G има подграф, хомеоморфен на K_5 или $K_{3,3}$, противно на допускането, че G няма такива подграфи.

Случай 1. $o_1 \neq i_2$ и $o_2 \neq i_1$. Да си представим къде може да се намират i_1 и i_2 спрямо четирите върха x, y, o_1 и o_2: или $i_1 \in y \rightsquigarrow o_2$, или $i_1 \in o_2 \rightsquigarrow x$; или $i_2 \in x \rightsquigarrow o_1$, или $i_2 \in o_1 \rightsquigarrow y$. Сега да си представим тези възможности заедно с възможностите за j_1 и j_2. Всичко това е илюстрирано на следната фигура, като възможностите за i_1 са маркирани в синьо, възможностите за i_2 са маркирани в зелено, възможностите за j_1 са маркирани в оранжево, а възможностите за j_2 са маркирани в сиво.



Лема 20

Поне едното от следните две е вярно.

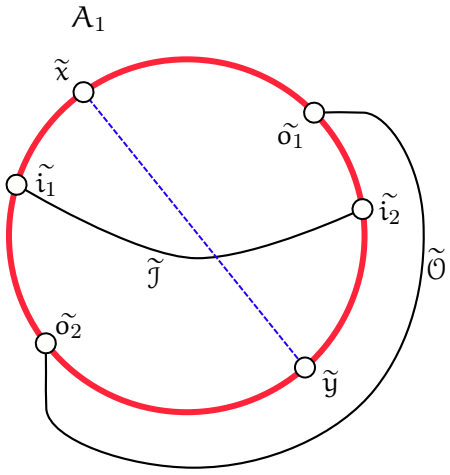
- Има връх от $\{i_1, i_2, j_1, j_2\}$ в $o_2 \rightsquigarrow x$ и има връх от $\{i_1, i_2, j_1, j_2\}$ в $o_1 \rightsquigarrow y$.
- Има връх от $\{i_1, i_2, j_1, j_2\}$ в $y \rightsquigarrow o_2$ и има връх от $\{i_1, i_2, j_1, j_2\}$ в $x \rightsquigarrow o_1$.

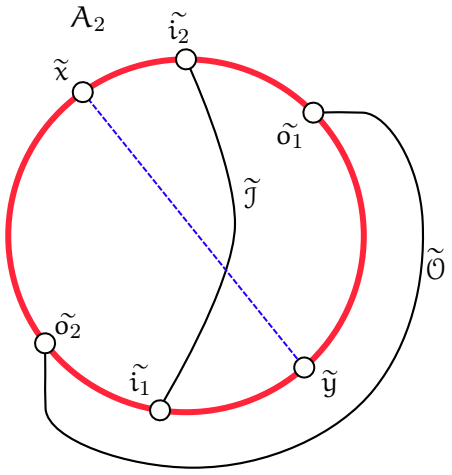
Доказателство: Да допуснем противното. Получаваме "в поне едното от $o_2 \rightsquigarrow x$ и $o_1 \rightsquigarrow y$ няма нито един от i_1, i_2, j_1 и j_2, и освен това в поне едното от $y \rightsquigarrow o_2$ и $x \rightsquigarrow o_1$ няма нито един от i_1, i_2, j_1 и j_2". Това обаче влече, че i_1, i_2, j_1 и j_2 се намират в някое от следните: $o_1 \rightsquigarrow o_2$, $o_2 \rightsquigarrow o_1$, $x \rightsquigarrow y$, $y \rightsquigarrow x$. А това вече е очевидно невъзможно. ✓

Ще разгледаме двете възможности поотделно, а именно:

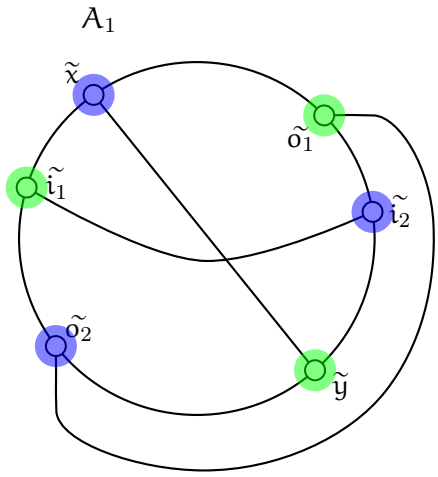
- Връх от $\{i_1, i_2, j_1, j_2\}$, да кажем i_1, да е в $o_2 \rightsquigarrow x$ и друг връх от $\{i_1, i_2, j_1, j_2\}$, да кажем i_2, да в $o_1 \rightsquigarrow y$. Да наречем графа в този подслучай A_1.
- Връх от $\{i_1, i_2, j_1, j_2\}$, да кажем i_1, да е в $x \rightsquigarrow o_1$ и друг връх от $\{i_1, i_2, j_1, j_2\}$, да кажем i_2, да в $y \rightsquigarrow o_2$. Да наречем графа в този подслучай A_2.

Разбира се, $\mathcal{I}$ може да има и други точки на захващане към $\mathcal{C}$, но това не ни интересува. Достатъчно е да разгледаме току-що описаните i_1 и i_2. Да разгледаме A_1 и A_2 поотделно, мислейки за $\widetilde{\mathcal{I}}$ като за планарно ребро в F_2:

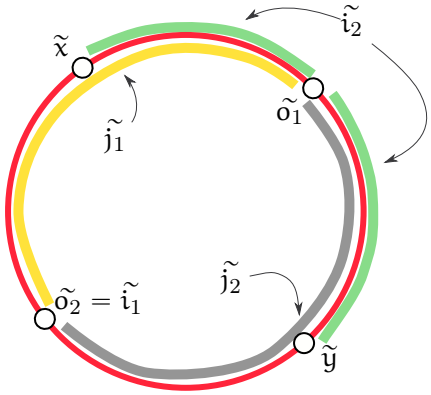




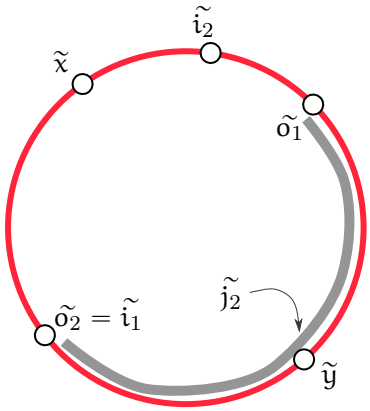
Показаните A_1 и A_2 са изоморфни. Подчертаваме отново, че A_1 и A_2 не са непременно две възможности за **целия** граф G, а—и в двата случая—само на **част** от него, която е същностно важна за нашето доказателство. Забелязваме, че A_1 всъщност е $K_{3,3}$. Следователно, G съдържа подграф, хомеоморфен на $K_{3,3}$, което довършва доказателството в **Случай 1**:



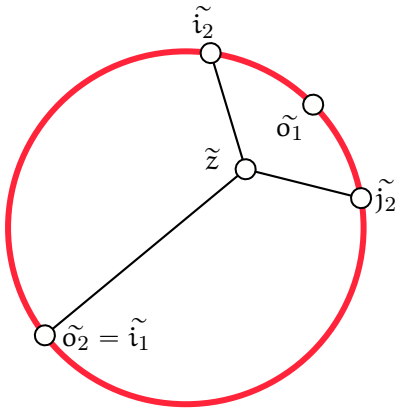
Случай 2. $o_1 \neq i_2$ и $o_2 = i_1$. Следната фигура илюстрира общото положение на върховете, които разглеждаме:



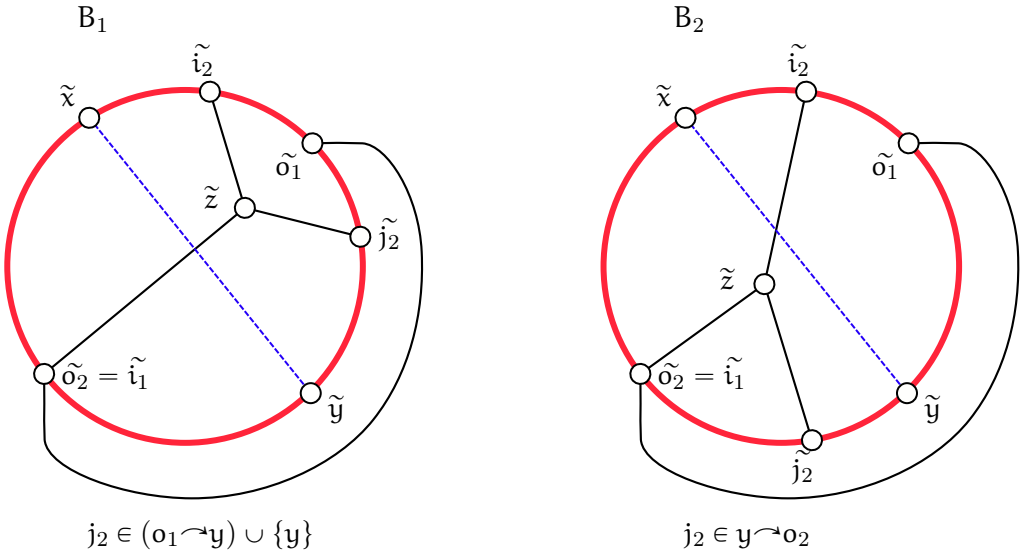
Без ограничение на общността, нека $i_2 \in x \rightharpoonup o_1$. Ще игнорираме j_1 до края на **Случай 2**, така че си представяме това:



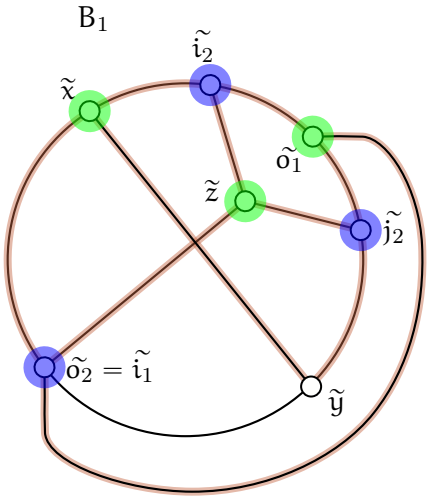
Очевидно, i_1, i_2 и j_2 са три различни върха. Но тогава I е съединение спрямо C, което има поне три различни точки на захващане i_1, i_2 и j_2. Прилагаме Лема 17 и заключаваме, че съществува $v \in \mathrm{inter}(I)$, такъв че има z-i_1 път, z-i_2 път и z-j_2 път, които са независими. Това е показано на следната фигура, като $\widetilde{z}$ е планарният образ на z, а $\widetilde{x}$ и $\widetilde{y}$ не са показани:



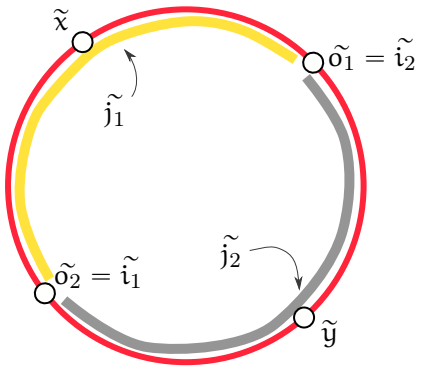
Сега да си представим и $\widetilde{x}$ и $\widetilde{y}$ плюс планарното ребро между тях плюс $\widetilde{\mathcal{O}}$ (показан само като планарно ребро), добавени към последната фигура. За j_2 ще разгледаме два подслучая: $j_2 \in (o_1 \curvearrowright y) \cup \{y\}$ и $j_2 \in y \curvearrowright o_2$. Да наречем съответните графи B_1 и B_2:



B_1 и B_2 са изоморфни. Лесно се вижда, че B_1 има подграф, хомеоморфен на $K_{3,3}$:

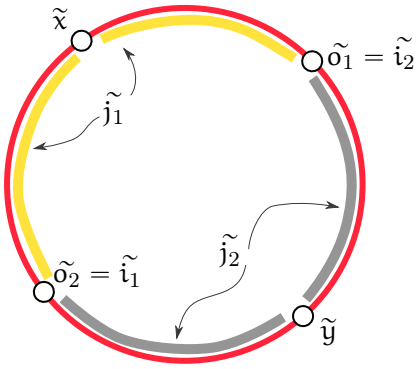


Случай 3. $o_1 = i_2$ и $o_2 = i_1$. Следната фигура илюстрира общото положение на върховете, които имаме предвид:

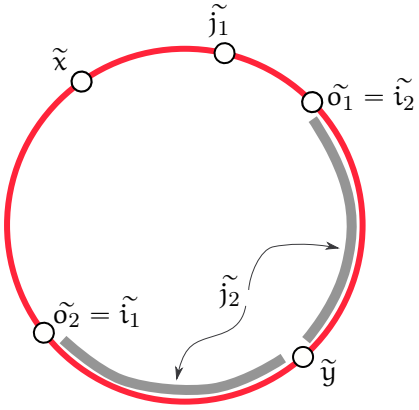


Очевидно i_1, i_2, j_1 и j_2 са четири различни върха. Следователно, I има четири точки на закачане на и съгласно Лема 18, в I има ✦ или ⋉.

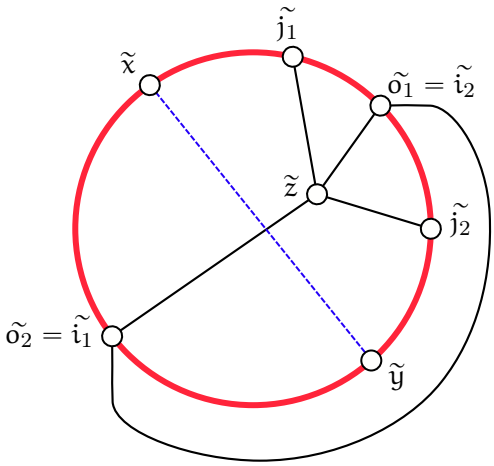
Случай 3.A $\{x, y\} \cap \{j_1, j_2\} = \varnothing$. Следната фигура илюстрира общото положение на върховете, които имаме предвид:



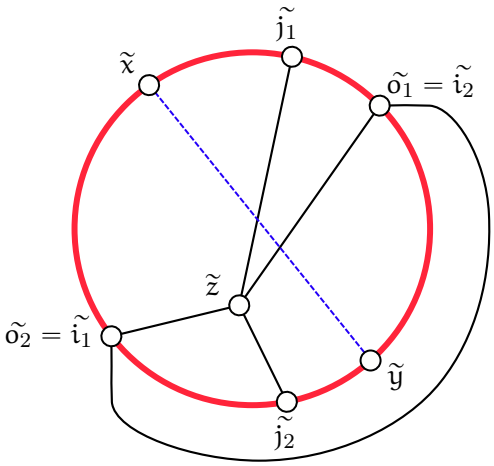
Случай 3.A-1 В I има ✦. Без ограничение на общността, нека $j_1 \in x \curvearrowright o_1$:



Ще разгледаме поотделно два случая: $j_2 \in o_1 \curvearrowright y$ и $j_2 \in y \curvearrowright o_2$:

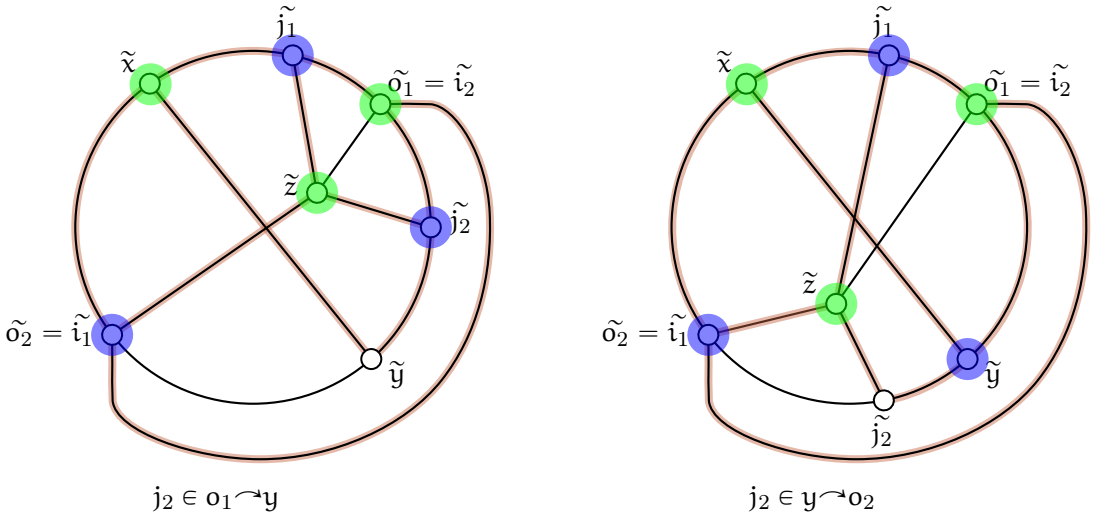


$j_2 \in o_1 \curvearrowright y$

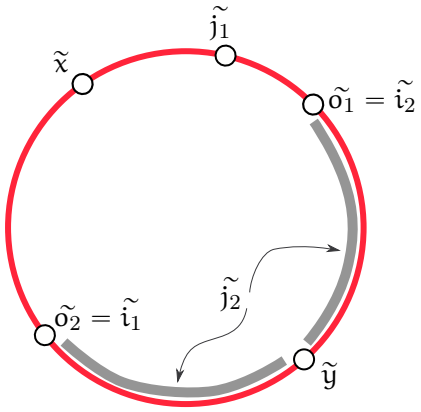


$j_2 \in y \curvearrowright o_2$

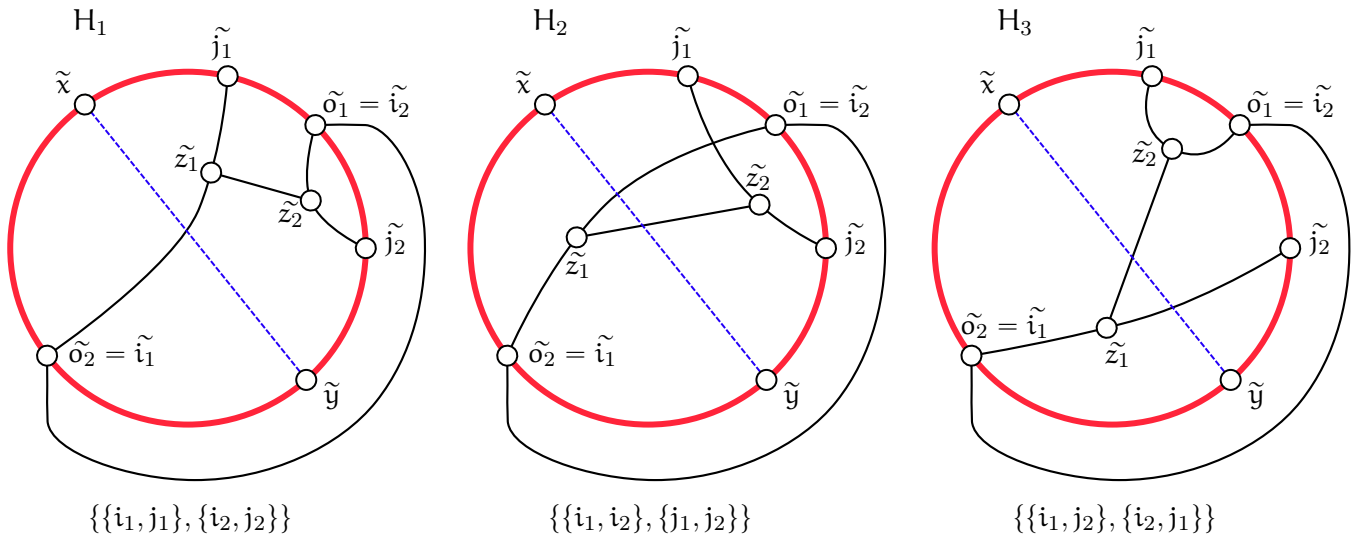
И в двата случая виждаме подграф, хомеоморфен на $K_{3,3}$:



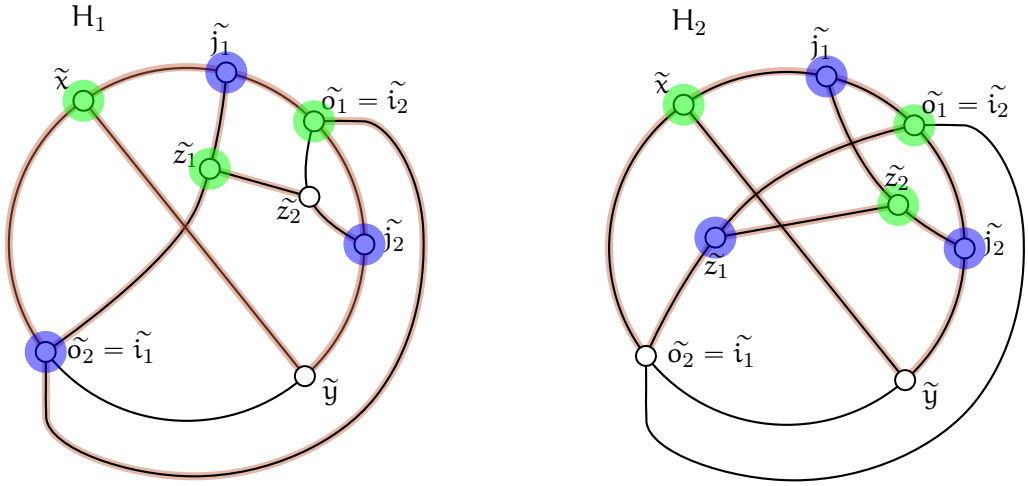
Случай 3.A-2 В I има $\rtimes$. Без ограничение на общността, нека $j_1 \in x \curvearrowright o_1$:



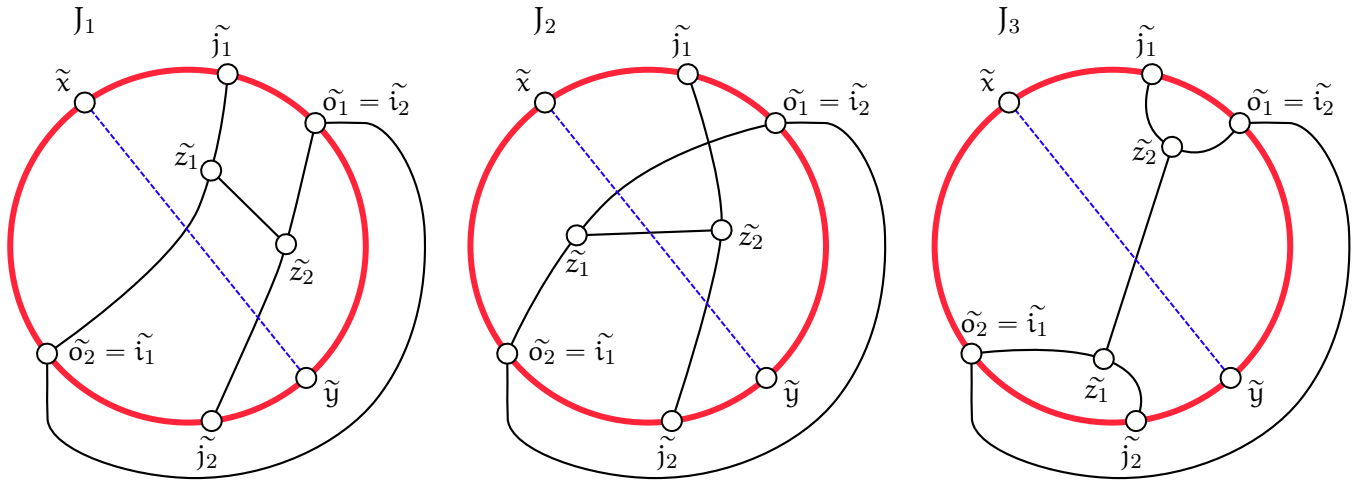
Първо допускаме, че $j_2 \in o_1 \curvearrowright y$. Това обаче не определя еднозначно конфигурацията $\rtimes$. Тя се определя и от разбиването на $\{i_1, i_2, j_1, j_2\}$ на две двуелементни множества. Има точно три такива разбивания, а именно $\{\{i_1, j_1\}, \{i_2, j_2\}\}$, $\{\{i_1, i_2\}, \{j_1, j_2\}\}$ и $\{\{i_1, j_2\}, \{i_2, j_1\}\}$. Да наречем трите съответни графа H_1, H_2 и H_3 и да ги разгледаме поотделно:



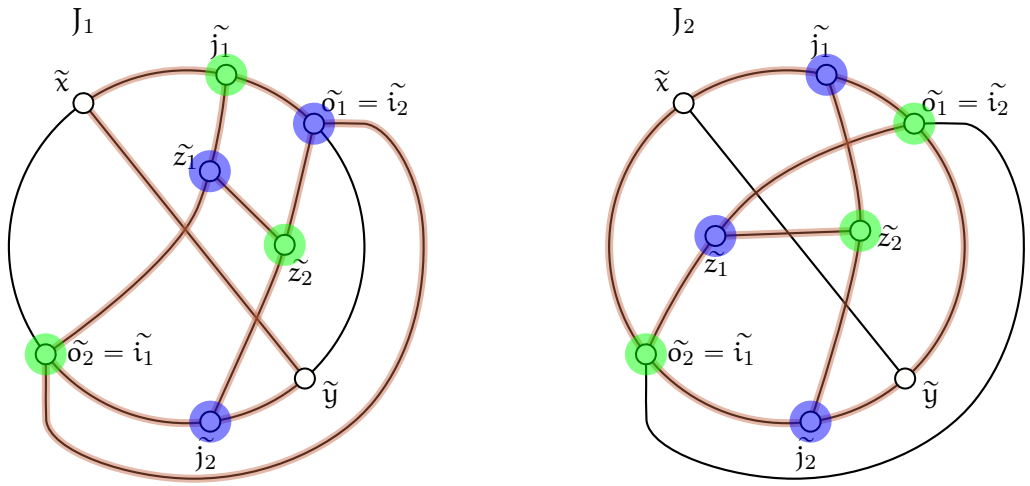
H_1 и H_3 са изоморфни. Ако биекцията на изоморфизма е $\phi : V(H_1) \to V(H_3)$, то $\phi(x) = z_2$, $\phi(o_2) = o_1$, $\phi(y) = j_2$, $\phi(j_2) = x$, $\phi(o_1) = o_2$, $\phi(j_1) = z_1$, $\phi(z_2) = y$, $\phi(z_1) = j_2$. H_1 и H_2 не са изоморфни. По тази причина ограничаваме нашето внимание само до H_1 и H_2. И в двата има подграф, хомеоморфен на $K_{3,3}$:



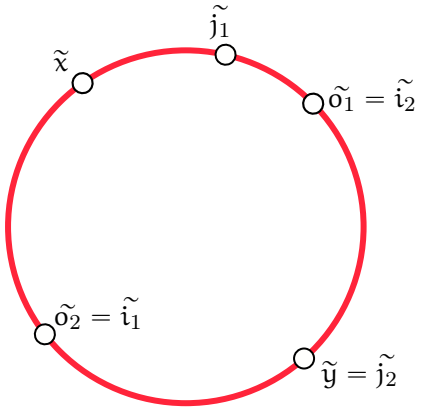
Сега допускаме, че $j_2 \in y \curvearrowright o_2$. Както и преди, конфигурацията $\curlywedge$ се определя и от разбиванията на $\{i_1, i_2, j_1, j_2\}$ на две двуелементни множества, а именно $\{\{i_1, j_1\}, \{i_2, j_2\}\}$, $\{\{i_1, i_2\}, \{j_1, j_2\}\}$ и $\{\{i_1, j_2\}, \{i_2, j_1\}\}$. Да наречем графите, които разглеждаме, съответно J_1, J_2 и J_3 и да ги разгледаме поотделно:



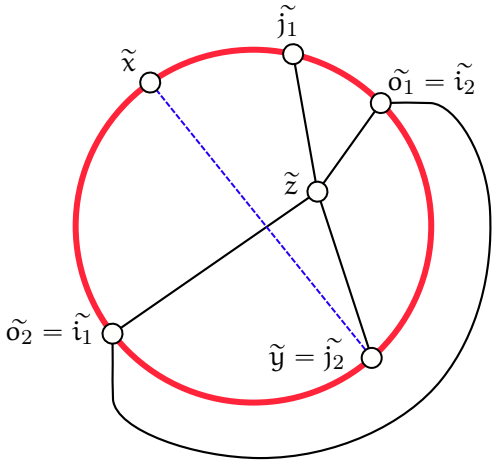
J_3 е изоморфен на вече разгледаните H_1 и H_3. Нито J_1 и J_2 са изоморфни помежду си, нито някой от е изоморфен на J_3 или на някой от H_1, H_2 или H_3. Налага се да разгледаме J_1 и J_2. Във всеки от тях има подграф, хомеоморфен на $K_{3,3}$:



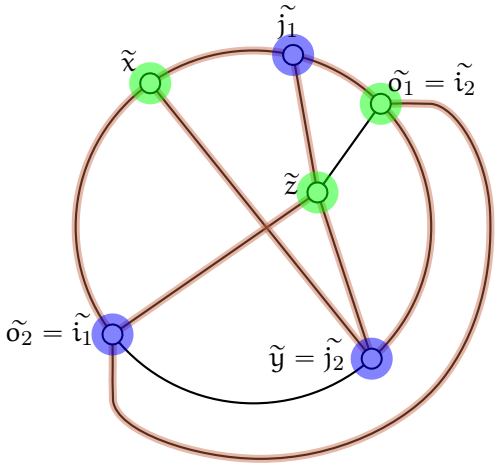
Случай 3.B $|\{x, y\} \cap \{j_1, j_2\}| = 1$. Без ограничение на общността, нека $y = j_2$ и $j_1 \in x \rightsquigarrow o_1$:



Случай 3.B-1 В I има ✦:

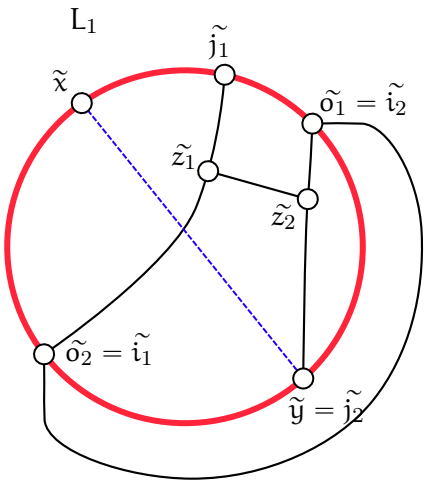


Веднага виждаме подграф, хомеоморфен на $K_{3,3}$:



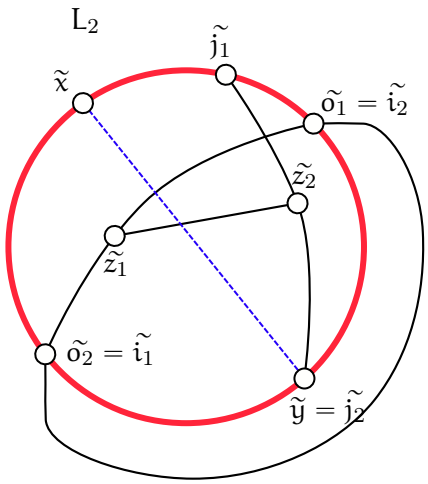
Случай 3.B-2 В I има ☩. Разглеждаме всички разбивания на $\{i_1, i_2, j_1, j_2\}$ на две двуелементни множества, а именно $\{\{i_1, j_1\}, \{i_2, j_2\}\}$, $\{\{i_1, i_2\}, \{j_1, j_2\}\}$ и $\{\{i_1, j_2\}, \{i_2, j_1\}\}$, всяка от които определя еднозначно конфигурацията ☩. Да наречем съответните графи L_1, L_2 и L_3 и да ги разгледаме поотделно:

L_1



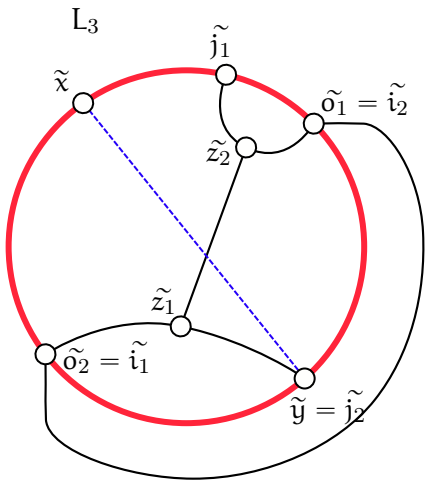
$\{\{i_1, j_1\}, \{i_2, j_2\}\}$

L_2



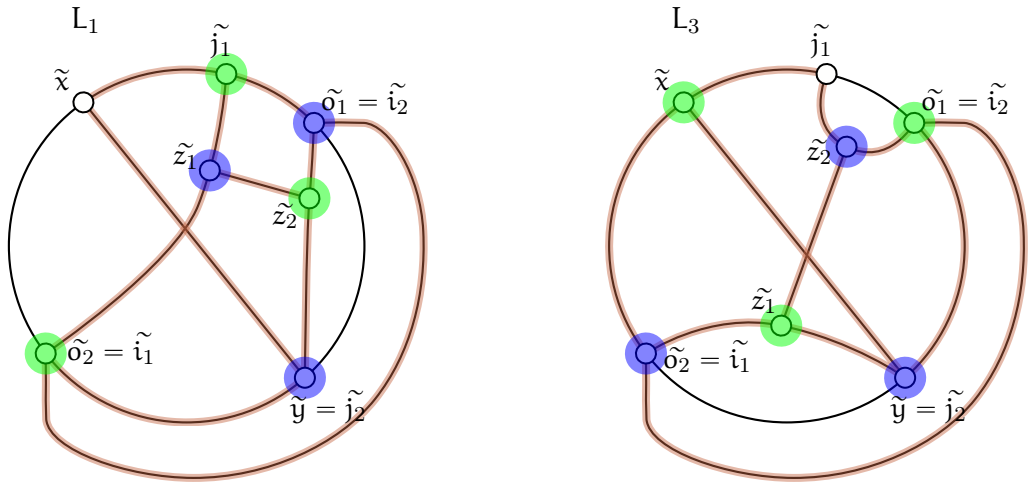
$\{\{i_1, i_2\}, \{j_1, j_2\}\}$

L_3



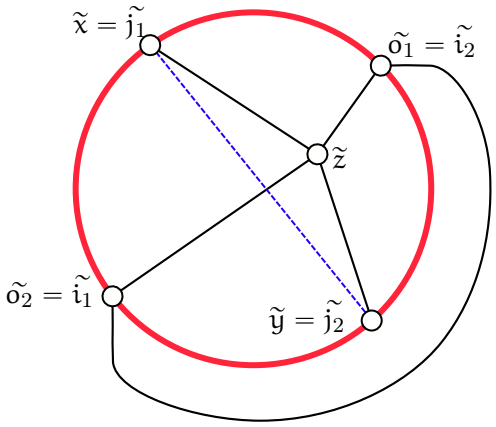
$\{\{i_1, j_2\}, \{i_2, j_1\}\}$

L_1 и L_2 са изоморфни. L_2 не е изоморфен на никой от тях, нито на някой от досега разглежданите графи. Разглеждаме L_1 и L_3. Във всеки от тях има подграф, хомеоморфен на $K_{3,3}$:



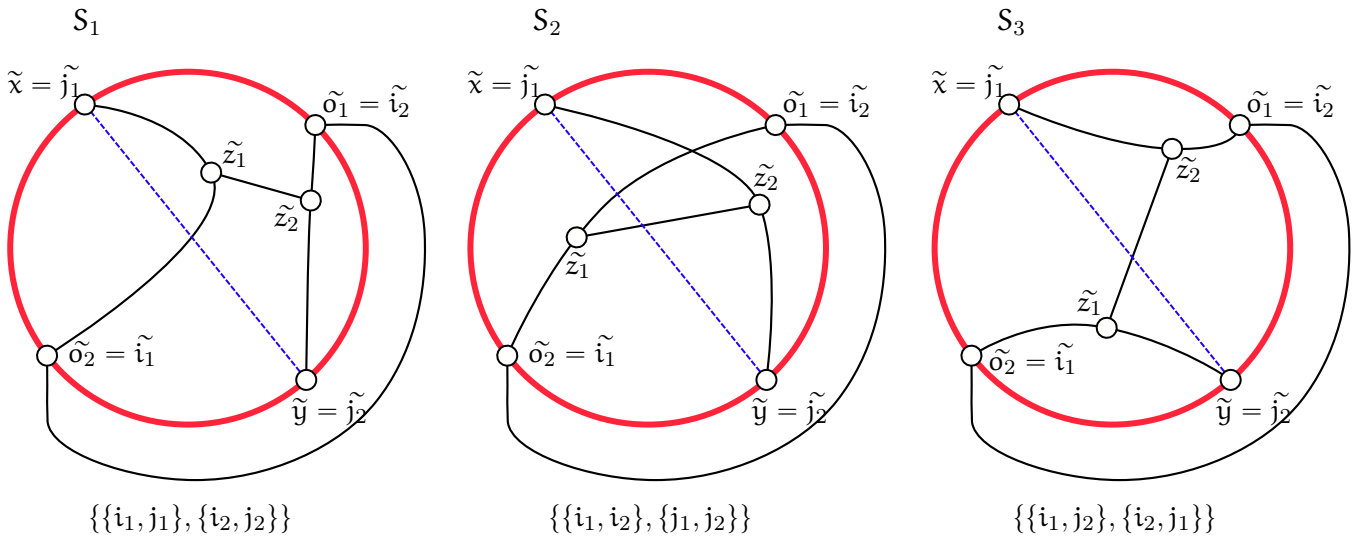
Случай 3.C $\{x, y\} = \{j_1, j_2\}$.

Случай 3.C-1 В I има ✦:

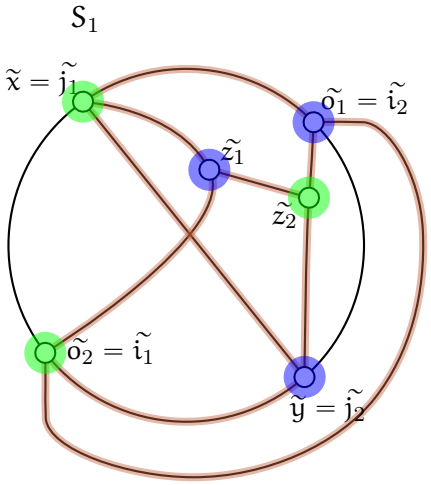


Очевидно този граф е хомеоморфен на K_5. Това е единственият случай в доказателството, в който използваме K_5, а не $K_{3,3}$.

Случай 3.C-2 В I има ✕. Разглеждаме всички разбивания на $\{i_1, i_2, j_1, j_2\}$ на две двуелементни множества, а именно $\{\{i_1, j_1\}, \{i_2, j_2\}\}$, $\{\{i_1, i_2\}, \{j_1, j_2\}\}$ и $\{\{i_1, j_2\}, \{i_2, j_1\}\}$, всяка от които определя еднозначно конфигурацията ✕. Да наречем съответните графи S_1, S_2 и S_3 и да ги разгледаме поотделно:

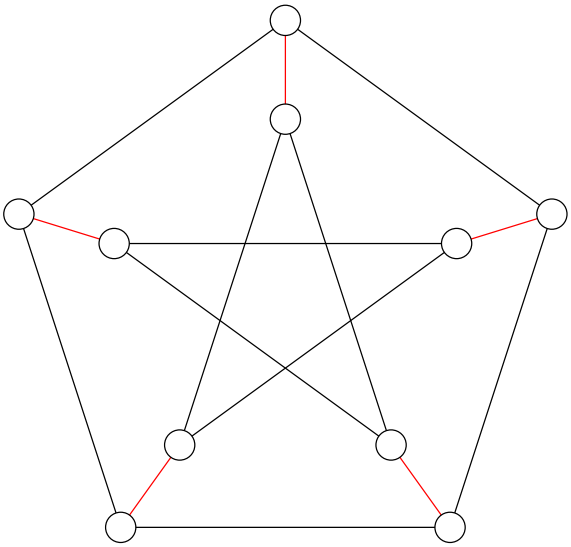


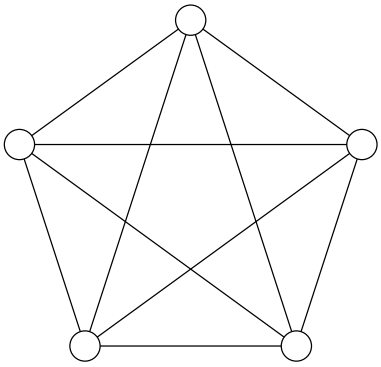
Оказва се, че S_1, S_2 и S_3 са изоморфни. Разглеждаме S_1 и виждаме, че в него има подграф, хомеоморфен на $K_{3,3}$:



Това е краят на доказателството на теоремата на Kuratowski. □

Ще разгледаме пример за прилагането на Теоремата на Kuratowski. Ще докажем, че графът на Petersen не е планарен. Естествено е да се опитаме да докажем това чрез K_5, понеже графът на Petersen съдържа K_5 в себе си по много очевиден начин – ако на следната рисунка на графа на Petersen "колабираме" петте червени ребра, за всяко от тях изтривайки го и идентифицирайки двата му края, ще получим точно K_5:

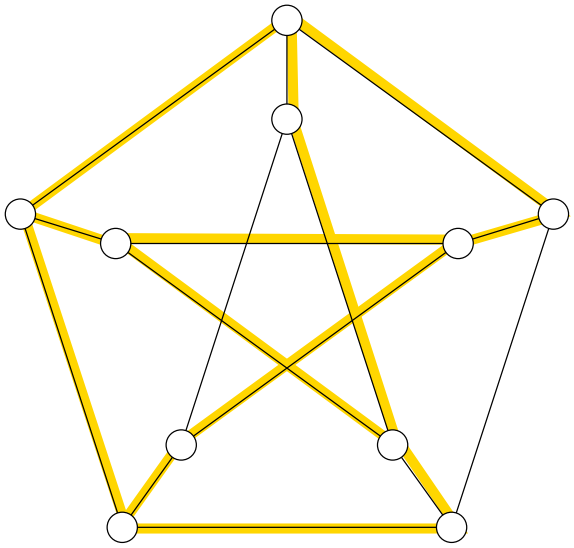




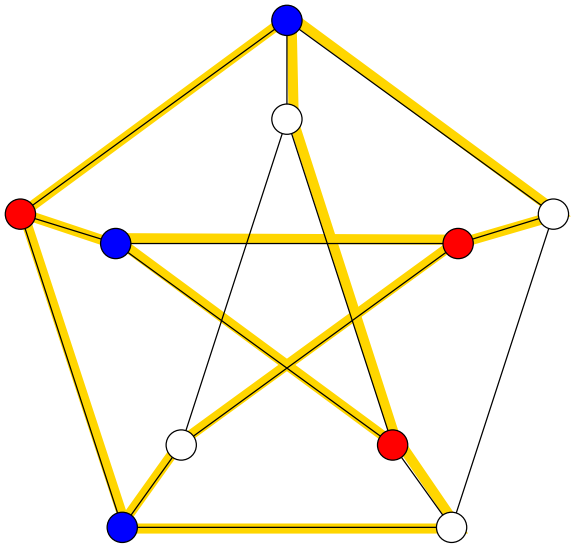
K_5

Но това "колабиране" на ребра не е измежду двете операции, за които става дума в Определение 69. И наистина, графът на Petersen **не съдържа** подграф, хомеоморфен на K_5. За да видим, че е така, да съобразим, графът на Petersen е 3-регулярен, а в K_5 е 4-регулярен; от друга страна, всеки два хомеоморфни графа очевидно имат един и същи брой върхове от степен, различна от 2.

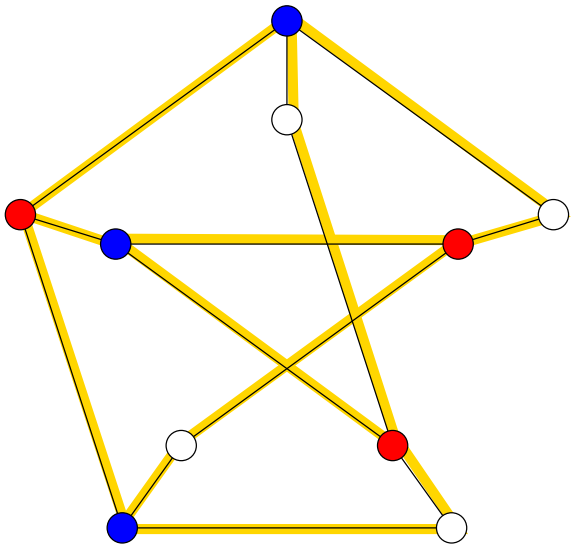
Доказателството с теоремата на Kuratowski, че графът на Petersen не е планарен, използва $\mathsf{K}_{3,3}$, а не K_5. Ще видим, че графът на Petersen съдържа подграф, хомеоморфен на $\mathsf{K}_{3,3}$. На следната рисунка е показа един такъв подграф, нарисуван с жълто върху графа на Petersen.



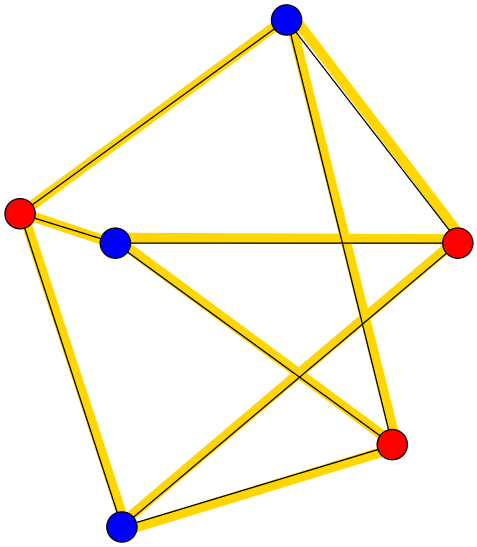
За да се убедим, че жълтият подграф наистина е хомеоморфен на $\mathsf{K}_{3,3}$, първо да съобразим, че жълтият подграф има върхове от (и само от) степени 2 и 3, и да оцветим върховете му от степен 3 с червено и синьо.



Второро, да изтрием тези ребра, които не са жълти, за да остане само въпросният подграф.



Сега да свием всички възможни ребра чрез върхове от степен 2, а това са точно белите върхове.



Вече се вижда ясно, че това е $K_{3,3}$, като върховете от единия му дял са червените върхове, а от другия му дял са сините върхове.

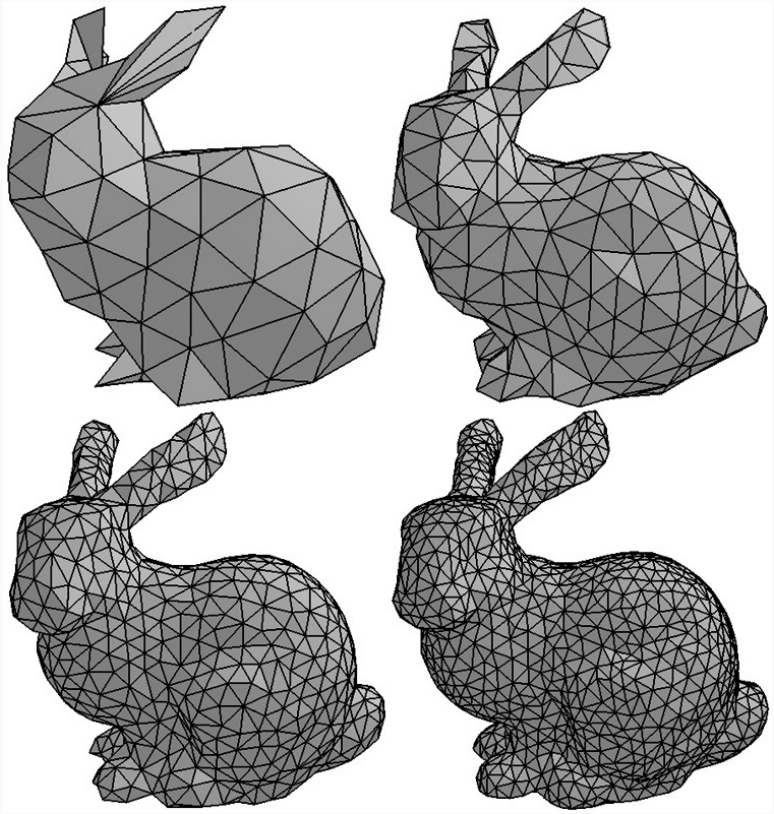
2.13.6 Вписвания на графи в повърхнини и 3D modeling

Съвсем кратко въведение в 3D modeling

Тази подсекция е преход между Секция 2.13 **Планарност на графи** и Секция 2.14 **Вписване на графи в повърхнини от по-висок род**. Ще разгледаме приложение на току-що развитата теория на планарните графи в областта на 3D modeling и ще открием нейните ограничения, разглеждайки примери, в които тя не е приложима.

3D modeling е дял на компютърната графика, който се занимава със създаването на математическо представяне (модел) на даден тримерен обект. Това се постига, като повърхнината на обекта се представя (апроксимира) чрез *мрежа от многоъгълници*, на английски *polygon mesh*, която е множество от точки, отсечки, свързващи някои двойки точки, и многоъгълници, оградени от такива отсечки. На практика най-често многоъгълниците са триъгълници. Като пример за 3D modeling да разгледаме Фигура 2.91. Тя показва няколко тримерни модела на известния *Станфордски заек*: от най-грубия модел горе вляво до най-прецизния модел долу вдясно.

Фигура 2.91 : Станфордския заек.



Повърхнината на тялото се нарича *2-многообразие*, на английски 2-*manifold*, ако за всяка точка, в достатъчно малка околност около точката, повърхнината е "като Евклидовата равнина". По-стриктно определение на "2-многообразие" ще дадем в Секция 2.14. Тук само ще кажем, че 2-многообразие не може да има срезове, цепнатини, дупки към вътрешността, прищипвания и така нататък. Примерно, цилиндърът не е 2-многообразие, докато сферата, кубът (и изобщо всеки многостен), както и тороидът, са 2-многообразия.

Повърхнината на Станфордския заек е 2-многообразие. Нещо повече, Станфордския заек е хомеоморфен със сферата (вижте Определение 71): лесно е да си представим, че заекът

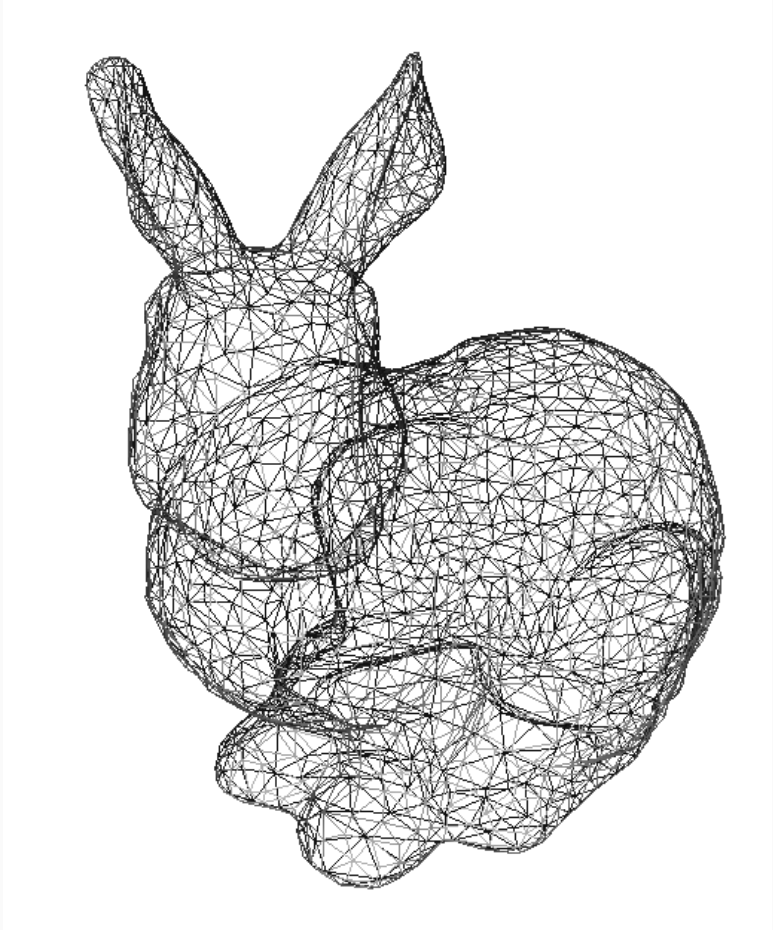
може да бъде трансформиран в сферата по еластичен начин, и обратно, сферата може да бъде трансформирана в заека еластично.

Вече виждаме връзката между мрежата от модела на заека и планарните графи. Ако мрежата представлява многостен, какъвто е случаят с моделите на Фигура 2.91, то, ако игнорираме геометричния аспект на мрежата, тя става комбинаторно планарно вписване на някакъв планарен граф.

Прости примери за мрежи на 3D модели

На сайта на *Maks Ovsjanikov* има свободен за ползване *уеб ресурс* – архив от 3D модели, някои от които са модели на Станфордския заек. За разглеждането на моделите е необходим софтуер като свободния софтуерен пакет *MeshLab*. С MeshLab може да отворите файловете (в директория `PS2/data`) с разширение .off и да разгледате моделите по начин, който е невъзможно да бъде възпроизведен пълноценно в този документ. Отворете в MeshLab файла `bunny_simple.off`. Фигура 2.92 показва заека като wireframe (кликнете `Render -> Render Mode -> Wireframe`).

Фигура 2.92 : Wireframe на Станфордския заек.



Подчертавам, че с MeshLab можете да разгледате модела по несравнимо по-пълноценен начин и да придобиете несравнимо по-ясна представа за него, отколкото от Фигура 2.92.

За да видите данни за мрежата (polygon mesh), кликнете `View -> Show Layer Dialogue` и после `Filters -> Quality Measures and Computations -> Compute Topological Measures`. В моя случай резултатът е:

```
Opened mesh
  /home/minko/Work/lec-graphs/meshes/site1/PS2/data/bunny_simple.off in 105 msec
All files opened in 105 msec
V: 2002 E: 6000 F: 4000
Unreferenced Vertices 0
Boundary Edges 0
Mesh is composed by 1 connected component(s)
Mesh has is two-manifold
Mesh has 0 holes
Genus is 0
```

"Mesh has is two-manifold" казва, че мрежата моделира 2-многообразие. Ако това не беше така, следните разсъждения нямаше да са смислени. "Genus is 0" означава, че 2-многообразието е хомеоморфно на сферата.

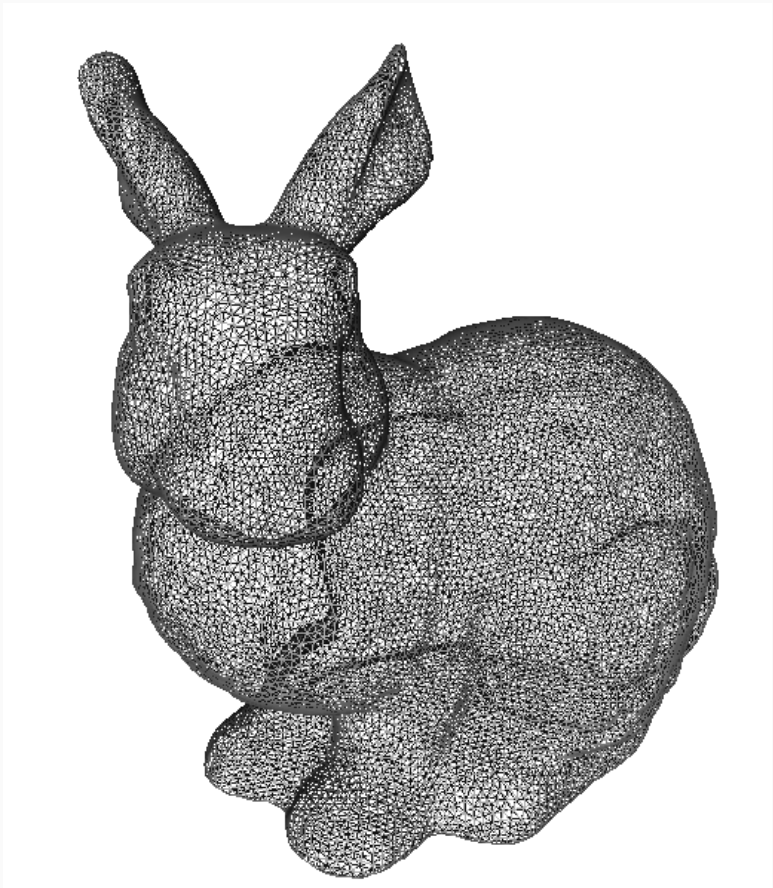
V е броят на върховете на мрежата, E е броят на ребрата, а F е броят на триъгълниците (лицата). В сила е

$$V - E + F = 2002 - 6000 + 4000 = 2$$

Това не е изненада, тъй като мрежата е многостен и съгласно Наблюдение 28 има характеристика 2.

Отворете файла `bunny.off`. Той съдържа мрежа на заека, която е много по-детайлна от мрежата на `bunny_simple.off` и съответно има много повече върхове, ребра и триъгълници. Мрежата е показана на Фигура 2.93.

Фигура 2.93 : Друг wireframe на Станфордския заек.



Данните за тази мрежа са:

```
Opened mesh
  /home/minko/Work/lec-graphs/meshes/site1/PS2/data/bunny.off in 155 msec
All files opened in 155 msec
V: 14290 E: 42864 F: 28576
Unreferenced Vertices 0
Boundary Edges 0
Mesh is composed by 1 connected component(s)
Mesh has is two-manifold
Mesh has 0 holes
Genus is 0
```

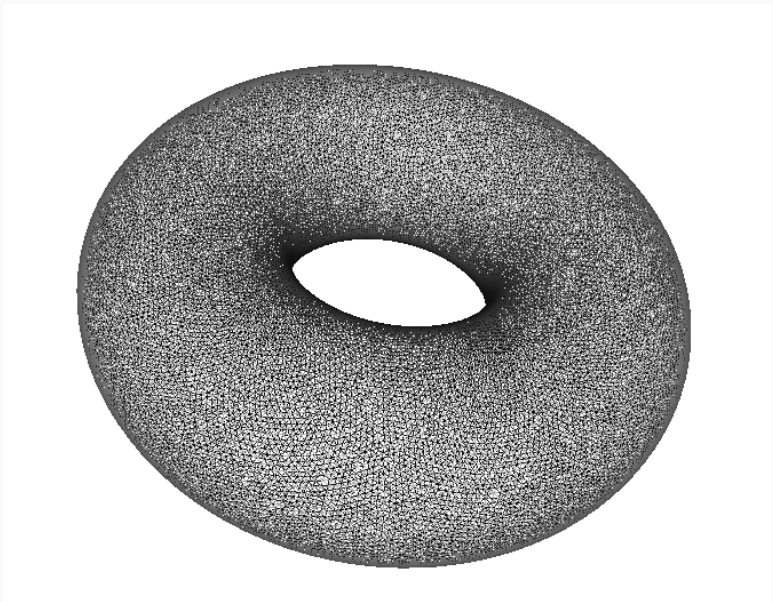
Отново в сила е:

$$\mathsf{V} - \mathsf{E} + \mathsf{F} = 14\,290 - 42\,864 + 28\,576 = 2$$

както очакваме от многостен.

Сега отворете файла `torus.off`. Той съдържа мрежа на тороид, иначе казано "геврек". Мрежата на тороида е показана на Фигура 2.94.

Фигура 2.94 : Wireframe на тороид.



Данните за тази мрежа са:

```
Opened mesh
  /home/minko/Work/lec-graphs/meshes/site1/PS2/data/torus.off in 185 msec
All files opened in 185 msec
V: 24003 E: 72009 F: 48006
Unreferenced Vertices 0
Boundary Edges 0
Mesh is composed by 1 connected component(s)
Mesh has is two-manifold
Mesh has 0 holes
Genus is 1
```

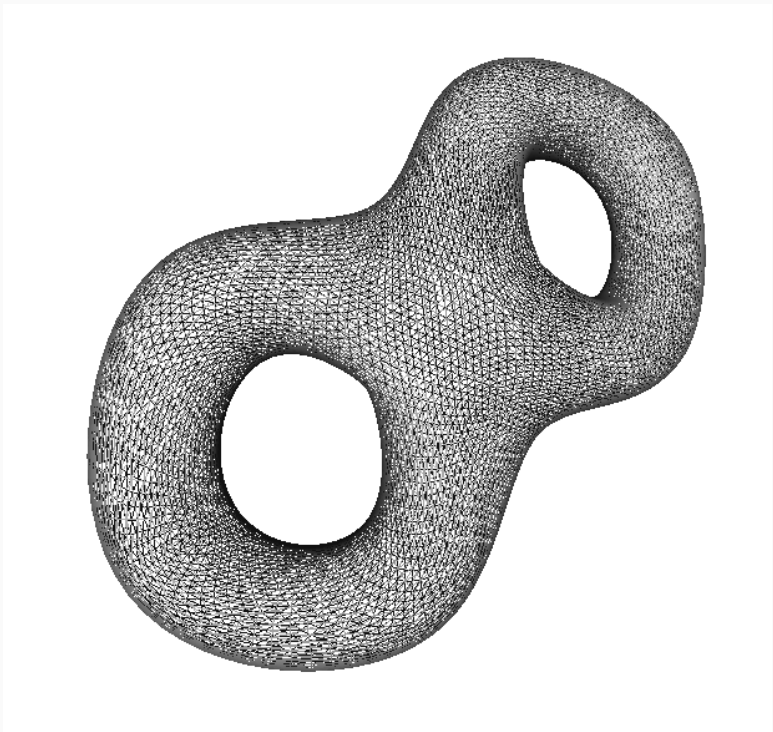
За тази мрежа, в сила е

$$\mathsf{V} - \mathsf{E} + \mathsf{F} = 24\,003 - 72\,009 + 48\,006 = 0$$

Сега вече $\mathsf{V} - \mathsf{E} + \mathsf{F}$ не е две, а **нула**. Причината е, че тороидът има "дупка", поради което е фундаментално различен от многостените и Наблюдение 28 не е в сила за него и обектите, които са му хомеоморфни. "Genus 1" (род едно) е понятие, което ще разгледаме подробно в Секция 2.14. Тук само ще кажем неформално, че повърхнина от род едно е хомеоморфна на сфера с една "залепена дръжка" върху нея. Залепването на дръжка е същото като наличието на дупка като тази на тороида. Тороидът е 2-многообразие, също както и сферата, но заради "дръжката" не е хомеоморфен на сферата. Броят на "дръжките" е родът (genus) на повърхнината[†]. Сферата има нула дръжки и има род нула, тороидът има една дръжка и има род едно, и така нататък.

Сега отворете файла `double_torus2.off`. Той съдържа мрежа на двоен тороид, иначе казано "два залепени геврека". Мрежата на двойния тороид е показана на Фигура 2.95.

Фигура 2.95 : Wireframe на двоен тороид.



Данните за тази мрежа са:

```
Opened mesh
  /home/minko/Work/lec-graphs/meshes/site1/PS2/data/double_torus2.off in 126 msec
All files opened in 127 msec
V: 8595 E: 25791 F: 17194
Unreferenced Vertices 0
```

[†]Това е вярно само за така наречените *ориентируеми повърхнинаи* – всяка такава, ако е 2-многообразие, е хомеоморфна на сфера с някакъв брой дръжки, който е родът на повърхнината е броят на дръжките. Има и 2-многообразия, които не са ориентируеми повърхнинаи и които не се получават от сферата с добавяне на дръжки – например, лентата на Мьобиус или бутилката на Клайн. Повече по този въпрос има в Секция 2.14.

```
Boundary Edges 0
Mesh is composed by 1 connected component(s)
Mesh has is two-manifold
Mesh has 0 holes
Genus is 2
```

За тази мрежа, в сила е

$$V - E + F = 8\,595 - 25\,791 + 17\,194 = -2$$

Сега вече $V - E + F$ е минус две. Двойният тороид е хомеоморфен със сфера с две дръжки и има род (genus) две, като също е 2-многообразие, но не е хомеоморфен нито на сферата, нито на тороида.

Една хипотеза за "сфери с дръжки"

От изложението дотук правим следната хипотеза: добавянето на всяка нова "дръжка" към сферата намалява с точно 2 характеристиката на повърхнината, започвайки от +2 за сферата без дръжки.

Хипотеза 3

За всяка повърхнина P , която е хомеоморфна на сфера с k дръжки, за всяко вписване на граф в P, чиито лица са хомеоморфни с кръга, е изпълнено

$$n - m + f = 2 - 2k$$

където n е броят на върховете, m е броят на ребрата, а f е броят на лицата на вписването.

Очевидно Теорема 32 е частен случай на този резултат за $k = 0$, тоест за сфера без дръжки. Хипотеза 3 е вярна и ще я докажем формално в Секция 2.14.

Сложен пример за 3D мрежа

И накрая да разгледаме една много по-сложна тримерна мрежа. Тя е взета от *този сайт на Georgia Institute of Technology*. Тази мрежа е резултат много прецизно тримерно сканиране на дървена статуетка, наречена Happy Buddha. Тримерният модел се предоставя в два форма́та: .off и .iv. Да разгледаме *.off варианта*. Предупреждение: този модел е много голям, като некомпресирания .ply файл е над 40 мегабайта.

Информация за оригиналната статуетка може да бъде видяна на *тази страница на Stanford University*, където може да разгледате оригинала много подробно и да оцените сложността му. Фигура 2.96 показва снимка на статуетката, взета от станицата на Stanford University.

Фигура 2.96 : Щастлив Буда – статуетка.

Данните за мрежата–резултат от 3D сканирането са:

```
Opened mesh
  /home/minko/Work/lec-graphs/meshes/site4/happy.ply in 1135 msec
All files opened in 1137 msec
V: 543652 E: 1631574 F:1087716
Unreferenced Vertices 0
Boundary Edges 0
Mesh is composed by 1 connected component(s)
Mesh has is two-manifold
Mesh has 0 holes
Genus is 104
```

В сила е

$$\mathsf{V} - \mathsf{E} + \mathsf{F} = 543\,652 - 1\,631\,574 + 1\,087\,716 = -206$$

Съгласно Хипотеза 3, ако повърхнината на статуетката е 2-многообразие—а тя е 2-многообразие съгласно данните от MeshLab, то в сила е $2-2\mathsf{k} = -206$, където k е броят на дръжките. Оттук броят на дръжките е 104, което е точно колкото MeshLab съобщава за рода на повърхнината.

2.14 Вписване на графи в повърхнини от по-висок род

2.15 Графи-хиперкубове

2.16 Съчетания в графи

2.17 Някои видове перфектни графи

2.17.1 Интервални графи

Да разгледаме една задача. Дадено е множество от изпити $X = \{x_1, \ldots, x_n\}$, които трябва да се проведат (присъствено!) в зали на някаква университетска сграда. Всеки изпит x_i се характеризира с времето си на започване s_i и времето си на приключване f_i, където $s_i < f_i$. Прецизно казано, всеки изпит е съответният затворен интервал, тоест, $x_i = \{z \in \mathbb{R} \mid s_i \leqslant z \leqslant f_i\}$. За удобство понякога идентифицираме x_i с наредената двойка (s_i, f_i).

Нека $i \neq j$. Казваме, че изпити x_i и x_j *не се застъпват*, ако $f_i < s_j$ или $f_j < s_i$; в противен случай x_i и x_j *се застъпват*[†]. Искаме да проведем изпитите в колкото е възможно по-малко зали. Допускаме, че всеки изпит може да се проведе в коя да е от залите. Причината да не използваме само една зала може да е само една: има застъпващи се изпити. Да проведем изпитите в минимален брой зали е същото като да разбием множеството от изпитите на минимален брой дялове, като във всеки дял няма застъпващи се изпити.

Дълбочината на X може да се дефинира по два еквивалентни начина

$$\operatorname{depth}(X) \stackrel{\text{def}}{=} \max\{|Y| : (Y \subseteq X) \wedge (\forall x_i, x_j \in Y : i \neq j \to x_i \text{ и } x_j \text{ се застъпват})\}$$

$$\operatorname{depth}(X) \stackrel{\text{def}}{=} \max\{|Y| : ((Y \subseteq X) \wedge \exists z \in \mathbb{R} : z \in \bigcap Y)\}$$

Твърдим, че $\operatorname{depth}(X)$ е необходимият и достатъчен брой зали за провеждането на изпитите. Необходимостта е очевидна. Достатъчността обаче не е напълно очевидна. Един от начините да се покаже е конструктивен е със следния алчен алгоритъм.

1. Сортирай изпитите по начално време. БОО, нека сортираната редица е $\langle x_1, \ldots, x_n \rangle$.
2. Резервирай зала за x_1.
3. За i от 2 до n:
   (а) ако в момент s_i има свободна зала от досега използваните, сложи x_i в нея,
   (б) в противен случай резервирай нова зала—която до момента не е използвана—за x_i.

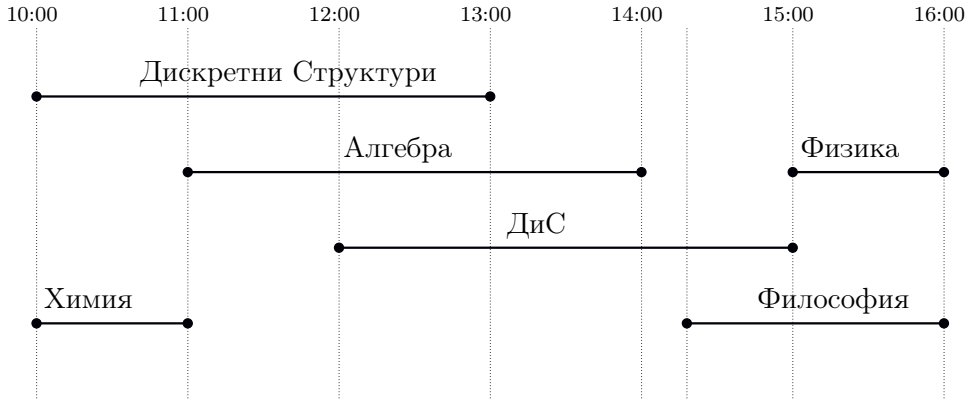
След това доказваме коректността на алгоритъма. Забелязваме, че алгоритъмът резервира $\operatorname{depth}(X)$ на брой зали и това доказваме, че $\operatorname{depth}(X)$ е достатъчен брой зали.

Можем обаче да докажем, че $\operatorname{depth}(X)$ е достатъчен брой зали и по друг начин. Да конструираме обикновен граф $G = (X, E)$, където

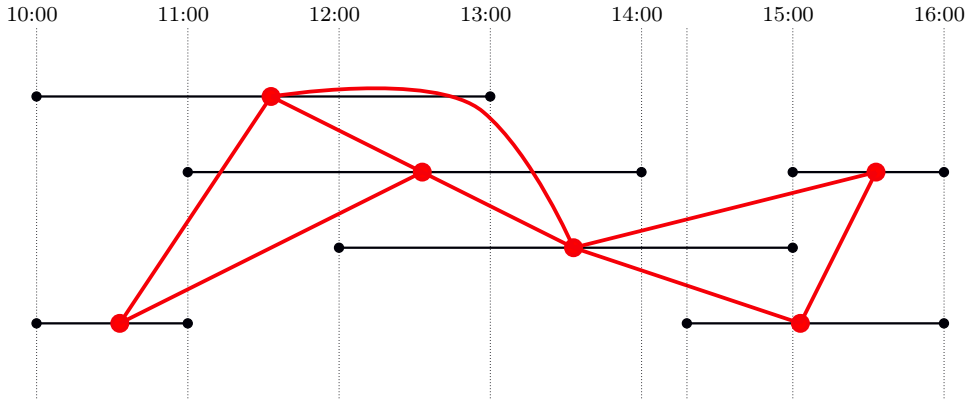
$$E = \{(x_i, x_j) \mid (i \neq j) \wedge (x_i \text{ и } x_j \text{ се застъпват})\}$$

Това е граф, чиито върхове са интервалите, а ребрата отговарят точно на застъпванията. По отношение на изпитите, този граф моделира несъвместимостите. Ето пример за някакви изпити, в който всички изпити започват и завършват на кръгъл час, с изключение на изпита по Философия, който е от 14:15 ч.

[†]Забележете, че според тази дефиниция, ако един изпит завършва в 10:00 часа, а друг изпит започва в 10:00 часа, те се застъпват, но ако вторият започваше в 10:01 ч., те нямаше да се застъпват. Как точно е дефинирано застъпването не е важно. Важното е да е дефинирано недвусмислено и да може да се изчислява бързо дали два изпита се застъпват.



Ето графът на застъпванията (несъвместимостите), нарисуван върху изпитите.



Графите като този си имат име.

Определение 78: Интервален граф

Граф $G = (V, E)$ се нарича *интервален граф*, ако съществува множество $I = \{i_1, \ldots, i_n\}$ от затворени интервала върху реалната ос, такива че съществува биекция $f : V \to I$, такава че

$$\forall u, v \in V : u \neq v \to ((u, v) \in E \leftrightarrow f(u) \cap f(v) \neq \varnothing)$$

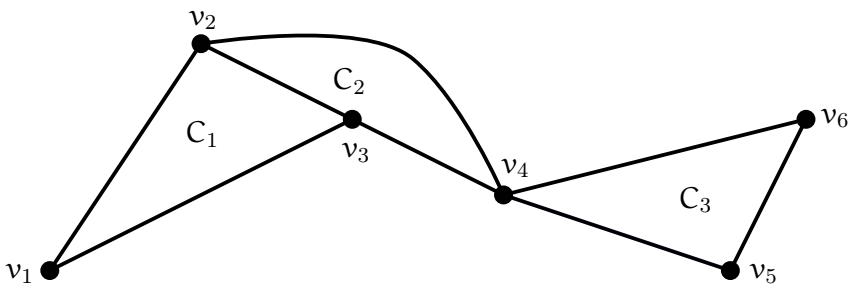
Дали интервалите са затворени и как точно се дефинира застъпването (достатъчно ли е да имат обща точка или трябва да имат обща отсечка) не е съществено.

Оригиналната задача беше да се покаже, че дълбочината на множеството от интервали-изпити е горна граница за броя на залите. Преведено в термините на графа, задачата е: да се покаже, че кликовото число на графа е горна граница за хроматичното му число. Забележете, че разполагането на изпитите в зали по такъв начин, че в нито една зала няма застъпване, е същото като разбиване на множеството от върховете на графа на антиклики, а, както знаем от Подсекция 2.7.1, това е същото като да се оцветят върховете в цветове, съответстващи на антикликите; в случая, това са залите.

Добре известно е [28], че интервалните графи са вид перфектни графи (Определение 36), откъдето веднага следва, че дълбочината на множеството от интервалите е достатъчна за броя на залите за провеждането на изпитите.

Определение 78 взема за върхове не директно интервалите, а някакви абстрактни върхове и ги съпоставя биективно на интервалите. Това е нарочно. Интервалите може да не са дадени, а да е даден само графът и да се пита дали има такова множество от интервали; тоест, за произволен граф може има смисъл да питаме дали е интервален или не. Не всеки граф е интервален. Известно е [28, стр. 182, Theorem 8.1], че интервалните може да се характеризират и по друг начин: граф е интервален тстк максималните му по включване клики може да се наредят линейно по такъв начин, че за всеки връх x, максималните (по включване) клики, съдържащи x, се появяват в непрекъсната подредица (*consecutively* на английски).

Като пример да разгледаме същия граф, но сега нарисуван самостоятелно, с имена на върховете, които нямат нищо общо с имената на изпитите.



Трите максимални по включване клики са C_1, C_2 и C_3. Имайте предвид, че кликите са множества от върхове, а не подграфи, така че $C_1 = \{v_1, v_2, v_3\}$, $C_2 = \{v_2, v_3, v_4\}$ и $C_3 = \{v_4, v_5, v_6\}$. И наистина, кликите може да се наредят линейно по такъв начин, че всеки връх се появява в непрекъсната подредица от клики:

$$\langle \quad C_1 = \{v_1, v_2, v_3\} \qquad C_2 = \{v_2, v_3, v_4\} \qquad C_3 = \{v_4, v_5, v_6\} \quad \rangle$$

Полуформално казано, максималните по включване клики на интервалния граф са "залепени" една за друго по линеен начин (а не, да кажем, по кръгов или по дървовиден или по някакъв по-общ начин). Оттук читателят лесно може да си направи малък пример за граф, който не е интервален.

Интервалните графи са много полезни на практика и възникват като моделиращо средство във важни практически задачи [28].

Допълнение 24: Приложения на интервалните графи

Golumbic в [28] посочва интересни приложения на интервалните графи. Ето някои от тях.

1. Дадени са някакви химически съединения c_1, c_2, ..., c_n. За всяко $i \in \{1, \ldots, n\}$ е дадено, че c_i трябва да се съхранява в температурния интервал $[t'_i, t''_i]$, където t'_i и t''_i са градуси, като $t'_i < t''_i$. Колко хладилника са ни необходими, за да съхраняваме тези съединения? Очевидно се допуска, че всеки хладилник може да побере всички (контейнери със) съединения, и освен това всеки хладилник поддържа само една температура (защото има само едно отделение).

   Ако ползваме n хладилника, нещата стават тривиални: просто слагаме всяко съединение в собствен хладилник.

Има ситуации, в които можем да минем само с един хладилник: да кажем, $\mathfrak{n} = 3$ и интервалите са $[-5°, +10°]$, $[-2°, +20°]$ и $[-15°, +25°]$. Очевидно един хладилник, поддържащ $0°$, е достатъчен. Причината е, че трите интервала имат общо непразно сечение. Ако третият интервал беше $[+11°, +25°]$, един хладилник нямаше да е достатъчен.

Да построим интервалния граф върху тези интервали. Минималният брой необходими хладилници е равен на неговото минимално *число на кликово покриване*. Числото на кликово покриване на произволен граф G (не само интервален) е минималното k, такова че можем да разбием $V(G)$ на k дяла, като всеки от тях е клика. Съединенията, отговарящи на всяка клика, ще държим в отделен хладилник.

2. В зората на генетиката, Benzer [8] казва:

> From the classical researches of Morgan and his school, the chromosome is known as a linear arrangement of hereditary elements, the "genes". These elements must have an internal structure of their own. At this finer level, within the "gene" the question arises again: what is the arrangement of the sub-elements? Specifically, are they linked together in a linear order analogous to the higher level of integration of the genes in the chromosome?
>
> ...
>
> A number of cases have been investigated on this level. As a rule, closely linked mutations affecting the same characteristic can be seriated in an unambiguous way, suggesting a linear model. However, the "distances" (i.e., recombination frequencies) between mutations are not always strictly additive, and certain complexities ("negative interference" effects) make quantitative analysis difficult.
>
> ...
>
> A crucial examination of the question should be made from the point of view of *topology*, since it is a matter of how the parts of the structure are *connected* to each other, rather than of the distances between them. Experiments to explore the topology should ask *qualitative* questions (e.g., do two parts of the structure touch each other or not?) rather than *quantitative* ones (how far apart are they?).

Въпросът, който Benzer поставя, е дали елементите вътре в гена са линейно наредени. За да се отговори на този въпрос се разглеждат данни за мутациите на гена. При определени допускания, мутациите възникват (по отношение на стандартния ген) чрез промени в **свързани** части на вътрешността на гена. Експериментално може да се установи дали променените части на два мутирали гена се пресичат.

Събират се данни за голямо множество мутирали гени и се отчитат данните за пресичанията им. Прави се граф на пресичанията G. Въпросът дали данните за пресичанията са съвместими с хипотезата за линейност на елементите на гена е същият като въпросът дали G е интервален граф. Както посочва Golumbic [28, стр. 172], положителен отговор не потвърждава непременно тази хипотеза, но отрицателен отговор категорично я отхвърля.

3. *Сериализация* (на английски, *seriation*) е опит да се поставят някакви археологически находки в истинския хронологически ред, в който са се появили в миналото.

Дадено е множество от артефакти (по-точно, видове артефакти, а не индивиди-артефакти), намерени в гробове (да кажем, от епохата на Древния Египет). Построяваме граф G, чийто върхове са артефактите, а ребро се слага тстк съответните артефакти са намерени в един и същи гроб. При хипотезата, че всяка двойка артефакти, които са съществували в един и същи момент от времето, се намира заедно в поне един гроб, G е интервален граф, като всяко множество от интервали, което може да е неговото множество от съответни интервали, е потенциален кандидат за сериализация на артефактите.

2.17.2 k-дървета

Глава 3

Други видове графи

3.1 Ориентирани графи и ориентирани мултиграфи

3.1.1 Основни определения

Определение 79: Ориентиран граф

Ориентиран граф е наредена двойка $G = (V, E)$, където V е непразно множество, чиито елементи се наричат *върхове*, E е множество, чиито елементи се наричат *ребра*, като

$$E \subseteq (V \times V) \backslash \{(u, u) \mid u \in V\}$$

На английски ориентиран граф е *directed graph*, което понякога се съкращава до *digraph*. Някои автори като Gibbons [27] не ползват термина "oriented" изобщо. Други автори, например Diestel, ползват и "directed", и "oriented", но с различен смисъл. Според Diestel [18, стр. 28], oriented graph е ориентиран граф, който се получава от неориентиран граф (Определение 1) чрез даване на *ориентация* на всяко неориентирано ребро; това означава, че заменяме всяко неориентирано ребро $\{u, v\}$, което е двуелементно множество[†], с точно една от наредените двойки (u, v) или (v, u). Съгласно това определение, oriented graph е частен случай на directed graph – разликата е в това, че в oriented graphs не може да има и двете ребра (u, v) и (v, u), докато в directed graphs нищо не пречи да ги има и двете.

Конвенция 10

Ако кажем само "граф", разбираме неориентиран граф. За да кажем, че имаме предвид ориентиран граф, трябва да кажем експлицитно "ориентиран".

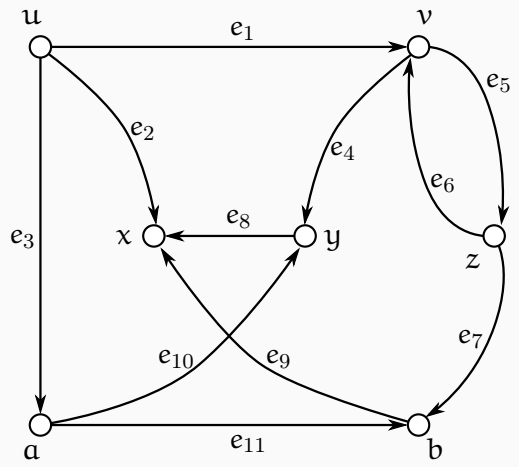
Изключение от това правило ще правим само в случаите, когато от контекста е напълно ясно, че говорим за ориентирани графи.

Обикновено върховете се записват с малки латински букви като u, v и т. н. Имената на ребрата обикновено се записват като e_1, e_2 и т. н. Тъй като всяко ребро е наредена двойка от различни върхове, записваме ребрата като, например, "$e_1 = (u, v)$".

Пример за ориентиран граф е $G = (\{u, v, x, y, z, a, b\}, \{e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8, e_9, e_{10}, e_{11}\})$, където $e_1 = (u, v)$, $e_2 = (u, x)$, $e_3 = (u, a)$, $e_4 = (v, y)$, $e_5 = (v, z)$, $e_6 = (z, v)$, $e_7 = (z, b)$, $e_8 = (y, x)$, $e_9 = (b, x)$, $e_{10} = (a, y)$ и $e_{11} = (a, b)$. Графът G е нарисуван на Фигура 3.1. Ребрата на ориентираните графи по правило се рисуват със стрелки, като стрелката, съответна на реброто (u, v), има посока от u към v. Отново подчертаваме, че "граф" е теоретико-множествено понятие и че граф и негова рисунка са принципно различни неща; това е в сила за ориентираните графи по същия начин, по който е в сила за неориентираните.

[†]Да си припомним: при неориентираните графи, въпреки че пишем това ребро като "(u, v)", имаме предвид "$\{u, v\}$".

Фигура 3.1 : Рисунка на ориентиран граф.



Възможно е ребрата да не бъдат именувани явно и тогава пишем

$$G = (\{u, v, x, y, z, a, b\}, \{(u, v), (u, x), (u, a), (v, y), (v, z), (z, v), (z, b), (y, x), (b, x), (a, y)\})$$

3.1.2 Родители и деца, входна и изходна степен

Определение 80

Нека G е ориентиран граф. Ако $e = (u, v)$ е ребро в G, казваме, че u *е родител на* v, а v *е дете на* u. Също така казваме, че u *е началото на* e и v *е краят на* e.

Като пример да разгледаме графа G на Фигура 3.1. u е родител на a и a е дете на u. v е родител на z и z е дете на v, но също така z е родител на v и v е дете на z.

Определение 81

Нека $G = (V, E)$ е ориентиран граф. За всеки връх $u \in V$, *входната степен на* u е $|\{e \in E \mid u \text{ е край на } e\}|$, а *изходната степен на* u е $|\{e \in E \mid u \text{ е начало на } e\}|$.

На английски "входна степен" е *indegree*, а "изходна степен" е *outdegree*. Ние ще бележим входната степен на u с "$d^-(u)$", а изходната степен на u с "$d^+(u)$" [27, стр. 6].

Неформално казано, входната степен на u е броят на ребрата, с които "влизат" в u, а изходната степен е броят на ребрата, които "излизат" от u. Като пример да разгледаме пак графа G на Фигура 3.1. Вярно е, че $d^-(u) = 0$, $d^+(u) = 3$, $d^-(z) = 1$, $d^+(z) = 2$ и така нататък.

Определение 82

Нека $G = (V, E)$ е ориентиран граф. За всеки връх $u \in V$, такъв че $d^-(u) = 0$ казваме, че u е *източник*. За всеки връх $u \in V$, такъв че $d^+(u) = 0$ казваме, че u е *сифон*.

На английски "източник" е *source*, а "сифон" е *sink*. В примерния граф на Фигура 3.1, u е източник, а x е сифон.

Следната лема е аналог на Лема 1 от неориентираните графи.

Лема 21

Нека $G = (V, E)$ е ориентиран граф. Тогава

$$\sum_{u \in V} d^+(u) = \sum_{u \in V} d^-(u) = |E|$$

3.1.3 Ориентирани графи с примки

Определение 83: Ориентиран граф с възможни примки

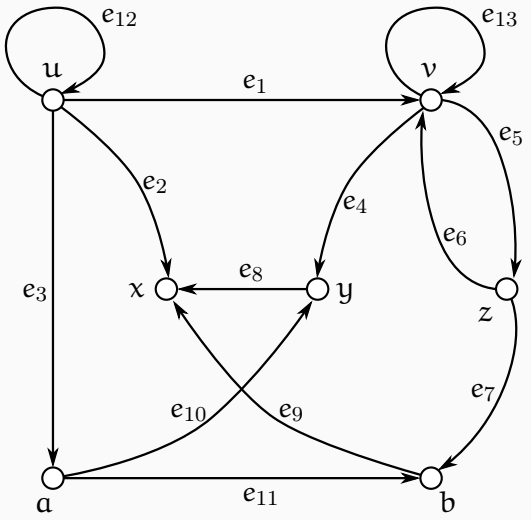
Ориентиран граф с възможни примки е наредена двойка $G = (V, E)$, където V е непразно множество, чиито елементи се наричат *върхове*, E е множество, чиито елементи се наричат *ребра*, като

$$E \subseteq V \times V$$

Този вид графи точно съответстват на релации над краен декартов квадрат. Забележете, че ориентираните графи без примки точно съответстват на антирефлексивните релации.

Пример за ориентиран граф с две примки е $G = (\{u, v, x, y, z, a, b\}, \{e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8, e_9, e_{10}, e_{11}, e_{12}, e_{13}\})$, където $e_1 = (u, v)$, $e_2 = (u, x)$, $e_3 = (u, a)$, $e_4 = (v, y)$, $e_5 = (v, z)$, $e_6 = (z, v)$, $e_7 = (z, b)$, $e_8 = (y, x)$, $e_9 = (b, x)$, $e_{10} = (a, y)$, $e_{11} = (a, b)$, $e_{12} = (u, u)$ и $e_{13} = (v, v)$. Примките са e_{11} и e_{12}. Графът G е нарисуван на Фигура 3.2. Примките се рисуват със стрелки.

Фигура 3.2 : Рисунка на ориентиран граф с примки.



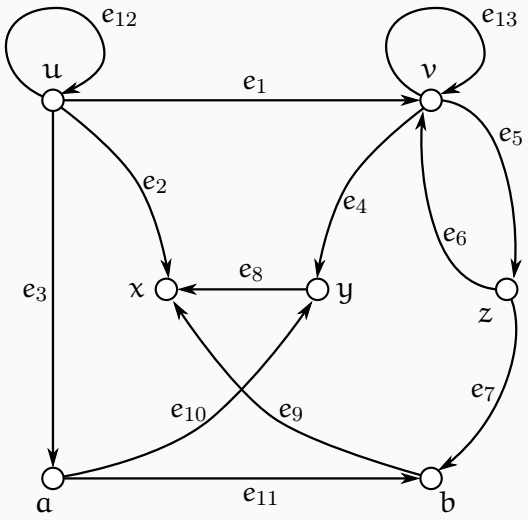
Определение 84: Съответен неориентиран граф

За всеки ориентиран граф, без значение дали може или не може да има примки, *съответният неориентиран граф*[a] е неориентираният граф, който се получава след премахването на ориентацията на ребрата. При това, ако в ориентирания граф има двойка ребра (u, v) и (v, u), тя се заменя с едно единствено неориентирано ребро – иначе бихме получили мултиграф.

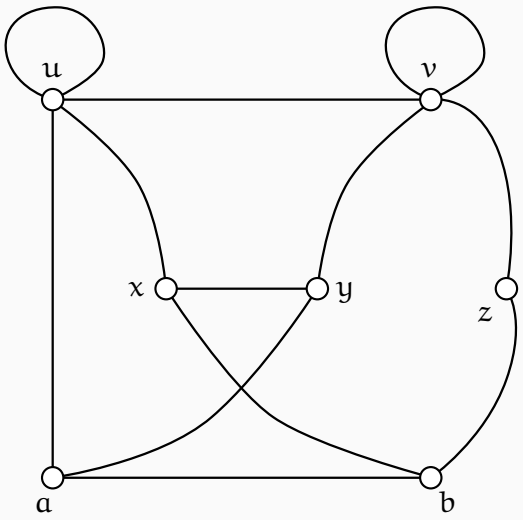
[a] На английски терминът е *the underlying undirected graph of a directed graph*.

Тази дефиниция не е особено формална, защото не ползва теоретико-множествения език, а по-скоро има предвид рисунката на графа. Формално прецизна дефиниция на "съответния неориентиран граф" може да се направи, ако гледаме на ориентирания граф G като на релация, направим нейното симетрично затваряне G' и след това заменим всяка двойка ориентирани ребра (u, v) и (v, u), за различни u и v, с едно единствено неориентирано ребро (u, v). Ако G има примки, те не биват засегнати от симетричното затваряне, така че в резултатния неориентиран граф тези примки остават. Фигура 3.3 изобразява ориентирания граф от Фигура 3.2 и съответния му неориентиран граф.

Фигура 3.3 : Ориентиран граф и съответния му неориентиран граф.



графът от Фигура 3.2



съответният неориентиран граф

Забележете, че докато ребрата в ориентирания граф са показани със свои имена като e_1, e_2 и така нататък, ребрата в съответния неориентиран граф са показани без имена; тоест, всяко ориентирано ребро просто се идентифицира с наредената двойка от началото си и края си. Върховете обаче запазват имената си при преминаването към съответния ориентиран граф.

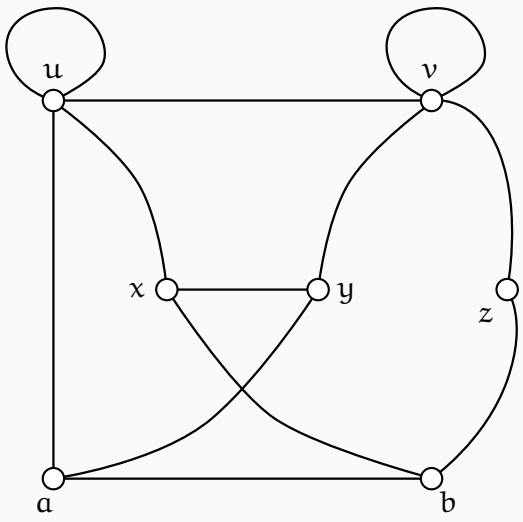
Съответствието между ориентирани и неориентирани графи може да бъде и в другата посока – от неориентирани към ориентирани.

Определение 85: Съответен ориентиран граф

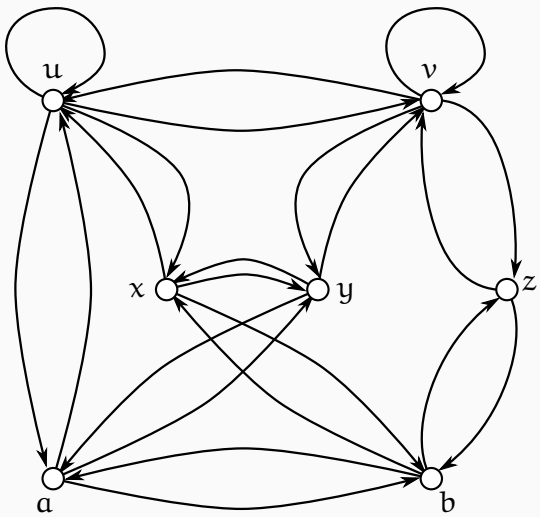
За всеки неориентиран граф, без значение дали може или не може да има примки, *съответният ориентиран граф* е ориентираният граф, който се получава след замяната на всяко неориентирано ребро (u, v) с двете ориентирани ребра (u, v) и (v, u), ако $u \neq v$; ако $u = v$, заменяме неориентираната примка, асоциирана с u, с ориентирана примка (u, u).

Фигура 3.4 изобразява неориентирания граф от Фигура 3.3 и съответния му ориентиран граф. Забележете, че при това **не получаваме** началния ориентиран граф от Фигура 3.2.

Фигура 3.4 : Неориентираният граф от Фигура 3.3 и съответния му ор. граф.



неор. граф от Фигура 3.3



съответният ориентиран граф

3.1.4 Подграфи и индуцирани подграфи. Изоморфизъм.

Понятията "подграф", "подграф, индуциран от подмножество на върховете" и "покриващ подграф" при ориентираните графи се дефинират напълно аналогично на начина, по който се дефинират в неориентираните графи (Определения 7, 8 и 10).

Изоморфизъм между ориентирани подграфи се дефинира напълно аналогично на начина, по който се дефинира при неориентираните графи с Определение 39. Ако G' и G'' в Определение 39 са ориентирани графи с възможни примки, то това определение ни дава точно това, което искаме: графите са изоморфни тстк можем да "нахлупим" единия върху другия, съблюдавайки посоките на ребрата. Имайте предвид, че ако G' и G'' са ориентирани, то (u, v) и $(\phi(u), \phi(v))$ са наредени двойки.

3.1.5 Ориентирани мултиграфи.

Сравнете Определение 86 с Определение 13 и Определение 14.

Определение 86: Ориентиран мултиграф

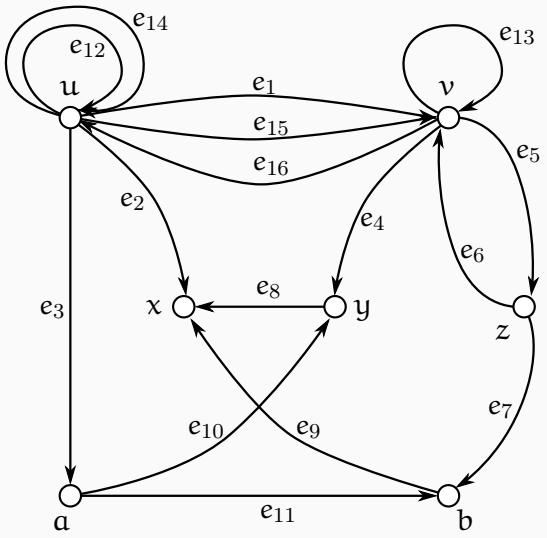
Ориентиран мултиграф е наредена тройка $G = (V, E, f_G)$, където V е непразно множество, чиито елементи се наричат *върхове*, E е множество, чиито елементи се наричат *ребра*, $V \cap E = \varnothing$ и

$$f_G : E \to V \times V$$

е *свързващата функция*.

Забележете, че това определение позволява примки. Когато кажем "ориентиран мултиграф", подразбираме, че той може да има примки. Фигура 3.5 съдържа рисунка на ориентиран мултиграф.

Фигура 3.5 : Рисунка на ориентиран мултиграф.



Мултиграфът, нарисуван на Фигура 3.5, е $G = (\{u, v, x, y, z, a, b\}, \{e_1, \ldots, e_{16}\}, f_G)$, където

$$\begin{array}{llll} f_G(e_1) = (u,v), & f_G(e_2) = (u,x), & f_G(e_3) = (u,a), & f_G(e_1) = (v,y), \\ f_G(e_5) = (v,z), & f_G(e_6) = (z,v), & f_G(e_7) = (z,b), & f_G(e_8) = (y,x), \\ f_G(e_9) = (b,x), & f_G(e_{10}) = (a,y), & f_G(e_{11}) = (a,b), & f_G(e_{12}) = (u,u), \\ f_G(e_{13}) = (v,v), & f_G(e_{14}) = (u,u), & f_G(e_{15}) = (u,v), & f_G(e_{16}) = (v,u) \end{array}$$

При ориентираните мултиграфи също можем да говорим за съответен неориентиран мултиграф и, обратно, ако е даден неориентиран мултиграф, може да говорим за съответен ориентиран мултиграф. Би трябвало да е очевидно как да обобщим Определения 84 и 85 за мултиграфи.

3.1.6 Ориентирани пътища и цикли. Дагове.

Навсякъде в тази подсекция $G = (V, E, f_G)$ е ориентиран мултиграф. Определенията, които ще направим за ориентиран мултиграф, съвсем естествено остават в сила за ориентиран граф, с или без примки.

Забележете приликите и разликите между понятията в тази подсекция и понятията в Секция 2.3. Отново може да си мислим за някакво същество, което живее във върховете на ориентирания мултиграфграф и може да се придвижва, като минава от връх във връх-дете. Последното влече, че трябва да има ориентирано ребро с начало първия връх и край втория връх. Посоките на ребрата **имат значение** и придвижване чрез дадено ребро може да стане **само ако се спазва посоката**. Това е и основната разлика със Секция 2.3, в която за посока не ставаше дума.

Определение 87: Ориентиран път.

Ориентиран път в G наричаме всяка алтерниращa редица от върхове и ребра, за някое $t \geqslant 0$:

$$p = (u_{i_0}, e_{k_0}, u_{i_1}, e_{k_1}, u_{i_2}, \ldots, u_{i_{t-1}}, e_{k_{t-1}}, u_{i_t})$$

където $u_{i_p} \in V$ за $0 \leqslant p \leqslant t$, $e_{k_p} \in E$ за $0 \leqslant p \leqslant t-1$, и освен това е изпълнено $f_G(e_{k_p}) = (u_{i_p}, u_{i_{p+1}})$ за $0 \leqslant p \leqslant t-1$. Връх u_{i_0} се нарича *начало на пътя*, а връх u_{i_t} се нарича *край на пътя*. Останалите върхове са *вътрешните върхове на пътя*. Още казваме, че p е път *от* u_{i_0} *до* u_{i_t}.
Дължината на пътя е броят на ребрата в него. Ще бележим дължината на пътя с $|p|$. В случая, $|p| = t$.
Ако всички елементи на пътя—върхове и ребра—са уникални, казваме, че p е *прост ориентиран път*.

Сравнете израза "p е път между u_{i_0} и u_{i_t}" от Определение 16 с израза "p е път от u_{i_0} до u_{i_t}" от Определение 87. Предлогът "между" изразява симетричност: със същия успех можеше да кажем "p е път между u_{i_t} и u_{i_0}" в Определение 16. От друга страна, двата предлога "от . . . до" изразяват несиметричност: може да няма ориентиран път от u_{i_t} до u_{i_0} в Определение 87, така че в общия случай "p е ориентиран път от u_{i_0} до u_{i_t}" не може да бъде заменено с "p е ориентиран път от u_{i_t} до u_{i_0}".

Определение 88: Ориентиран цикъл.

Нека p е ориентиран път в G, където:

$$p = (u_{i_0}, e_{k_0}, u_{i_1}, e_{k_1}, u_{i_2}, \ldots, u_{i_{t-1}}, e_{k_{t-1}}, u_{i_t})$$

Казваме, че p е *ориентиран цикъл*, ако $u_{i_0} = u_{i_t}$. Казваме, че p е *прост ориентиран цикъл*, ако p е ориентиран цикъл с поне едно ребро и освен това, всички елементи освен $u_{i_0} = u_{i_t}$ са уникални.

Конвенция 11

Ако от контекста е ясно, че става дума за ориентиран граф, няма смисъл да казваме "ориентиран път" или "ориентиран цикъл". Ще казваме само "път" или "цикъл", а това, че е ориентиран, се подразбира от това, че графът или мултиграфът е ориентиран.

Конвенция 12

Простите ориентирани пътища цикли се ползват по-често от другия вид. Затова, отсега нататък, като кажем "ориентиран път" или "ориентиран цикъл", разбираме съответно "прост ориентиран път" или "прост ориентиран цикъл". Ако имаме предвид ориентиран път или ориентиран цикъл, който не е прост, трябва изрично да спомена, че не е прост.

Въведохме понятията "Хамилтонов цикъл" и "Хамилтонов път" (Определение 41) в контекста на неориентираните графи. Тези понятия се пренасят буквално в ориентираните графи.

Определение 89: Ориентиран Хамилтонов цикъл и ориентиран Хамилтонов път

Нека е даден ориентиран граф G. *Ориентиран Хамилтонов цикъл* в G е всеки цикъл в G, който съдържа всички върхове на G. *Ориентиран Хамилтонов път* в G е всеки път в G, който съдържа всички върхове на G.

Ако е ясно, че G е ориентиран, може да казваме само "Хамилтонов цикъл" и "Хамилтонов път", имайки предвид съответно "ориентиран Хамилтонов цикъл" и "ориентиран Хамилтонов път".

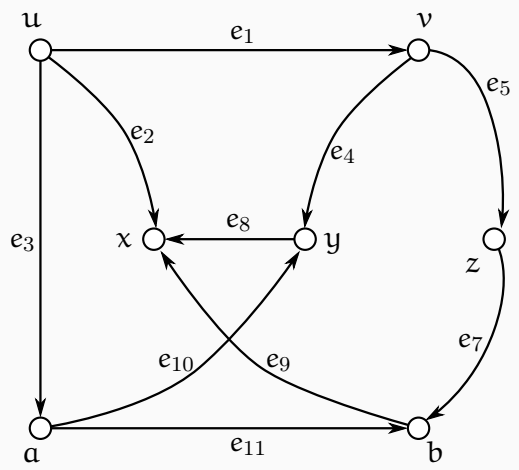
Определение 90: даг

Ориентиран граф или ориентиран мултиграф без цикли се нарича *даг*.

Терминът е транслитерация на *dag*, което идва от *directed acyclic graph*.

Ориентираният граф, показан на Фигура 3.1, не е даг. От него обаче може да получим дага, показан на Фигура 3.6, изтривайки реброто e_6.

Фигура 3.6 : Рисунка на даг.



Забележете, че никой даг не може да има примки, понеже всяка примка е цикъл.

Следната теорема е аналог на теоремата, казваща, че всяка частична наредба над краен домейн има поне един минимален елемент и поне един максимален елемент.

Теорема 45: Поне един източник и поне един сифон в даг

Всеки даг има поне един източник и поне един сифон.

Доказателство: БОО, ще докажем твърдението само за източник. Да допуснем противното: G е даг, в който няма източник. Нека a_1 е произволен връх от $V(G)$. По допускане, a не е източник, така че съществува ребро (a_2, a_1). Но a_2 също не е източник, тъй като сме допуснали, че източници няма. Тогава съществува ребро (a_3, a_2). Но a_3 също не е източник. И така нататък. По този начин можем да построим път

$$p = a_k, a_{k-1}, \ldots, a_3, a_2, a_1$$

за колкото искаме голямо k. Но, ако $k > n$, то по принципа на Dirichlet трябва да има поне едно повтаряне на връх в p. Иначе казано, съществуват j и i, такива че $j > i$ и a_j и a_i са върхове в p, като a_j и a_i са един и същи връх в G. Тогава подпътят $a_j, \ldots, a_i$ на p представлява цикъл в G. Но G няма цикли, защото е даг. Заключаваме, че допускането, че няма нито един източник, е невярно. □

Като пример за приложението на Теорема 45 да вземем дага, показан на Фигура 3.6: u е източник, x е сифон и други източници или сифони няма.

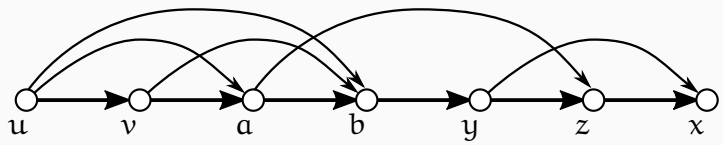
Забележете, че всеки изолиран връх е едновременно източник и сифон, следователно всеки празен ориентиран граф е даг, в който всеки връх е едновременно източник и сифон.

Очевидно е, че даговете нямат Хамилтонови цикли, защото нямат цикли изобщо. Даг обаче може да има Хамилтонов път. Ако даг $G = (\{v_1, v_2, \ldots, v_n\}, E)$ има Хамилтонов път

$$p = v_{i_1}, v_{i_2}, \ldots, v_{i_n}$$

където $(i_1, i_2, \ldots, i_n)$ е пермутация на $\{1, 2, \ldots, n\}$, можем да мислим за G като за въпросния Хамилтонов път плюс още ребра, ако има такива. Ако има още ребра, всяко от тях е от вида (v_{i_j}, v_{i_k}), където $j < k$; в някакъв смисъл, всяко ребро e, което не е от Хамилтоновия път, има същата посока като посоката на Хамилтоновия път. Последното е доста очевидно – инак би имало цикъл в графа. Фигура 3.7 илюстрира даг с Хамилтонов път, като Хамилтоновият път е акцентиран. Ясно е в какъв смисъл ребрата, които не са от Хамилтоновия път, имат същата посока като неговата.

Фигура 3.7 : Даг с Хамилтонов път.



Хамилтоновият път в дага, илюстриран на Фигура 3.7, е u, v, a, b, y, z, x. Други Хамилтонови пътища няма и това не е случайно, както става ясно от следната теорема.

Теорема 46: Единственост на Хамилтонов път в даг

Всеки даг, който не е мултиграф, има най-много един Хамилтонов път.

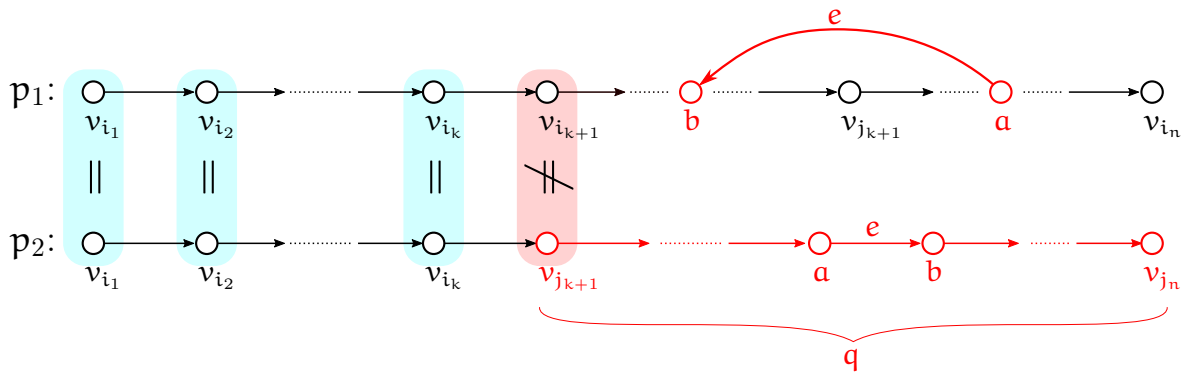
Доказателство: Да допуснем, че $G = (\{v_1, \ldots, v_n\}, E)$ е даг, който не е мултиграф и в който има два различни Хамилтонови пътища

$$p_1 = v_{i_1}, v_{i_2}, \ldots, v_{i_n}$$
$$p_2 = v_{j_1}, v_{j_2}, \ldots, v_{j_n}$$

където $(i_1, i_2, \ldots, i_n)$ и $(j_1, j_2, \ldots, j_n)$ са различни пермутации на $\{1, \ldots, n\}$. Да разгледаме най-левия връх, в който p_1 и p_2 се различават. Такъв трябва да има, защото, ако $i_1 = j_1$ и $i_2 = j_2$ и $\ldots$ и $i_n = j_n$, пермутациите биха съвпадали и p_1 би бил същият път като p_2. И така, щом пътищата се различават, има най-ляв връх, който е различен в тях. Да кажем, че

$i_1 = j_1, \ldots, i_k = j_k$, но $i_{k+1} \neq j_{k+1}$, така че до позиция k включително двата пътя съвпадат, но в позиция $k+1$ имат различни върхове.

Забележете, че p_1 не съдържа $v_{j_{k+1}}$ на позиция $k+1$, понеже $v_{j_{k+1}}$ и $v_{i_{k+1}}$ са различни върхове, но p_1 съдържа $v_{j_{k+1}}$, защото p_1 е Хамилтонов път. Обаче $v_{j_{k+1}}$ не може да е нито на първа, нито на втора, ..., нито на k-та позиция в p_1 поради допускането, че $i_1 = j_1, \ldots, i_k = j_k$. Ерго, $v_{j_{k+1}}$ се среща в p_1 на позиция, по-голяма от $k+1$. Но тогава подпътят q на p_2 от $v_{j_{k+1}}$ включително до v_{j_n} е по-дълъг от подпътя на p_1 от $v_{j_{k+1}}$ включително до v_{i_n}. Очевидно е, че q съдържа ребро $e = (a, b)$, такова че по отношение на p_1 е вярно, че a е вдясно от b. Последното влече наличие на цикъл в графа, което е невъзможно, тъй като той е даг.



□

3.1.7 Силна и слаба свързаност в ориентирани графи

Разглеждаме обикновени ориентирани графи, тъй като наличието на примки или паралелни ребра е без значение за силната или слабата свързаност.

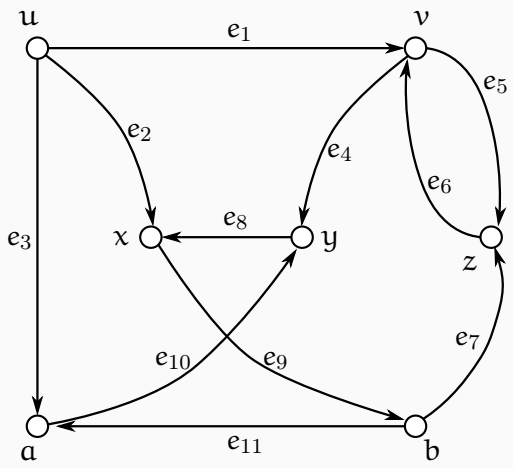
Забележете сходството между Определение 19 и Определение 91.

Определение 91: Силна свързаност в ориентиран граф. Силно свързан ориентиран граф.

Нека $G = (V, E)$ е граф. За всеки два върха $u, v \in V$ казваме, че u и v са *силно свързани*, ако съществува път от u до v и съществува път от v до u. G е *силно свързан*, ако всеки два върха в него са силно свързани.

Примерно, в ориентирания граф, показан на Фигура 3.8, всеки два върха от $\{a, b, v, x, y, z\}$ са силно свързани, докато u и b не са силно свързани, понеже съществува път от u до b, но не съществува път от b до u.

Фигура 3.8 : Всеки два върха от $\{a, b, v, x, y, z\}$ са силно свързани.



Ако гледаме на силната свързаност като на релация, то тя е

- рефлексивна, защото всеки връх е силно свързан със себе си чрез тривиален път с дължина 0,
- симетрична, по очевидни причини,
- транзитивна, защото, ако u и v са силно свързани и v и w са силно свързани, то u и w също са силно свързани; простите пътища, необходими, за да покажем силната свързаност на u и w може да бъдат получени от простите пътища, задаващи силните свързаности между u и v и между v и w, като може да се наложи да "изрежем" подпътища по начин, аналогичен на този от Лема 4.

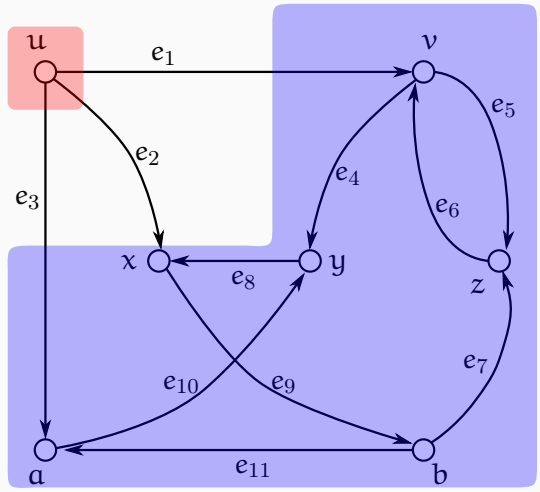
Следователно, релацията на силна свързаност е релация на еквивалентност.

Определение 92: Силно свързани компоненти.

Нека $G = (V, E)$ е ореинтиран граф. Подграфите на G, индуцирани от класовете на еквивалентност на релацията на силна свързаност, се наричат *силно свързаните компоненти на* G.

Като пример да разгледаме ориентирания граф на Фигура 3.8. Той има две силно свързани компоненти: едната е $(\{u\}, \varnothing)$, а другата е $(\{a, b, v, x, y, z\}, \{e_4, e_5, e_6, e_7, e_8, e_9, e_{10}, e_{11}\})$. Тези силно свързани компоненти са означени с цветни фонове на Фигура 3.9.

Фигура 3.9 : Двете силно свързани компоненти на графа от Фигура 3.8.



Забележете, че при ориентираните графи е възможно да има ребра извън силно свързаните компоненти. Примерно, на Фигура 3.9, ребрата e_1, e_2 и e_3 не принадлежат на никоя свързана компонента. В контраст с това, при неориентираните графи е невъзможно да има ребро, което не е в никоя свързана компонента.

Нещо повече. Ако ориентираният граф $G = (V, E)$ е даг, то множеството от силно свързаните му компоненти е $\{\{v\} \mid v \in V\}$. С други думи, всеки връх е една силно свързана компонента сам по себе си. И нито едно ребро не принадлежи на никоя силно свързана компонента. Също така е вярно, че ако множеството от силно свързаните компоненти е $\{\{v\} \mid v \in V\}$, то ориентираният граф е даг.

Можем да дефинираме "силно свързаните компоненти" и с алтернативно определение, аналогично на Определение 21: силно свързаните компоненти са максималните по включване силно свързани подграфи.

Определение 93: Фактор-граф на ориентиран граф

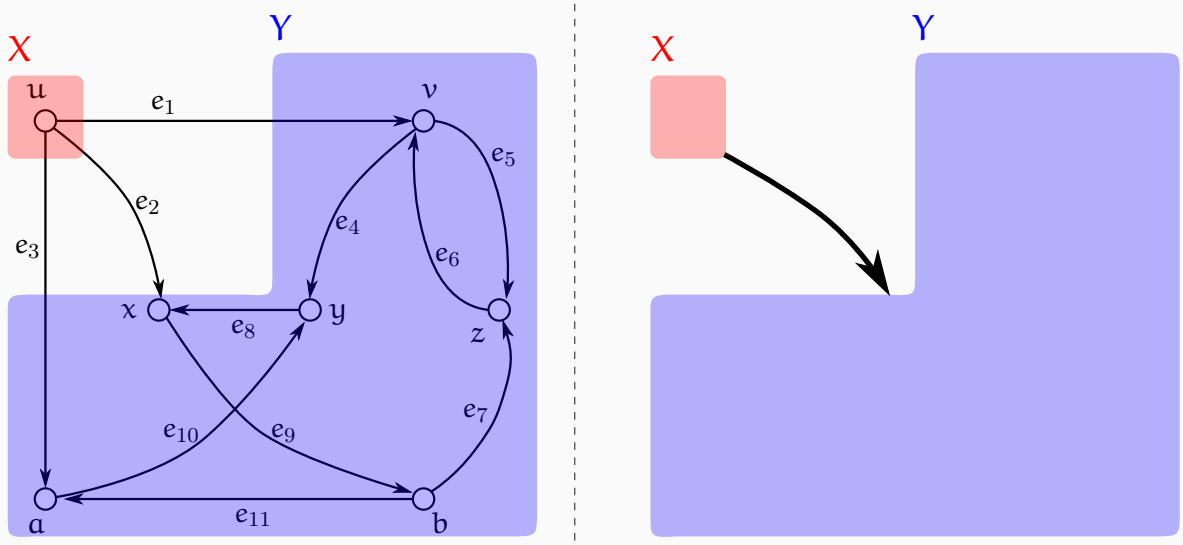
Нека $G = (V, E)$ е ориентиран граф. Нека $\mathcal{W}$ е разбиването на V, което се задава от релацията на силна свързаност. *Фактор-графът на* G, който означаваме с "$G/\sim$", е графът с множество от върхове $\mathcal{W}$, в който за всеки $X, Y \in \mathcal{W}$ има ребро (X, Y) тогава и само тогава, когато в G има поне едно ребро от връх на X до връх на Y.

Определение 93 може да се направи по-общо: спрямо **произволно** разбиване на върховете[†], а не непременно това, което релацията на силна свързаност задава. За целите на тези лекции, както и за приложението им в алгоритмите, това би било безполезно, така че за нас "фактор-граф" винаги е спрямо разбиването, което се дава от релацията на силна свързаност.

Като пример за фактор-граф да разгледаме графа, показан на Фигура 3.8 и на Фигура 3.9 заедно със силно свързаните си компоненти. Фактор-графът на този граф има два върха и едно ребро. Това е илюстрирано на Фигура 3.10, на която вляво е изобразен самият граф със своите две силно свързани компоненти, наречени X и Y, а вдясно е показан фактор-графът му с двата си върха X и Y и реброто (X, Y), което присъства, защото в графа има ребро от връх от X, а именно u, до поне един връх от Y (а именно, има ребра от u до a, x и v).

[†]Тоест, спрямо произволна релация на еквивалентност $\sim$. Както знаем, "разбиване на върховете" и "релация на еквивалентност върху множеството на върховете", в някакъв смисъл, е едно и също нещо

Фигура 3.10 : Фактор-графът на графа от Фигура 3.8.



Силно свързаните компоненти. Фактор-графът.

Лесно се вижда, че, тъй като силно свързаните компоненти на всеки даг имат по един единствен връх, то, ако G е даг, неговият фактор-граф, неформално казано, съвпада с него. Формално казано, той е изоморфен с него. Нещо повече. Вярно е, че ориентиран граф без примки е даг тстк е изоморфен със своя фактор-граф.

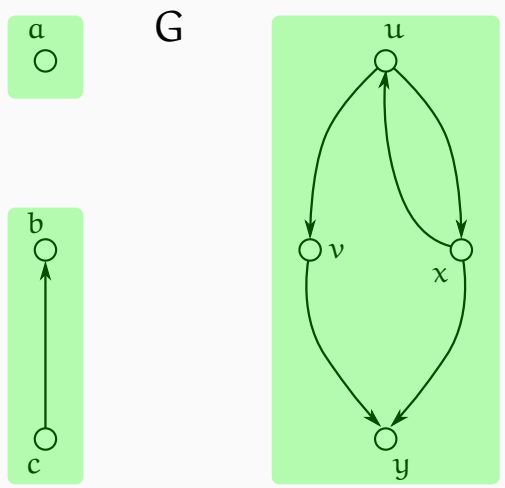
Следното определение ползва Определение 84

Определение 94: Слабо свързани компоненти.

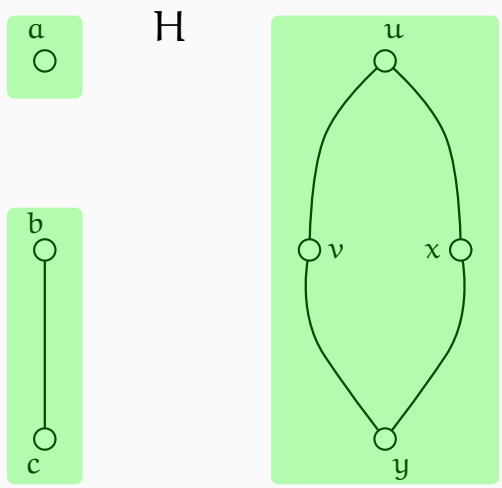
Нека $G = (V, E)$ е ориентиран граф. Нека H е съответният му неориентиран граф. Подграфите на G, индуцирани от класовете на еквивалентност на релацията на свързаност върху H, се наричат *слабо свързаните компоненти на* G.

Иначе казано, слабо свързаните компоненти са максималните по включване подграфи на G, между върховете на които няма пътища в никоя посока. Като пример да разгледаме Фигура 3.11. На нея е показан ориентиран граф G и съответния му неориентиран граф H. Вижда се, че трите слабо свързани компоненти на G, маркирани със зелен фон, точно съответстват на трите свързани компоненти на H, маркирани също със зелен фон.

Фигура 3.11 : Слабо свързани компоненти на ориентиран граф.



Ориентиран граф G, който има три слабо свързани компоненти, точно съответстващи на трите свързани компоненти на H.



Неориентираният граф H, съответен на G. Очевидно H има три свързани компоненти, очертани тук със зелен фон.

Очевидно е, че всяко ребро принадлежи на точно една слабо свързана компонента. Докато, както видяхме, по отношение на силната свързаност може да има ребра извън силно свързаните компоненти и дори може всички ребра да са извън силно свързаните компоненти (тстк графът е даг).

3.1.8 Ориентирано разстояние

Да си припомним Определение 22 и Определение 25. Ще въведем нещо подобно на Определение 25 за ориентирани графи, като аналогията е възможна само донякъде.

Определение 95: Разстояние в ориентиран граф.

Нека $G = (V, E)$ е ориентиран граф. За всеки два не непременно различни върха u и v, *разстоянието от* u *до* v е

- дължината на най-къс път от u до v, ако съществува път от u до v,
- ∞, в противен случай.

И при ориентираните графи бележим разстоянието с "$\mathrm{dist}(u, v)$". При ориентираните графи обаче е възможно $\mathrm{dist}(u, v) \neq \mathrm{dist}(v, u)$, като може едното от $\mathrm{dist}(u, v)$ и $\mathrm{dist}(v, u)$ да е число, а другото, ∞, а може и двете да са числа, но различни. Естествено, $\mathrm{dist}(u, u) = 0$ винаги.

Наблюдение 31

Функцията dist от Определение 95 не е метрика, понеже е възможно $\mathrm{dist}(u, v) \neq \mathrm{dist}(v, u)$.

За пример да вземем свързания граф, показан на Фигура 3.8. В него $\mathrm{dist}(u, v) = 1$, $\mathrm{dist}(v, u) = \infty$, $\mathrm{dist}(v, z) = \mathrm{dist}(z, v) = 1$, $\mathrm{dist}(b, z) = 1$, $\mathrm{dist}(z, b) = 4$ и така нататък.

И едно предупреждение, което не е излишно, макар че повтаря вече казани неща. Въпреки че използваме един и същи запис за разстояния в неориентирани и за разстояния в ориентирани графи, тези две понятия за разстояние са много различни. Когато видим "$\mathrm{dist}(u, v)$", първото, което трябва да съобразим, е дали става дума за неориентирани графи, в които разстоянието е симетрично и е метрика, или за ориентирани графи, в които разстоянието не е непременно симетрично и не е метрика.

3.1.9 Ориентирани коренови дървета

Определение 96: арборесценция

Арборесценция, на английски *arborescence* или *out-tree*, е всеки даг $G = (V, E)$, в който съществува връх $r \in V$, такъв че за всеки връх $u \in V$ е вярно, че съществува уникален ориентиран път от r до u.

Неформално, всяка арборесценция се получава от някое кореново дърво, което тук разглеждаме като неориентиран граф, с даване на ориентация на ребрата "навън от корена".

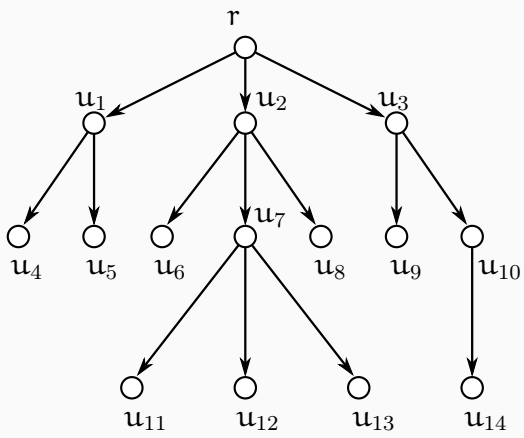
Определение 97: антиарборесценция

Антиарборесценция, на английски *anti-arborescence* или *in-tree*, е всеки даг $G = (V, E)$, в който съществува връх $r \in V$, такъв че за всеки връх $u \in V$ е вярно, че съществува уникален ориентиран път от u до r.

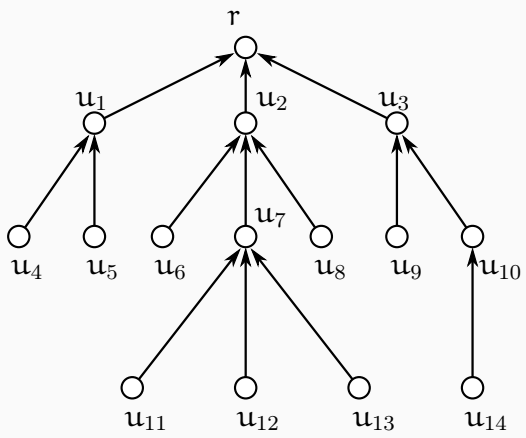
Неформално, всяка антиарборесценция се получава от някое кореново дърво, което тук разглеждаме като неориентиран граф, с даване на ориентация на ребрата "навътре към корена".

Фигура 3.12 показва арборесценция и антиарборесценция, чиито съответен неориентиран граф е едно и също кореново дърво.

Фигура 3.12 : Арборесценция и антиарборесценция.



Арборесценция с корен r.



Антиарборесценция с корен r.

3.2 Тегловни графи

Има много житейски ситуации, в които върху някакви обекти е дефинирано двуместно отношение[†] и това може да се опише с граф (ориентиран или не, мултиграф или не, в момента няма значение), като обаче отношението е по-сложно и по-богато от булевото "два обекта или са в отношение, или не са".

Да кажем, че имаме компютърна мрежа, в която върховете са устройства като лаптопи, настолни компютри, сървъри, суичове и така нататък, и две устройства може да са или да не са свързани директно. Това се моделира лесно с граф, в който върховете са устройствата, а ребрата са директните връзки. Дали директните връзки са двупосочни или не, е същото като дали ребрата са неориентирани или ориентирани; в момента това няма значение. Да кажем, че в даден момент различните директни връзки имат различни капацитети са пренос на информация за единица време: примерно, 155.56 килобита в секунда, 1.014 гигабита в секунда и така нататък. Този аспект на мрежата не може да бъде моделиран адекватно с обикновен граф! Графът ни казва само къде има и къде няма директна връчка. В крайно ограничената ситуация, в която, да кажем, има само четири различни скорости на предаване на информация—1, 2, 3 и 4 мегабита в секунда—можем да опитаме да моделираме с мултиграф, при който сноповете съдържат или 1, или 2, или 3, или 4 ребра, като всяко ребро моделира предаване на 1 мегабит в секунда. Тази идея очевидно "не работи" в общия житейски случай, в който скоростите са някакви положителни реални числа[‡]. Можем обаче да моделираме адекватно чрез граф, всяко ребро на който е асоциирано с число, което число има смисъл на въпросната скорост на предаване.

Аналогично, ако имаме пътна мрежа с върхове-градове и шосета-директни връзки, като дължините на шосетата имат значение, можем да моделираме адекватно с неориентиран (ако шосетата са двупосочни) граф (или мултиграф, ако допускаме повече от един директно шосе между два града; това е напълно възможно), в който ребрата са асоциирани с положителни реални числа, които имат смисъл на дължините на съответните шосета. Опитите да избегнем асоцииране на ребрата с числа и наместо това да моделираме с дискретни трикове като например този:

> множеството от ребрата на графа да се замени с множество от независими пътища (в графовия смисъл; вижте Определение 17) по такъв начин, че на шосе, дълго k километра, да отговаря път в графа с точно k ребра;

са обречени, също както в предишния пример. Ако не асоциираме ребрата с числа, не можем да моделираме адекватно и това е.

Ето формалната дефиниция, мотивирана от подобни съображения. Ще я направим за ориентиран мултиграф с възможни примки, а за всички други видове графи е очевидно как да се промени дефиницията.

[†]Нарочно не се казва "релация", защото това понятие има строго дефиниран смисъл, от който в момента искаме да се отдалечим.

[‡]Ако сме педантични, скоростите са рационални числа, понеже "истинските" трансцендентни реални числа са в някакъв смисъл недостъпни директно—те и затова се казват "трансцендентни"—но толкова педантични няма да ставаме.

Определение 98: Тегловен граф

Тегловен ориентиран мултиграф е наредена четворка $G = (V, E, f_G, w)$, където V е непразно множество, чиито елементи се наричат върхове, E е множество, чиито елементи се наричат ребра, $V \cap E = \varnothing$,

$$f_G : E \to V \times V$$

е свързващата функция и

$$w : E \to \mathbb{R}$$

е *тегловната функция*.

Възможно е да се дадат тегла и на върховете, но в задачите, които ще разгледаме, от това няма смисъл, така че тегла ще имат само ребрата.

Не искаме да се ограничаваме до положителни тегла в общия случай, затова кодомейнът на тегловната функция е $\mathbb{R}$, а не $\mathbb{R}^+$. Наистина, ако теглата моделират физически характеристики като време или разстояние, има смисъл да постулираме, че са положителни. Но в доста реални задачи теглата моделират аспекти на човешка дейност, в която абстракцията на отрицателните числа е полезна. Примерно, ако графът моделира някакви потоци, може да е много полезно да допускаме отрицателни тегла:

- ако потокът е от стоки, то да кажем, че са пренесени 10 тона цимент от Варна до София в някакъв смисъл е същото като да кажем, че са пренесени -10 тона цимент от София до Варна;
- ако потокът е от пари, то да кажем, че Ангел е дал на Боби 10 лева в някакъв смисъл е същото като да кажем, че Боби е дал на Ангел -10 лева.

Забележете, че функцията на теглата е тотална, а не частична. С други думи, ако има тегла, всички ребра са с тегла. Авторът на лекционните записки не се сеща за естествена житейска задача, която би се моделирала с граф, в който само част от ребрата имат тегла.

Задачите с графи без тегла по правило може да се разглеждат като частни случаи на задачи върху тегловни графи, само че теглата са едни и същи; да кажем, всички тегла са единици. По този начин, графите без тегла са частен случай на тегловните графи – просто теглата са само единици и не сме ги споменали. А наличието на тегла на ребрата води до поява на дефиниции, които нямат смисъл при графи без тегла, или промяна на съществуващи дефиниции. Ето два примера.

- При тегловните неориентирани свързани графи има смисъл да говорим за сумата от теглата на ребрата на покриващо дърво (вижте Секция 4.3). В графите без тегла това би било безсмислено: ние знаем, че, ако броят на върховете на графа е n, то всяко покриващо дърво има точно $n-1$ ребра (Лема 11), така че всички покриващи дървета биха имали едно и съща сума от теглата на ребрата. В тегловните графи обаче тази сума е, в общия случай, различна за различни покриващи дървета,, така че задачата за намиране на покриващо дърво с минимална или максимална сума от теглата на ребрата е смислена (вижте Секция 4.3).

  Естествено, тези съображения са валидни, ако мярката за дадено покриващо дърво е сумата от теглата на ребрата. Възможни са и други мерки, примерно, може мярката

да е диаметърът на дървото (припомнете си Определение 24). В такъв случай задачата би била смислена и върху графи без тегла.

- При тегловните графи по правило дължина на път се дефинира като сумата от теглата на ребрата му (вижте Секция 4.4). Възможни са и други мерки за дължина на път в тегловен граф—примерно, максимумът от теглата на ребрата на пътя—но сумата е най-често използвана мярка в практиката.

  И така, в контекста на тегловните графи, "дължина на път" по правило означава сумата от теглата на ребрата, а не броят на ребрата, както е в графите без тегла. Това е пример за промяна за общоизвестна дефиниция от областта на графите при разглеждането на тегловни графи.

3.3 Хиперграфи

Ако трябва да сме прецизни, хиперграфите не са вид графи, а, точно обратното, графите са вид хиперграфи. "Хипергpаф" е обобщение на "граф" по горе-долу същия начин, по който "n-арна релация" е обобщение на "бинарна релация". Авторът на тези лекционни записки обаче реши да не въвежда отделна глава за хиперграфи, тъй като тук върху тях няма акцент.

За повече информация вижте класиките *Graphs and Hypergraphs* [10] и *Hypergraphs: Combinatorics of Finite Sets* [9] на Berge, съвременната *Hypergraph Theory: An Introduction* на Bretto [14] и обзорната статия *Hypergraphs: an introduction and review* на Ouvrard [46]. За някои приложения на хиперграфите вижте статията *Learning with Hypergraphs: Clustering, Classification, and Embedding* [61].

Следното определение по същество повтаря определението на Berge [9, стр. 1], но е формулирано като обобщение на Определение 14.

Определение 99: Хиперграф

Хиперграф е наредена тройка $H = (V, E, f_H)$, където V е непразно множество, чиито елементи се наричат *върхове*, E е множество, чиито елементи се наричат *ребра*, $V \cap E = \varnothing$ и

$$f_H : E \to 2^V \setminus \varnothing$$

е *свързващата функция*. Хиперграфът е *прост* (казваме още, че е фамилия на Sperner), ако

$$\forall e', e'' \in E : f_H(e') \subseteq f_H(e'') \to e' = e''$$

Забележете, че "хипергpаф" се явява обобщение на "мултиграф", а не просто на "граф". Определение 99 позволява няколко ребра да се изобразяват върху едно и също множество от върхове.

Ако e е ребро и $f_H(e) = \{v_1, \ldots, v_k\}$, казваме "$v_1, \ldots, v_k$ са върховете на e", тоест, третираме e като (непразно) подмножество на V. При хиперграфите не говорим за "краища на ребро".

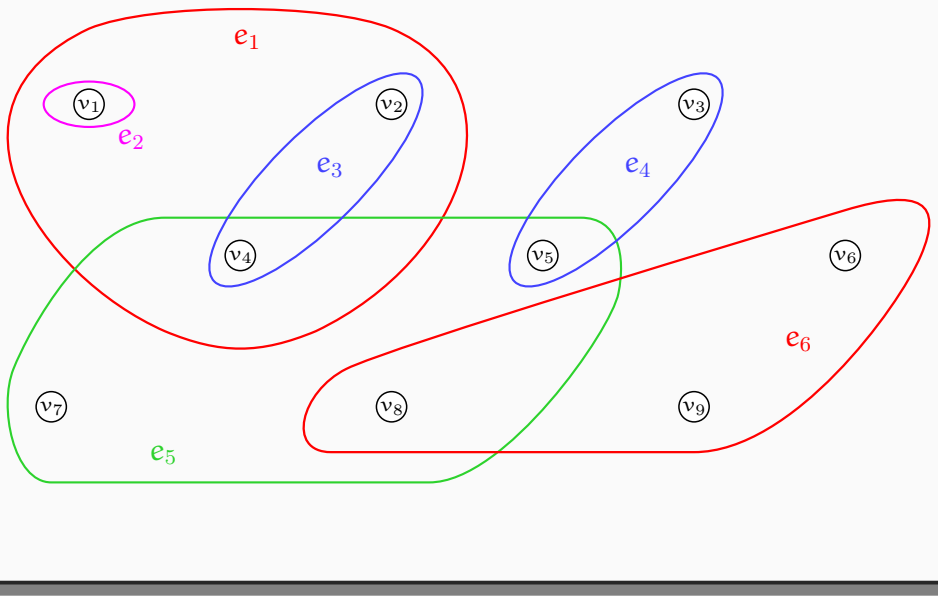
Прост хиперграф очевидно е такъв, в който свързващата функция е инекция и освен това ребрата—ако гледаме на тях като на множества от върхове—не може да се съдържат същински.

Някои автори като Bretto [14] казват "хипер-ребра" наместо "ребра". Също така, Определение 99, както и Bretto [14], допуска върхове, които не са в никое (хипер-)ребро и които отговарят на изолираните върхове от графите, докато Berge [9, стр. 1] не допуска такива изолирани върхове; при него всеки връх е в поне едно ребро. Ако няма изолирани върхове, то "прост хипергpаф" и "покриване на множеството от върховете" са съвпадащи понятия.

Забележете, че обикновен граф (Определение 1) е прост хипергpаф, в който всяко ребро има кардиналност 2, а мултиграф с възможни примки (Определение 14) е хипергpаф, в който всяко ребро има кардиналност 1 или 2.

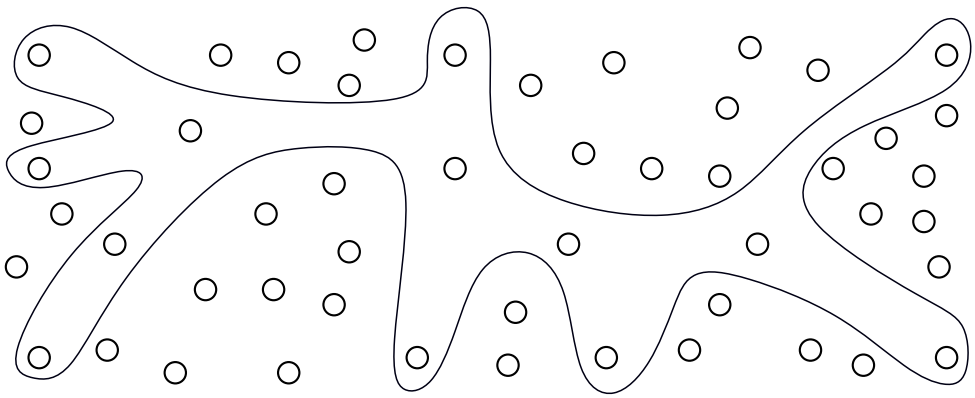
Фигура 3.13 показва пример за хипергpаф. Типично, ребрата се рисуват с прости затворени криви—окръжности или овали, ако е възможно—като образът на ребро на рисунката огражда образите на неговите върхове в хиперграфа.

Фигура 3.13 : Хиперграф H.



Хиперграфът, показан на Фигура 3.13, е $H = (V, E, f_H)$, където $V = \{v_1, \ldots, v_9\}$, $E = \{e_1, \ldots, e_6\}$ и $f_H(e_1) = \{v_1, v_2, v_4\}$, $f_H(e_2) = \{v_1\}$, $f_H(e_3) = \{v_2, v_4\}$, $f_H(e_4) = \{v_3, v_5\}$, $f_H(e_5) = \{v_4, v_5, v_7, v_8\}$ и $f_H(e_6) = \{v_6, v_8, v_9\}$. Хиперграфът няма изолирани върхове, така че удовлетворява формалните изисквания на Определение 99. Той не е прост, защото има ребра, съдържащи други ребра, например e_1 съдържа и e_2, и e_3, ако мислим за ребрата като за множества от върхове; и не е "мулти", защото свързваща функция е инекция (няма две или повече ребра, изобразяващи се върху едно и също множество от върхове, което Определение 99 позволява). Реброто e_2 е примка, в терминология на графите.

Веднага се вижда, че хиперграфи се рисуват трудно. Докато при обикновените графи можем лесно да изобразим граф с десетки върхове и много десетки ребра по такъв начин, че човек лесно да си го представи, то хиперграф може да бъде възприет визуално само ако е много малък, какъвто е примерът на Фигура 3.13. Ако ребрата на този хиперграф бяха много десетки, би било практически невъзможно да се нарисува ясно, с отчетливи ребра, дори с умело ползване на цветове. Хиперграфът на Фигура 3.13 има само шест ребра, като само три от тях имат повече от два върха. Това позволява групирането на образите на върховете в рисунката по такъв начин, че ребрата да са нарисувани с изпъкнали криви. Читателят лесно може да си представи ситуация, в която ребра се налага ребра да се рисуват с вдлъбнати "амеби", за да не обхващат образите на върхове, които не влизат в реброто:



Очевидно е, че рисунка с десетки такива "амеби" не би могла да бъде възприета визуално и поради това би била безполезна.

Така дефинираните хиперграфи се явяват обобщение на **неориентираните** графи, тъй като върховете, отговарящи на дадено ребро, са **множество** – обект, който няма наредба. Възможно е да се дефинират и ориентирани хиперграфи, наречени *dirhypergraphs* в [14, стр. 95], при което всяко ориентирано (хипер-)ребро има начало, което е непразно множество от върхове, и край, който също е непразно множество от върхове. Авторът на тези лекционни записки вижда и друг начин да се дефинират ориентирани (хипер-)ребра: посоката да е една от възможните пермутации на върховете на (хипер-)реброто. Тук няма да разглеждаме ориентирани хиперграфи. В класическите книги на Berge [10] и [9] не се говори за ориентирани хиперграфи, така че това е сравнително ново понятие.

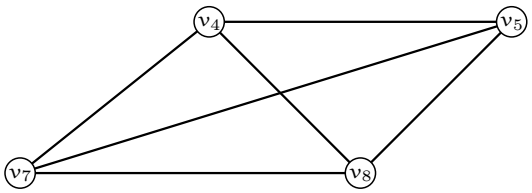
Ето пример, в който хиперграф възниква естествено. Да си представим множество от автори и множество от статии, написани от тези автори. Да кажем, че множеството от авторите е $\{v_1, \ldots, v_9\}$, а множеството от статиите е $\{e_1, \ldots, e_6\}$. Нека

- v_1, v_2 и v_4 са авторите на e_1,
- v_1 е авторът на e_2,
- v_2 и v_4 са авторите на e_3,
- v_3 и v_5 са авторите на e_4,
- v_4, v_5, v_7 и v_8 са авторите на e_5, и
- v_6, v_8 и v_9 са авторите на e_6.

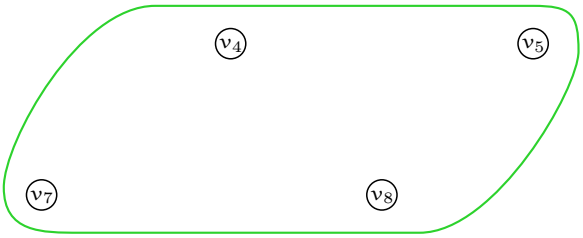
Но Фигура 3.13 показва точно това!

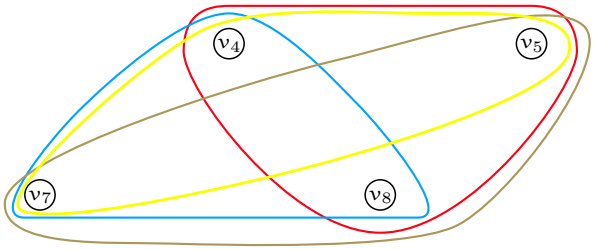
Хиперграфите може да бъдат представени недвусмислено чрез обикновени графи, но начинът да стане това не е непременно очевидният. Да вземем за пример хиперграфа от Фигура 3.13. Човек може да се изкуши да помисли, че, тъй като ребрата се идентифицират с множества върхове (хиперграфът не е "мулти"), можем да представим хиперграфа като граф със същото множество от върхове, като (хипер-)ребрата са клики. Тази идея, разбира се, не работи. Както се вижда от Фигура 3.13, наличието на ребра, съдържащи други ребра (примерно, e_1 съдържа e_3), обезсмисля опитите за представяне на ребрата чрез клики, защото 3-кликата на e_1 би съдържала същински 2-кликата на e_3, при което, при кликовото представяне, e_3 би изчезнало.

Но дори хипеграфът да е прост, което предотвратява съществуване на вложени ребра, недвусмислено кликово представяне по този начин е невъзможно. Като пример, ако се опитаме да представим реброто e_5 от хиперграфа на Фигура 3.13 като 4-клика, тя би изглеждала така:



Тази 4-клика би могла да представя (хипер-)ребро с 4 върха v_4, v_5, v_7 и v_8, което е случаят на Фигура 3.13 и което е показано вляво, но би могла да представя 4 (хипер-)ребра, всяко с по три върха, наредени в нещо цикъл, което е показано вдясно:





Само от 4-кликата не може да разберем какво точно има в хиперграфа. И така, не можем да представяме недвусмислено хиперграфа като обикновен граф със същото множество върхове, заместваики (хипер-)ребрата с клики.

До същия извод можем да стигнем и с чисто комбинаторни съображения: върху дадено множество върхове, хиперграфите са много повече от обикновените графи. Това е в сила дори ако разглеждаме само прости хиперграфи.

Допълнение 25: За броя на графите и хиперграфите

Нека е фиксирано множество върхове V, като $|V| = n$. Броят на всички възможни графи—неориентирани, без примки, не "мулти"—над V е

$$2^{\frac{n(n-1)}{2}}$$

Тук става дума за именувани графи. Тоест, смятаме изоморфните графи за различни (вижте Подсекция 2.8.2 за разликата между именуван и неименуван граф). Причината броят да е такъв е много проста: всички възможни ребра са $\binom{n}{2} = \frac{n(n-1)}{2}$ на брой (това следва веднага от Наблюдение 3, което пък е частен случай на Лема 6 при $k = 1$) и всяко от тях може да присъства или не независимо от другите.
Ако позволим и примки, броят на графите нараства до

$$2^{\frac{n(n+1)}{2}}$$

Броят на мултиграфите не може да бъде ограничен от функция на n по очевидни причини.
Определение 99 позволява много ребра да имат едно и също множество върхове, така че и броят на хиперграфите съгласно Определение 99 не може да бъде ограничен от функция на n. Да допуснем, че най-много едно ребро може да съдържа дадено множество от върхове; тоест, че свързващата функция е инекция. Тогава точната горна граница за броя на хиперграфите е:

$$2^{2^n-1}$$

Изваждането -1 в степеннния показател е заради това, че не може да има празно ребро (свързващата функция не изобразява ребра в празното множество), така че всички възможни (хипер-)ребра са $2^n - 1$ на брой.

Дори да се ограничим само до хиперграфите, чиито ребра имат точно три върха всяко, броят на тези хиперграфи е

$$2^{\binom{n}{3}} = 2^{\frac{n(n-1)(n-2)}{6}}$$

тъй като различните ребра са $\binom{n}{3} = \frac{n(n-1)(n-2)}{6}$. Очевидно е, че

$$\lim_{n\to\infty} \frac{2^{\frac{n(n-1)(n-2)}{6}}}{2^{\frac{n(n-1)}{2}}} = \infty$$

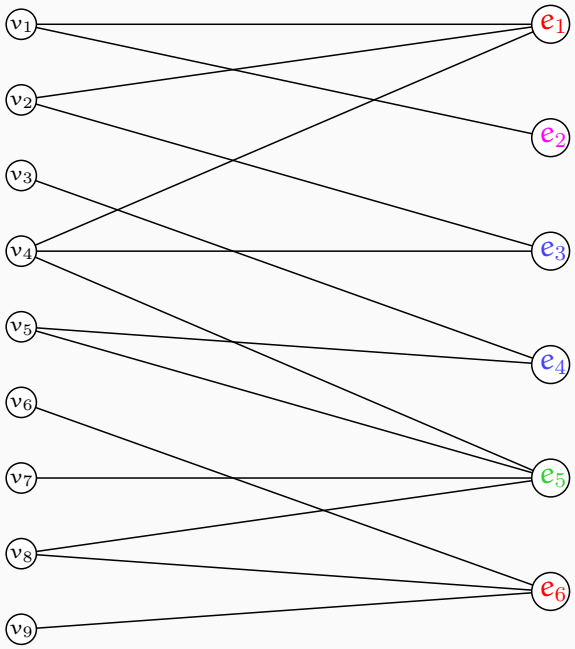
И така, броят на обикновените графи е изчезващо малка функция пред броя на доста ограничените прости хиперграфи с точно три върха в ребро.

Може да представим хиперграф $H = (V, E, f_H)$ чрез обикновен двуделен граф $G = (V, E, E')$, където

$$\forall x \in V\, \forall y \in E : (x, y) \in E' \leftrightarrow x \in f_H(y)$$

Да си припомним, че съгласно Нотация 3, $\boxed{G = (V, E, E')}$ означава двуделен (обикновен) граф с дялове V и E и множество от ребра E'. И така, върховете на G са върховете **и ребрата** на H, разбити в два дяла. Фигура 3.14 показва представяне на хиперграфа от Фигура 3.13 чрез двуделен граф.

Фигура 3.14 : Двуделният граф, описващ хиперграфа от Фигура 3.13.

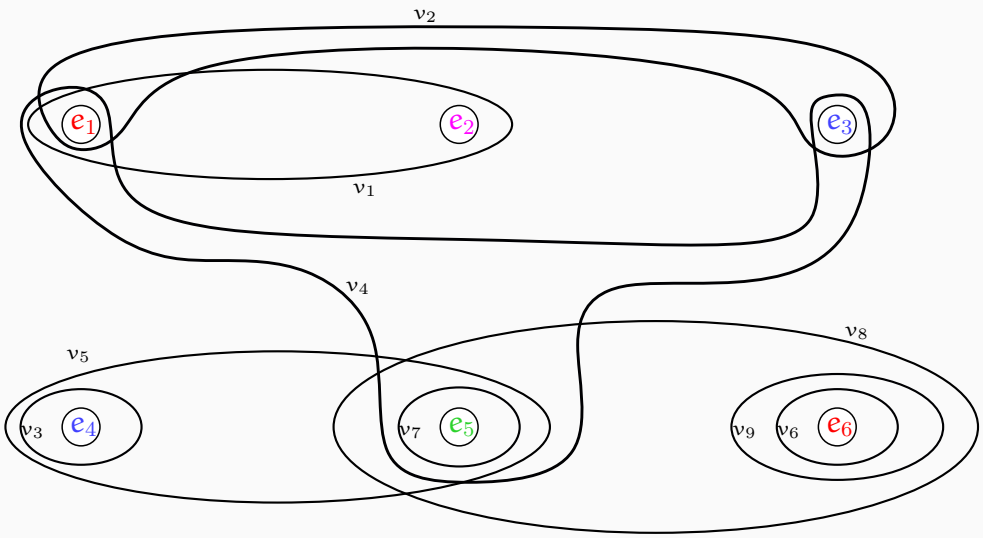


Идеята да представим хиперграф чрез съответен двуделен граф по същество съвпада с идеята да представим хиперграф чрез мартица на инцидентност – но за това ще говорим повече в Подсекция 4.1.2.

Този двуделен граф може да има размер, експоненциален във $|V|$, тъй като ребрата на хиперграфа може да са експоненциално много в броя на върховете дори при прости хиперграфи, а единият от дяловете на двуделния граф е точно множеството от ребрата на хиперграфа.

Ако видим само двуделния граф G, без да сме видели съответния хиперграф H, можем ли да разберем кой от дяловете отговаря на върховете на H и кой, на ребрата на H? Отговорът е "не". Примерно, G на Фигура 3.14 би могъл да представлява хиперграф с върхове $e_1, \ldots, e_6$ и ребра $v_1, \ldots, v_9$[†]. Фигура 3.15 показва точно този хиперграф; такъв хиперграф ще наричаме "дуален".

Фигура 3.15 : Хиперграфът, дуален на H от Фигура 3.13.



Ако пак гледаме на $v_1, \ldots, v_9$ като на автори, а на $e_1, \ldots, e_6$ като на техни статии, хиперграфът от Фигура 3.15 наистина изобразява дуалния на Фигура 3.13 начин да подгледнем на всичко това. На Фигура 3.13 опорното множество са авторите и статиите се идентифицират с авторите си, а на Фигура 3.15 опорното множество са статиите и авторите се идентифицират със своите статии. Както показва двуделния граф от Фигура 3.14, тези два различни погледа върху нещата могат да бъдат обединени в едно, при което и авторите, и статиите са опорни множества, като нито едното от тях не се идентифицира с подмножества на другото, а релацията между тях (ребрата на двуделния граф) ги третира по еднакъв начин.

Определение 100: Дуален хиперграф

Нека $H = (V, E, f_H)$ е хиперграф. *Дуалният хиперграф на* H е хиперграфът $H^* = (E, V, f_{H^*})$, където

- множеството от ребрата E на H е множеството от върховете на H^*,
- множеството V от върховете на H е множеството от ребрата на H^*,
- $\forall x \in V : f_{H^*}(x) = \{y \in E \mid x \in y\}$.

Дуалният хиперграф на дуалния хиперграф е оригиналният хиперграф. Тоест, $(H^*)^* = H$. Това се вижда много лесно, ако мислим за матрицата на инцидентност (вижте Подсекция 4.1.2): матрицата на инцидентност на H^* е транспонираната матрица на инцидентност на H; второ транспониране ни дава първоначалната матрица.

[†]Не е забранено върховете да се именуват с "e", а ребрата, с "v".

Примерът с H от Фигура 3.13 и дуалният му H^* показва, че дуалният хиперграф може да е мулти-хиперграф дори и оригиналният да не е такъв: забележете, че ребрата ν_6 и ν_9 на H^* имат едно и също множество от върхове, а именно $\{e_6\}$. Лесно може да се измисли пример за прост хиперграф, чийто дуален е мулти-хипеграф в този смисъл; да кажем, $H = (\{u, v\}, \{e\}, \{(e, \{u, v\})\})$, който е прост, чийто дуален е $H^* = (\{e\}, \{u, v\}, \{(u, \{e\}), (v, \{e\})\})$.

Понятието "степен на връх" в хиперграфите е обобщение на това понятие от мултиграфите, като при различни автори може да има леки разлики в дефинициите. Berge [9, стр. 3] казва, че степента на връх x, която се бележи с $d(x)$, е броят на ребрата, съдържащи x. Да кажем, че *примка в хиперграф* е ребро, съдържащо един единствен връх (по аналогия с примките в мултиграфите); примерно, на Фигура 3.13 примка е e_2. Berge не третира примките по особен начин, що се отнася до степените на върховете, поради което ν_1 от Фигура 3.13 има степен 2. В контраст с това, Bretto [14, стр. 2] казва

> The *star* $H(x)$ centered in x is the family of hyperedges $(e_j)_{j\in J}$ containing x; $d(x) = |J|$ is the *degree* of x except for a loop $\{x\}$ where the degree $d(x) = 2$.

Формално, това не е съвсем ясно казано, но най-естествената интерпретация е, че всяка примка допринася 2, а не 1, към степента на върха x, точно както в мултиграфите. Според Bretto, ν_1 от Фигура 3.13 има степен 3.

И двете конвенции (по отношение на примките) имат резон. Ако броим всяка примка два пъти, то понятието "степен на връх" в хиперграфите е точно обобщение на това понятие от мултиграфите с възможни примки; ако всяко ребро на хиперграфа има кардиналност $\leqslant 2$, степените на върховете са едни и същи независимо дали гледаме на него като на хиперграф или мултиграф. Ако броим всяка примка по един път като Berge това вече не е вярно, но тогава пък степента на всеки връх в хиперграфа съвпада точно със степента на този връх в съответния двуделен граф. Ако вземем за пример пак H от Фигура 3.13, то на Фигура 3.14 ясно се вижда, че степента на ν_1 в съответния двуделен граф е 2, а не 3.

Ако хиперграфът няма примки, дефинициите на Berge и Bretto съвпадат.

Също както при графите и мултиграфите, $\Delta(H)$ означава максималната степен на връх в H, а $\delta(H)$ означава минималната степен на връх в H, и H е регулярен, ако $\Delta(H) = \delta(H)$. Ако $\Delta(H) = \delta(H) = k$, то H е k-регулярен.

При хиперграфите има понятие "ранг", което няма аналог при графите и мултиграфите. *Ранг на хипеграфа* H е максималната кардиналност на ребро в него. Наистина, за всеки нетривиален граф и за всеки мултиграф, в който не всички ребра за примки, рангът е 2—ако гледаме на обекта като на хиперграф—така че е безсмислено да се говори за ранг при графи и мултиграфи. При хиперграфите обаче това е важно понятие. Примерно, рангът на хиперграфа от Фигура 3.13 е 4. Ако всички ребра имат една и съща кардиналност r, казваме, че хиперграфът е r-*унифомен*, или просто, че е *унифомен*.

Ако ползваме опроделението на "степен на връх" на Berge, очевидно е, че за всеки хиперграф $H = (V, E, f_H)$ е вярно, че H е k-регулярен тстк H^* е k-унифомен. В контекста на съответния двуделен граф G:

- H да е регулярен е същото като върховете от дяла на G, съответен на V, да имат една и съща степен, и
- H да е унифомен е същото като върховете от дяла на G, съответен на E, да имат една и съща степен.

Това е краят на изложението за хиперграфи в тези лекционни записки. При хиперграфите може да се дефинират пътища, цикли, свързаност, ацикличност, оцветяване и така нататък

по начини, които се явяват обощение на тези понятия от графите, но за това и повече вижте [9] и [14].

Глава 4

Задачи върху графи

4.1 Представяния на графи

Без съмнение думата "граф" идва от това, че такъв обект може да се нарисува; тоест, има **граф**ично представяне. Графичните представяния обаче са подходящи за

- хора – компютър трудно би разчел рисунката надеждно;
- малки графи – опитайте да нарисувате граф със стотици хиляди върхове и десетки милиони ребра, нещо, което лесно може да възникне при някоя практическа задача.

Ние искаме да решаваме задачи върху графи с компютър. За тази цел трябва да представяме графите по начини, удобни за алгоритмична обработка. Сега ще разгледаме основните такива представяния на графи.

В цялата Глава 4 върховете на графа, който разглеждаме, са целите положителни числа $1, 2, \ldots, n$. В теоретичен контекст е удобно да именуваме върховете с u, v и така нататък, или $u_1, \ldots, u_n$ и така нататък, но в програма, решаваща задача върху графи, най-удобно е върховете да са тип `int`, като върховете **са** целите числа от 1 до n, в Pascal-оподобна конвенция, или числата от 0 до $n-1$, в конвенцията на C. По този начин "минаването" през всички върхове на графа, което в теорията записваме като

foreach $v \in V(G)$ **do**
прави нещо

в софутуера става

```
for v from 1 to n do
begin
      прави нещо
end
```

или

```
for(v = 0; v <= n-1; v ++) {
      прави нещо
}
```

И преди да разгледаме представянията, една забележка. Въпросните представяния представят графи с **именувани** върхове (вижте Подсекция 2.8.2) или дори именувани и върхове, и ребра, ако говорим за матрица на инцидентност. Примерно, при представяне с матрица на съседство, разместването на редове и колони на матрицата в общия случай води до **друго представяне**, което е на **друг граф**. Той е изоморфен на първия, но като именуван граф е различен. Ако искаме да установим дали две различни представяния представят един и същи неименуван граф, трябва да пуснем алгоритъм, който взема за вход двете представяния и връща дали те представят изоморфни графи (и може би самия изоморфизъм, ако да). Но задачата за изоморфизъм е изключително тежка, от алгоритмична гледна точка, и далече извън обхвата на тези лекционни записки.

4.1.1 Матрици на съседство

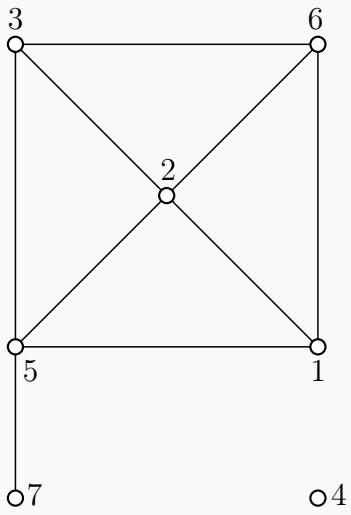
Матрицата на съседство на граф G е абсолютно същото нещо като матричното представяне на съответната бинарна релация. Тоест, ако мислим графа като за бинарна релация над $\{1, \ldots, n\}^2$, която е симетрична (G е неориентиран) и антирефлексивна (G няма примки), то

матрицата на съседство е точно същата матрица, която съответства на релацията. А именно, квадратна, $n \times n$, симетрична булева матрица M с нули по главния диагонал, в която за $i, j \in \{1, \ldots, n\}$, клетка $M[i, j]$ съдържа:

- 0, ако в G няма ребро (i, j), и
- 1, ако в G има ребро (i, j).

Фигура 4.1 показва граф и неговата матрица на съседство.

Фигура 4.1 : Матрица на съседство на граф.



граф G

	1	2	3	4	5	6	7
1	0	1	0	0	1	1	0
2	1	0	1	0	1	1	0
3	0	1	0	0	1	1	0
4	0	0	0	0	0	0	0
5	1	1	1	0	0	0	1
6	1	1	1	0	0	0	0
7	0	0	0	0	1	0	0

матрицата на съседство на G

Ето няколко важни свойства на матриците на съседство на обикновените графи. Нека е даден обикновен граф G и M е неговата матрица на съседство. Тук, както и навсякъде другаде в Подсеция 4.1.1, когато казваме "път", имаме предвид път, който не е непременно прост.

1. M е симетрична, което вече бе споменато.
2. За всяко i, сумата от елементите в ред i, или в колона i, е равна на степента на връх i.
3. Матрицата на съседство на празния граф с n върха е нулевата $n \times n$ матрица.
4. G е свързан тстк няма пермутация на редовете и колоните, която да преобразува M в блокова диагонална форма.
5. M_1 и M_2 са матрици на съседство на изоморфни графи тстк съществува *пермутационна матрица* P, такава че $M_2 = P^{-1} M_1 P$.
6. При повдигане на M на степен k, клетка $[i, j]$ съдържа броя на не непременно простите пътища между i и j с дължина k.
7. Следствие от Свойство 6 е, че $\forall i : M^2[i, i] = d(i)$. Причината е ясна: всяко ребро $e = (u, v)$ представлява път u, e, v, e, u, който не е прост, с дължина 2 от u до u.

Свойство 6 следва от Теорема 47. Теорема 47 е за ориентирани графи, но, ако я приложим към ориентирания граф, съответен на G (припомнете си Определение 85), получаваме Свойство 6.

Ако разрешим наличието на примки, има две възможности за записването на примка в матрицата на съседство. Да кажем, че връх i има примка.

- Първо, можем да запишем 1 в $M[i,i]$. Тази възможност точно съответства на отбелязването, в контекста на бинарните релации $R \subseteq A^2$, че даден елемент $a \in A$ е в релация със себе си, ако представяме R с матрица.

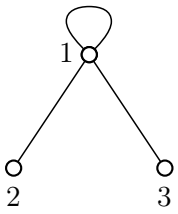
  Обаче Свойство 2 престава да е вярно! В Определение 15 казахме, че всяка примка примка се брои **два** пъти в степента на върха, към който е "вързана". Ако броим приносът на примката към степента на върха, към който е "вързана", като единица, ще "счупим" Лема 3.

  Свойства 6 и 7 остават в сила.

- Второ, може да запишем 2 в $M[i,i]$. При това матрицата не би била булева.

  В такъв случай Свойство 2 би останало в сила и не "чупим" Лема 3.

  Свойства 6 и 7 престават да са в сила. Като малък пример разгледайте следния граф с примка, неговата матрицата M на съседство и нейната втора степен M^2.



M

	1	2	3
1	2	1	1
2	1	0	0
3	1	0	0

M^2

	1	2	3
1	6	2	2
2	2	1	1
3	2	1	1

  Примката е записана с "2" в клетка $M[1,1]$. Обаче $M^2[1,1]$ съдържа 6, което не е степента на връх 1. Неговата степен е 4.

Няма начин да излезем от това затруднение, причинено от наличието на примки, ако хем искаме Лема 3 да е в сила и сумата по ред/колона да дава степента, хем искаме повдигането на матрицата на степен да дава броеве на пътища с определена дължина.

Абстрактно погледнато, всички (разумни) представяния на графи са еквивалентни, защото всяко от тях представя графите недвусмислено и можем да конвертираме едно представяне в друго по избор. Практически погледнато, всяко представяне има своите предимства и недостатъци.

Основните предимства на представянето с матрица на съседства са:

- бързата проверка дали между връх i и връх j има ребро;
- бързото добавяне на ребро между връх i и връх j, ако такова няма;
- бързото изтриване на реброто между връх i и връх j, ако такова има.

Проверката за ребро става с един единствен достъп до матрицата – до клетка $[i,j]$ или $[j,i]$ (няма значение коя, матрицата е симетрична). Добавянето и изтриването стават с два достъпа, ако $i \neq j$: променя се съдържанието на $[i,j]$ и на $[j,i]$.

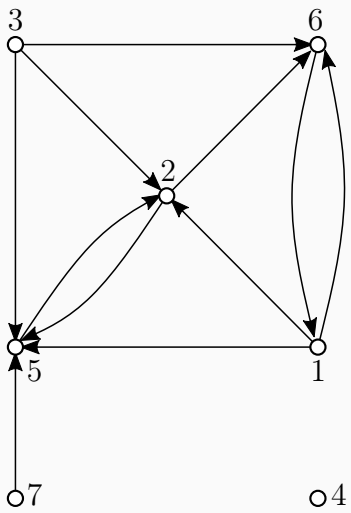
Недостатък на представянето с матрица на съседства е потенциалната му неикономичност. Ако n е голямо, но m е далече по-малко от максималната си стойност $\frac{1}{2}n(n-1)$, матрицата става огромна, понеже размерът ѝ е n^2 клетки, а съдържанието ѝ е основно от нули. Въпреки че всяка клетка ползва само един бит, фактът, че общо клетките са n^2, доминира в това съображение. Съществуват важни класове графи като:

- дърветата, при които $m = n - 1$;
- планарните графи, при които $m \leqslant 3n - 6$;
- графите с малка максимална степен на връх[†].

за които можем да кажем, че m е далече по-малко от максимума $\frac{1}{2}n(n-1)$. За графите от тези класове е ясно, че, ако ги представяме с матрици и n е много голямо, огромната част от всички n^2 клетки ще е запълнена с нули и тук-там ще има по някоя единица.

Ориентиран граф G се представя с матрица на съседство M по подобен начин. Клетката $M[i,j]$ съдържа 1 тстк в G има ребро (i,j). Фигура 4.2 показва ориентиран граф и неговата матрица на съседство.

Фигура 4.2 : Матрица на съседство на ориентиран граф.



ориентиран граф G'

	1	2	3	4	5	6	7
1	0	1	0	0	1	1	0
2	0	0	0	0	1	1	0
3	0	1	0	0	1	1	0
4	0	0	0	0	0	0	0
5	0	1	0	0	0	0	0
6	1	0	0	0	0	0	0
7	0	0	0	0	1	0	0

матрицата на съседство на G'

Ето няколко важни свойства на матриците на съседство на ориентираните графи. Нека е даден ориентиран граф G и M е неговата матрица на съседство.

1. M не е непременно симетрична.
2. За всяко i, сумата от елементите в ред i е равна на изходната степен на връх i (вижте Определение 81).
3. За всяко i, сумата от елементите в колона i е равна на входната степента на връх i (вижте Определение 81).

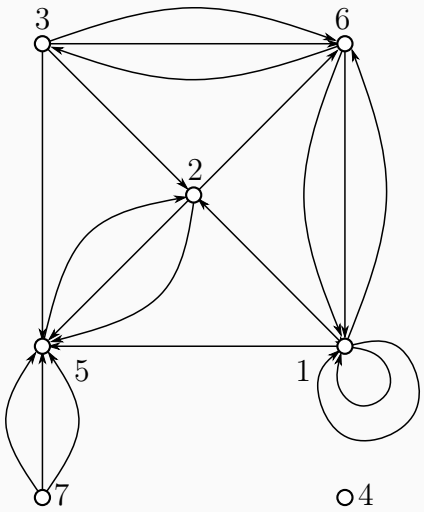
[†]Имайте предвид, че $m \leqslant \frac{1}{2}n\Delta(G)$ за всеки граф G.

4. При повдигане на M на степен k, клетка $[i, j]$ съдържа броя на (ориентираните) пътища от i до j с дължина k.

Четвъртото свойство и този път следва от Теорема 47, но сега е директно, понеже G се явява частен случай на ориентиран мултиграф.

Мултиграфите също може да се представят с матрици на съседство, като обаче при тях матриците не са булеви, а от естествени числа. Ето как става това за ориентирани мултиграфи. Ако мултиграфът е $G = (V, E, f_G)$ според Определение 86, то матрицата му на съседство е $n \times n$ матрица от естествени числа, като $M[i, j] = |\{e \in E : f_G(e) = (i, j)\}|$. Фигура 4.3 показва ориентиран мултиграф граф G'' и неговата матрица на съседство.

Фигура 4.3 : Матрица на съседство на ориентиран мултиграф.



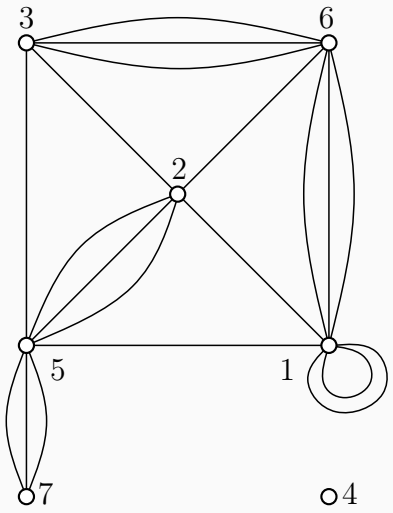
ориентиран мулгиграф граф G''

	1	2	3	4	5	6	7
1	2	1	0	0	1	1	0
2	0	0	0	0	2	1	0
3	0	1	0	0	1	2	0
4	0	0	0	0	0	0	0
5	0	1	0	0	0	0	0
6	2	0	1	0	0	0	0
7	0	0	0	0	3	0	0

матрицата на съседство на G''

За неориентирани мултиграфи нещата са сходни. Ако мултиграфът е $G = (V, E, f_G)$ според Определение 14, то матрицата му на съседство е $n \times n$ матрица от естествени числа, като $M[i, j] = |\{e \in E : f_G(e) = \{i, j\}\}|$. Забележете, че това определение третира примките правилно, защото, ако $i = j$, то $\{i, j\} = \{i\}$, а кодомейнът на f_G според Определение 14 включва и едноелементните подмножества на V. Фигура 4.4 показва неориентиран мултиграф и неговата матрица на съседство.

Фигура 4.4 : Матрица на съседство на неориентиран мултиграф.



неориентиран мулгиграф граф G'''

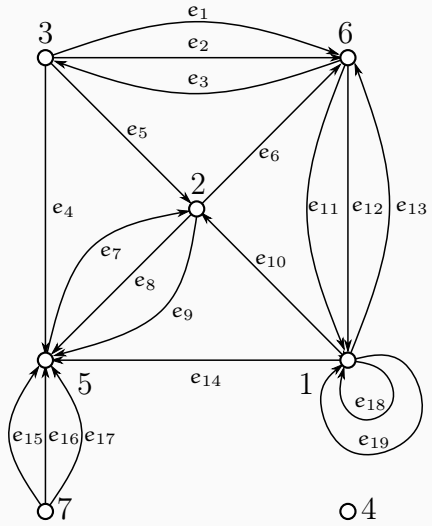
	1	2	3	4	5	6	7
1	2	1	0	0	1	3	0
2	1	0	1	0	3	1	0
3	0	1	0	0	1	3	0
4	0	0	0	0	0	0	0
5	1	3	1	0	0	0	3
6	3	1	3	0	0	0	0
7	0	0	0	0	3	0	0

матрицата на съседство на G'''

Както се вижда и от фигурата, при неориентираните мултиграфи матрицата е симетрична.

И накрая ще видим едно любопитно свойство на матриците на съседство. Като пример да разгледаме отново G'' от Фигура 4.3, но този път с имена на ребрата, и неговата матрица на съседство M, показани на Фигура 4.5.

Фигура 4.5 : G'' и неговата матрица на съседство M.



G''

	1	2	3	4	5	6	7
1	2	1	0	0	1	1	0
2	0	0	0	0	2	1	0
3	0	1	0	0	1	2	0
4	0	0	0	0	0	0	0
5	0	1	0	0	0	0	0
6	2	0	1	0	0	0	0
7	0	0	0	0	3	0	0

M

По дефиниция $\mathsf{M}[\mathtt{i},\mathtt{j}]$ е броят на ребрата с начало $\mathtt{i}$ и край $\mathtt{j}$; с други думи, пътищата с

дължина 1 от i до j. Да разгледаме произведението на M със себе си, тоест M^2:

$$\begin{bmatrix} 2 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 2 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 2 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 2 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 6 & 3 & 1 & 0 & 4 & 3 & 0 \\ 2 & 2 & 1 & 0 & 0 & 0 & 0 \\ 4 & 1 & 2 & 0 & 2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 1 & 0 \\ 4 & 3 & 0 & 0 & 3 & 4 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Лесно се вижда, че $M^2[i,j]$ е броят на пътищата с дължина 2 от i до j, по всички i и j. Примерно, има точно 3 пътя с дължина 2 от връх 1 до връх 6:

1, e_{10}, 2, e_6, 6
1, e_{18}, 1, e_{13}, 6
1, e_{19}, 1, e_{13}, 6

а от друга страна, $M^2[1,6] = 3$. Това не е случайно съвпадение! Въпросното 3 се получава като скаларно произведение на първия ред на M с шестата колона на M:

$$[2,1,0,0,1,1,0] \cdot [1,1,2,0,0,0,0] = 2 \cdot 1 + 1 \cdot 1 + 0 \cdot 2 + 0 \cdot 0 + 1 \cdot 0 + 1 \cdot 0 + 0 \cdot 0 = 3$$

Щом първият ред съдържа броя на ребрата от връх 1 до всеки друг връх, а шестата колона съдържа броя на ребрата от всеки връх до връх 6, то първото събираемо $2 \cdot 1$ е броят на пътищата с точно две ребра от 1 до 6 с вътрешен връх 1, второто събираемо $1 \cdot 1$ е броят на пътищата с точно две ребра от 1 до 6 с вътрешен връх 2, третото събираемо $0 \cdot 2$ е броят на пътищата с точно две ребра от 1 до 6 с вътрешен връх 3, и така нататък. Както вече видяхме, наистина има два пътя с дължина 2 с връх 1 като вътрешен, един път с дължина 2 с връх 2 като вътрешен, нула пътища с дължина 2 с връх 3 като вътрешен, и така нататък.

С аналогични разсъждения установяваме, че M^3 съдържа броевете на пътищата с дължина 3:

$$M^3 = \begin{bmatrix} 18 & 11 & 3 & 0 & 13 & 11 & 0 \\ 4 & 3 & 0 & 0 & 7 & 6 & 0 \\ 10 & 8 & 1 & 0 & 8 & 9 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 2 & 1 & 0 & 0 & 0 & 0 \\ 16 & 7 & 4 & 0 & 10 & 7 & 0 \\ 0 & 0 & 0 & 0 & 6 & 3 & 0 \end{bmatrix}$$

Да вземем $M^3[1,6] = 11$. Наистина, има 11 пътя с дължина 3 от 1 до 6:

1, e_{18}, 1, e_{18}, 1, e_{13}, 6
1, e_{18}, 1, e_{19}, 1, e_{13}, 6
1, e_{19}, 1, e_{18}, 1, e_{13}, 6
1, e_{19}, 1, e_{19}, 1, e_{13}, 6
1, e_{13}, 6, e_{11}, 1, e_{13}, 6
1, e_{13}, 6, e_{12}, 1, e_{13}, 6
1, e_{18}, 1, e_{10}, 2, e_6, 6
1, e_{19}, 1, e_{10}, 2, e_6, 6
1, e_{14}, 5, e_7, 2, e_6, 6
1, e_{13}, 6, e_3, 3, e_1, 6
1, e_{13}, 6, e_3, 3, e_2, 6

И така нататък. Сега ще докажем строго интуицията, получена от тези наблюдения.

Теорема 47: $M^k[i,j]$ **е броят на ориентираните пътища с дължина** k

За всеки ориентиран мултиграф G и за всяко $k \geqslant 0$ е вярно, че $M^k[i,j]$ е броят на (ориентираните) пътища с дължина k от i до j в G, където M е матрицата на съседство на G.

Доказателство: С индукция по k. Базата е за $k = 0$. Но M^0 е единичната матрица с единици по главния диагонал и нули извън главния диагонал. Лесно се вижда, че за всички $i, j \in \{1, \ldots, n\}$, $M^0[i,j]$ наистина е броят на пътищата с дължина 0 от i до j:

- ако $i \neq j$, такива пътища няма, което точно отговаря на факта, че $M^0[i,j] = 0$ при $i \neq j$;
- ако $i = j$, има точно един такъв път, а именно самият връх i, което точно отговаря на факта, че $M^0[i,i] = 1$.

С това базата е доказана. ✓

Да допуснем, че твърдението е вярно за стойност на аргумента k. Ще докажем, че то остава вярно за стойност на аргумента $k+1$. Нека $T^\ell_{i,j}$ означава множеството от пътищата от i до j с дължина ℓ, за всяко $\ell \geqslant 0$. Ще докажем, че $M^k[i,j] = |T^k_{i,j}|$ влече $M^{k+1}[i,j] = |T^{k+1}_{i,j}|$, за всички $i, j \in \{1, \ldots, n\}$.

- За лявата страна $M^{k+1}[i,j]$ знаем от линейната алгебра, че

$$M^{k+1}[i,j] = \sum_{1 \leqslant s \leqslant n} M^k[i,s] \cdot M[s,j] \tag{4.1}$$

- Да разгледаме дясната страна $|T^{k+1}_{i,j}|$. Да разгледаме $T^{k+1}_{i,j}$. Тъй като $k+1 \geqslant 1$, всеки път от това множество има предпоследен връх, при това точно един. Това е очевидно. Тогава $T^{k+1}_{i,j}$ се разбива на n множества (говорим за обобщено разбиване – някои от тези n множества може да са празни) по предпоследен връх.

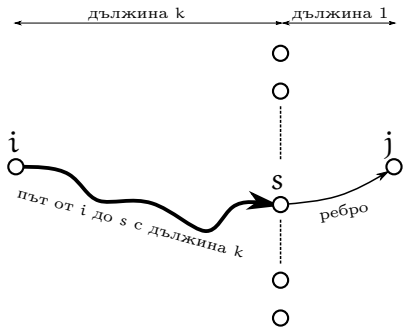
  Да въведем още нотация, за да изразим същото нещо формално. Нека $W^\ell_{i,j,s}$ означава множеството от пътищата от i до j с предпоследен връх s и дължина ℓ, където $\ell \geqslant 1$, а $i, j, s \in \{1, \ldots, n\}$. И така, забелязахме, че $T^{k+1}_{i,j}$ има разбиване

$$\{W^{k+1}_{i,j,1}, W^{k+1}_{i,j,2}, \ldots, W^{k+1}_{i,j,n}\}$$

където някои или дори всички дялове на това разбиване може да са празни. Съгласно комбинаторния принцип на разбиването,

$$|T_{i,j}^{k+1}| = \sum_{1 \leqslant s \leqslant n} |W_{i,j,s}^{k+1}| \tag{4.2}$$

Ето илюстрация на (4.2) и по-точно на разбиването на множеството от пътищата с дължина $k+1$ по предпоследния връх s:



Тази илюстрация обаче е леко подвеждаща, защото показва i, s и j като два по два различни върхове. А това не е непременно така. Първо, може $i = j$. Второ, независимо от това дали $i = j$ или не, може $i = s$ или $j = s$. Поради това има смисъл индексната променлива s в (4.2) да взема стойности от 1 до n. Ако за някоя такава стойност s не може да е предпоследен връх (примерно, на Фигура 4.5, връх 2 не може да е предпоследен по път до връх 1, защото няма ребро от 2 до 1, а връх 4 не може да е предпоследен по път до връх 1, защото от 4 до 1 изобщо няма пътища), то $W_{i,j,s}^{k+1} = \varnothing$.

След като се убедихме, че (4.2) е вярно, да съобразим, че за всяко $s \in \{1, \ldots, n\}$, $W_{i,j,s}^{k+1}$ (независимо дали е празно или не) се явява, в някакъв смисъл, Декартовото произведение на $T_{i,s}^{k}$ и множеството от ребрата от s до j. Тук аргументацията е леко неформална, защото не казваме точно в какъв смисъл, но важното е, че всеки път от $W_{i,j,s}^{k+1}$ се явява комбинация на един път от $T_{i,s}^{k}$ и едно ребро от s до j. Тогава $|W_{i,j,s}^{k+1}|$ е произведението от $|T_{i,s}^{k}|$ и броя на ребрата от s до j.

Да повторим: броят на начините да стигнем от i до j с точно $k+1$ ребра и s за предпоследен връх е произведението от:

- броя на начините да стигнем от i до s с точно k ребра и
- броя на начините да стигнем от s до j с точно 1 ребро.

Имайки предвид, че броят на ребрата от s до j е $M[s,j]$, показахме, че

$$|W_{i,j,s}^{k+1}| = |T_{i,s}^{k}| \cdot M[s,j] \tag{4.3}$$

От (4.2) и (4.3) извеждаме

$$|T_{i,j}^{k+1}| = \sum_{1 \leqslant s \leqslant n} |T_{i,s}^{k}| \cdot M[s,j] \tag{4.4}$$

Но от индуктивното предположение знаем, че $|T_{i,s}^{k}| = M^k[i,s]$. Тогава

$$|T_{i,j}^{k+1}| = \sum_{1 \leqslant s \leqslant n} M^k[i,s] \cdot M[s,j] \tag{4.5}$$

Сравнявайки десните страни на (4.1) и (4.5), заключаваме, че $M^{k+1}[i,j] = |T_{i,j}^{k+1}|$. □

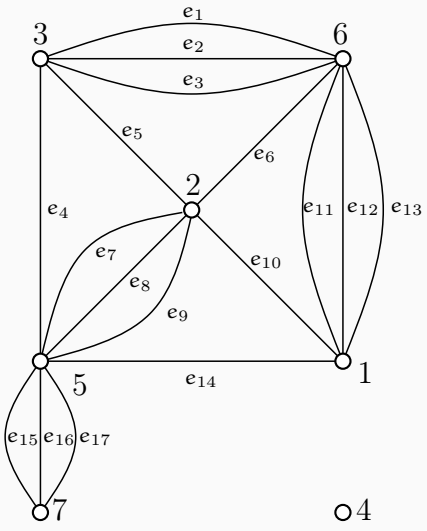
4.1.2 Матрици на инцидентност

Неориентирани графи и мултиграфи без примки. Нека е даден обикновен граф или неориентиран мултиграф без примки G, като $V(G) = \{1, \ldots, n\}$ и $E(G) = \{e_1, \ldots, e_m\}$. *Матрицата на инцидентност* на G е $n \times m$ матрица A, в която за $i \in \{1, \ldots, n\}$ и $j \in \{1, \ldots, m\}$, $A[i, j]$ съдържа:

- 0, ако връх i не е инцидентен с ребро e_j,
- 1, ако връх i е инцидентен с ребро e_j.

Фигура 4.6 показва неориентиран мултиграф без примки и неговата матрица на инцидентност. Типично, матрицата на инцидентност има n реда и m колони. На фигурата е показана транспонираната матрица на инцидентност, за да може фигурата да се побере на ширина.

Фигура 4.6 : Неор. мултиграф без примки и матр. на инцидентност.



мултиграф без примки

	1	2	3	4	5	6	7
1	0	0	1	0	0	1	0
2	0	0	1	0	0	1	0
3	0	0	1	0	0	1	0
4	0	0	1	0	1	0	0
5	0	1	1	0	0	0	0
6	0	1	0	0	0	1	0
7	0	1	0	0	1	0	0
8	0	1	0	0	1	0	0
9	0	1	0	0	1	0	0
10	1	1	0	0	0	0	0
11	1	0	0	0	0	1	0
12	1	0	0	0	0	1	0
13	1	0	0	0	0	1	0
14	1	0	0	0	1	0	0
15	0	0	0	0	1	0	1
16	0	0	0	0	1	0	1
17	0	0	0	0	1	0	1

транспонираната матр. на инцид.

Истинската, а не транспонираната матрица на инцидентност на графа от Фигура 4.6 е

$$\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1
\end{bmatrix}$$

Матрицата на инцидентност третира ребрата като атомарни обекти, също както е в Определение 13, а не ги идентифицира с краищата, както е в Определение 1. Поради това мултиграфите (но без примки) се представят с матрици на инцидентност по абсолютно същия начин, по който се представят и обикновените графи.

Матриците на инцидентност представят графи, в които са именувани както ребрата, така и върховете; разменянето на имена на ребра променя, в общия случай, и матрицата. За разликата между това да са именувани само върховете или и върховете, и ребрата вижте подсекциите на стр. 83 и на стр. 85.

Матриците на инцидентност имат следните свойства.

1. Матриците на инцидентност са булеви, като във всяка колона има точно две единици.

2. Ред само с нули съответства на изолиран връх. В примера на Фигура 4.6, връх 4 е изолиран и съответно ред четири на матрицата има само нули.

   Аналогично, ред с точно една единица съответства на висящ връх (на примера от Фигура 4.6 няма такъв).

   И изобщо, броят на единиците на ред $\mathtt{i}$ е точно $\mathtt{d}(\mathtt{i})$.

3. Празният граф има празна матрица на инцидентност—тъй като няма ребра—независимо от броя на върховете

4. Размяната на редове $\mathtt{i}$ и $\mathtt{j}$ има смисъл на размяна на имената на върхове $\mathtt{i}$ и $\mathtt{j}$. Размяната на колони $\mathtt{i}$ и $\mathtt{j}$ има смисъл на размяна на имената на ребрата $e_\mathtt{i}$ и $e_\mathtt{j}$.

5. На паралелни ребра в графа съответстват еднакви колони в матрицата.

6. Всяка матрица има "излишък" в смисъл, че всеки $n-1$ реда напълно определят съответния граф. Може да се обоснове с това, че всеки ред, поелементно, се получава от сумата на останалите редове по модул 2.

   Иначе казано, ако скрием, примерно, първия ред на матрицата на графа от Фигура 4.6, получавайки тази матрица:

$$\begin{bmatrix}
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1
\end{bmatrix}$$

   лесно можем да възстановим скрития ред, забелязвайки, че точно колони 10, ..., 14 имат по едно "1". Това означава, че точно ребра e_{10}, ..., e_{14} са инцидентни с връх 1.

   Ако гледаме на матриците на инцидентност като на матрици над $\mathsf{GF}(2)$, току-що направеното наблюдение казва, че множеството от редовете е линейно зависимо. Оттук рангът на матрицата е по-малък или равен на $n-1$.

7. Ако графът има две свързани компоненти G_1 и G_2, то матрицата на инцидентност A може да се запише в блокова диагонална форма

$$\begin{bmatrix}
A_1 & 0 \\
0 & A_2
\end{bmatrix}$$

където A_1 и A_2 са матриците на инцидентност на G_1 и G_2. Причината е ясна: нито едно ребро от $E(G_1)$ не е инцидентно с връх от $V(G_2)$, и обратно.

Очевидно е как да се обобщи това наблюдение за k свързани компоненти.

Лема 22

Ако G е свързан граф и матрицата му на инцидентност е A, то A **не може** да се запише по никакъв начин в блокова диагонална форма

$$\begin{bmatrix} A_1 & 0 \\ 0 & A_2 \end{bmatrix}$$

Доказателство: Да допуснем противното: матрицата на инцидентност на свързан граф може да се запише в блокова диагонална форма, каквато е показана горе.

Нека i е произволен връх "от горната част" на матрицата; тоест, ред номер i "минава" през A_1. И нека j е произволен връх от долната част на матрицата; тоест, ред номер j "минава" през A_2. Тъй като G е свързан, между i и j има път p. Всеки връх на p е или от горната, или от долната част на матрицата. Тогава в p съществуват съседни върхове i', който в матрицата е в горната част (тоест, ред i' "минава" през A_1), и j', който в матрицата е в долната част (тоест, ред j' "минава" през A_2). Да кажем, че реброто e има за краища i' и j'. Да кажем, че колона ℓ съответства на e.

Както знаем, в колона ℓ има точно две единици. Ще покажем, че не може и двете единици да се намират в A_1 или A_2.

- Ако колона ℓ е в лявата страна на матрицата (тоест, "минава" през A_1), то едната от тези единици е в долната лява подматрица, която би трябвало да се състои само от нули.
- Ако колона ℓ е в дясната страна на матрицата (тоест, "минава" през A_2), то едната от тези единици е в горната дясна подматрица, която би трябвало да се състои само от нули.

И в двата случая виждаме, че A не е в блокова диагонална форма. □

Теорема 48: Рангът на матрицата на инцидентност на свързан граф

Матрицата на инцидентност на свързан граф има ранг $n - 1$.

Доказателство: Както вече отбелязахме, рангът не може да надхвърля $n - 1$ за никой граф. Остава да покажем, че рангът е поне $n - 1$, ако графът е свързан.

Разглеждаме произволни k реда на A, където $1 \leqslant k \leqslant n - 1$. Лема 22 е ключова: A **не може** да се запише в блокова диагонална форма

$$\begin{bmatrix} A_1 & 0 \\ 0 & A_2 \end{bmatrix}$$

където A_1 има k реда. Тогава никои k реда не се сумират до ред от нули (по модул 2).

Но $GF(2)$ има само две константи: 0 и 1. Тогава всички възможни сумирания на k реда (по модул 2), за $k \in \{1, \ldots, n-1\}$, са всички възможни линейни комбинации на редовете. И, както казахме, нито една от тях не е нулевият ред. Тогава нито една линейна комбинация на k реда, за $k \in \{1, \ldots, n-1\}$, не е нулевият ред. Тогава всеки k реда, за $k \in \{1, \ldots, n-1\}$, са линейно независими. Тогава рангът на A е $n-1$. □

Следствие 15: Рангът на матрицата на инцидентност на свързан граф

Ако G е граф с k свързани компоненти, рангът на неговата матрица на инцидентност е $n-k$. □

Следствие 15 може да се използва за намиране на броя на свързаните компоненти, но при това трябва да се внимава: изчисленията върху матрицата трябва да са по модул 2! Иначе казано, матрицата е над $GF(2)$, а **не над полето на реалните числа**. Ако ползваме софтуер за работа с математика като Maple(TM), трябва да укажем да се работи в $GF(2)$; от това, че матрицата съдържа само нули и единици, софтуерът няма да се "сети" да не работи с реални числа. Като пример да вземем матрицата на инцидентност на Фигура 4.6. Ето как можем да я въведем и да ѝ намерим ранга в Maple(TM):

```
C:\Program Files\Maple 2018\bin.X86_64_WINDOWS>cmaple
    |\^/|     Maple 2018 (X86 64 WINDOWS)
._|\|   |/|_. Copyright (c) Maplesoft, a division of Waterloo Maple Inc. 2018
 \  MAPLE  /  All rights reserved. Maple is a trademark of
 <____ ____>  Waterloo Maple Inc.
      |       Type ? for help.
> with(LinearAlgebra):
> A := Matrix( [ [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0],
>                [0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0],
>                [1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
>                [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
>                [0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1],
>                [1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0],
>                [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1] ]):
> Rank(A);
                                                    6
```

Отговорът, който Maple(TM) връща, е 6. Но броят на свързаните компоненти е 2, а $n = 7$, така че според Следствие 15, рангът трябва да е $7 - 2 = 5$. Това размнаване се дължи на факта, че Maple(TM) не работи в $GF(2)$ по подразбиране. Ерго, за Maple(TM) в тези изчисления $1 + 1 = 2$, а не $1 + 1 = 0$, както би трябвало в $GF(2)$.

Нещата може да се оправят така.

```
C:\Program Files\Maple 2018\bin.X86_64_WINDOWS>cmaple
    |\^/|     Maple 2018 (X86 64 WINDOWS)
._|\|   |/|_. Copyright (c) Maplesoft, a division of Waterloo Maple Inc. 2018
 \  MAPLE  /  All rights reserved. Maple is a trademark of
 <____ ____>  Waterloo Maple Inc.
      |       Type ? for help.
> with(LinearAlgebra[Modular]):
> A := Matrix( [ [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0],
```

```
>                [0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0],
>                [1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
>                [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
>                [0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1],
>                [1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0],
>                [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1] ]):
> Rank(2, A);
                                                                  5
```

Сега вече рангът е 5, както очакваме. Първият аргумент на командата `Rank` е модулът.

Неориентирани графи и мултиграфи с примки. Наличието на примки е потенциален препъни-камък и за матриците на инцидентност. Ако реброто e_i е примка, "закачена" към връх j, какво да сложим в $A[j,i]$?

- Някои учебници като [31, стр. 102] казват, че има смисъл да сложим 2. Тогава остава в сила това, че сумата във всяка колона е 2, и че сумата във всеки ред е равна на степента на съответния връх.

  Обаче матрицата вече не е булева и елементите ѝ не са от $\mathsf{GF}(2)$ и редица свойства престават да са в сила.

- Други учебници като [3, стр. 139] разглеждат повече възможности.

  - Може да сложим 0, при което матрицата остава над $\mathsf{GF}(2)$, но това означава да има колона само от нули (съответстваща на реброто-примка) и тогава, гледайки само матрицата, не знаем къде е примката.
  - Може да сложим 1, при което матрицата остава над $\mathsf{GF}(2)$, но сега вече рангът ѝ не е $n-1$, а става n. При това важни свойства на матрицата престават да са в сила.

    Ако гледаме на графа като на частен случай на хиперграф, реброто e_i, инцидентно със и само със връх j, се идентифицира с едноелементното множество $\{j\}$, и $A[j,i]$ трябва да е 1. Да си припомним, че хиперграфите имат представяне с матрици на инцидентност (вижте материала на стр. 243).
  - Може да сложим 2, която възможност вече обсъдихме.

В крайна сметка казваме, че матриците на инцидентност не са подходящо представяне за графи с примки и съответно няма да използваме матрици на инцидентност, когато примки са възможни.

Ориентирани графи и мултиграфи. Матриците на инцидентност могат да представят и ориентирани графи. Нека е даден обикновен ориентиран граф или ориентиран мултиграф без примки G, като $V(G) = \{1, \ldots, n\}$ и $E(G) = \{e_1, \ldots, e_m\}$. Сега обаче елементите на матриците са реални числа, а не елементи на $\mathsf{GF}(2)$. Това означава, че сега $1 + 1 = 2$, а не $1 + 1 = 0$.

Матрицата на инцидентност на G е $n \times m$ матрица A с елементи от $\{-1, 0, 1\}$, в която за $i \in \{1, \ldots, n\}$ и $j \in \{1, \ldots, m\}$, $A[i,j]$ съдържа:

- 0, ако връх i не е инцидентен с ребро e_j,
- 1, ако връх i е началото на ребро e_j.
- -1, ако връх i е краят на ребро e_j.

В [3, стр. 139], [17, стр. 214] и [25, стр. 78] конвенцията за знаците е такава. В [31, стр. 102] е точно обратното: -1 маркира началото на реброто, а 1 маркира края му. Двете конвенции са еднакво добри, но са очевидно несъвместими. Ние избираме конвенцията, че 1 маркира началото, а -1, края на реброто.

Доказателството на Теорема 49 е аналогично на доказателството на Теорема 48, въпреки че Теорема 49 е за ориентирани, а Теорема 48, за неориентирани графи, поради което ще го прескочим. Има го в, примерно, в [17, стр. 214].

Теорема 49: Рангът на матрицата на инцидентност на слабо свързан ориентиран граф

Матрицата на инцидентност на слабо свързан ориентиран граф има ранг $n - 1$.

И оттук лесно се извежда съответствието на Следствие 15: матрицата на инцидентност на ориентиран граф с k слабо свързани компоненти има ранг $n - k$. □

4.1.3 Списъци на съседство

Това е много компактен начин за представяне на графи с голямо приложение в практиката. На английски терминът е *adjacency lists*. Забележете множественото число: не един списък, а много списъци.

Неориентирани мултиграфи с възможни примки. Примките не причиняват никакви затруднения при представяне със списъци на съседства, а мултиграфите се представят толкова естествено, колкото и обикновените графи, така че направо ще разгледаме най-общия неориентиран случай на мултиграфи с възможни примки.

За всеки връх на графа се конструира свързан списък от неговите съседи **в произволен ред**. Това си заслужава да се повтори: не иска съседите да се появяват в нарастващ ред по своите идентификатори.

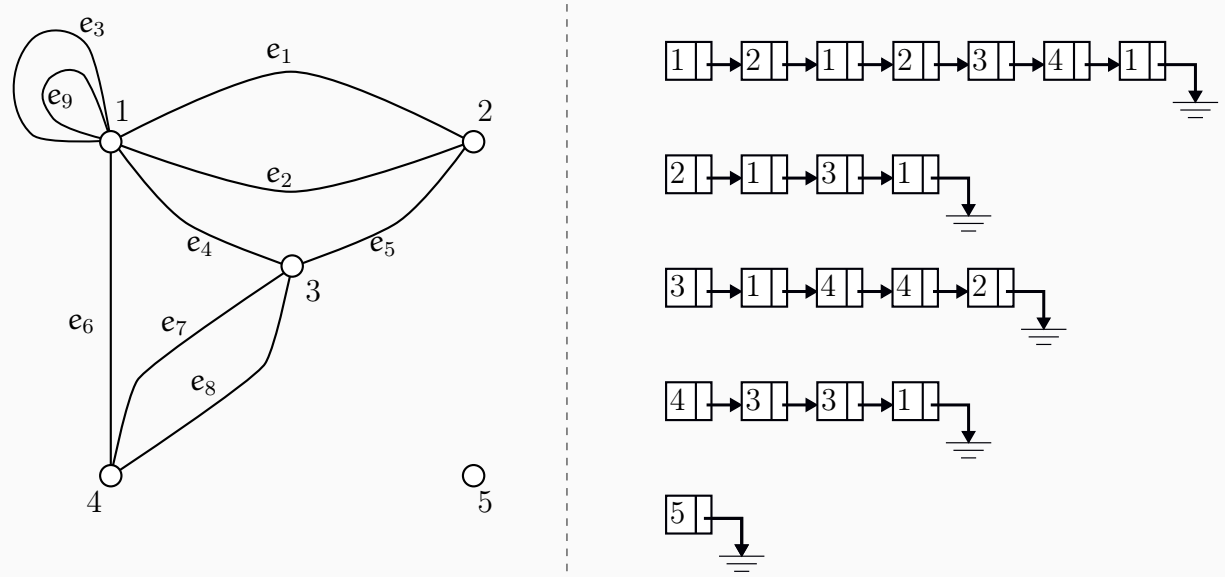
Ако даден връх няма съседи, тоест, е изолиран връх, то списъкът му е празен.

За всяко ребро с краища i и j, такива че $i \neq j$, връх j се появява в списъка на i, но и връх i се появява в списъка на j. Ерго, на всяко ребро, което не е примка, съответстват точно два елемента, в два различни списъка. Ако има сноп от точно k паралелни ребра с краища i и j, то j се появява точно k пъти в списъка на i, но и i се появява точно k пъти в списъка на j.

За всяко ребро-примка, ако i е върхът, към който е "закачена" примката, то i се появява в собствения си списък; ако i има точно k примки, то i се появява точно k пъти в собствения си списък.

Фигура 4.7 показва неориентиран мултиграф и списъците му на съседство.

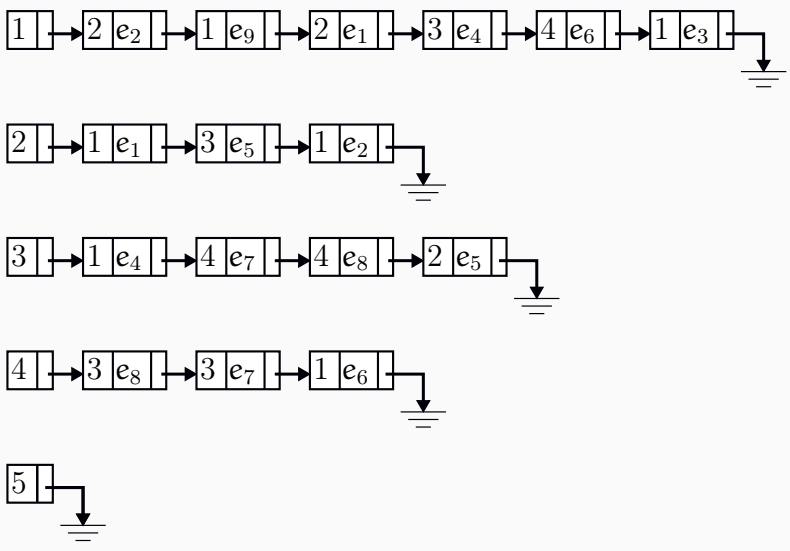
Фигура 4.7 : Списъците на съседство на неориентиран мултиграф.



неориентиран мултиграф | неговите списъци на съседство

Тези списъци не съдържат информация, идентифицираща ребрата. Възможно е елементите на списъците да съдържат информация за ребрата, както е показано на Фигура 4.8.

Фигура 4.8 : По-информативни списъци на съседства на графа от Фиг. 4.7.



Ако става дума за графи с тегла (Секция 3.2), естествено е да записваме теглата на ребрата в списъците на съседство, като всеки елемент на списък има клетка, в която се записва числото-тегло на съответното ребро.

В практически приложения може елементите на списъците да са дори още по-информативни. Ако графът представлява пътна мрежа, елементите на списъците описват някакви шосета, всяко от които има атрибути като категория, дата на последен ремонт и така нататък. Ако графът представлява компютърна мрежа, елементите на списъците описват някакви

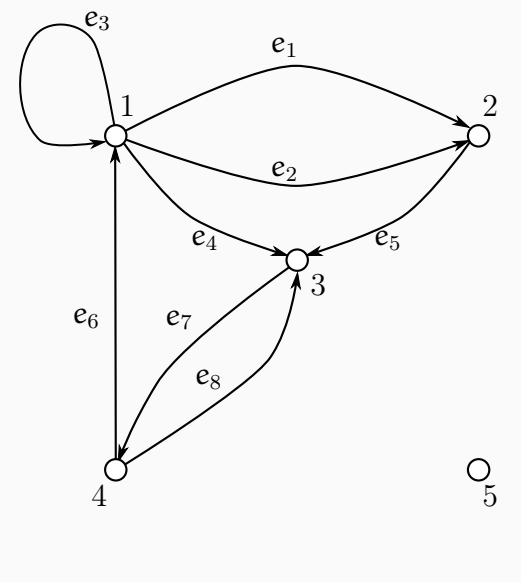
директни връзки между хостове, като тези директни връзки имат атрибути като капацитет за пренасяне на информация, вид (усукана двойка, оптика и така нататък), дължина и така нататък. Възможно е елементите на списъците да са записи с полета, които са подходящи за съответните атрибути.

Ориентирани мултиграфи с възможни примки. Ориентирани графи или мултиграфи, с или без възможни примки, се представят лесно със списъци на съседство: списъкът на всеки връх съдържа неговите деца в произволен ред. Списъкът на връх е празен тстк този връх няма деца.

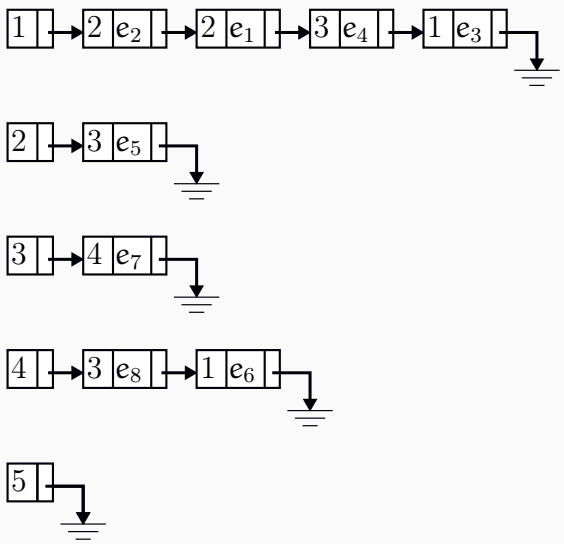
Забележете, че в ориентирания случай на всяко ребро съответства точно един елемент в списъците, за разлика от неориентирания случай, в който—както казахме горе—на всяко ребро, което не е примка, съответстват два елемента в списъците.

Фигура 4.9 показва ориентиран мултиграф и списъците му на съседство, като елементите на списъците съдържат имената на ребрата.

Фигура 4.9 : Списъците на съседство на ориентиран мултиграф.



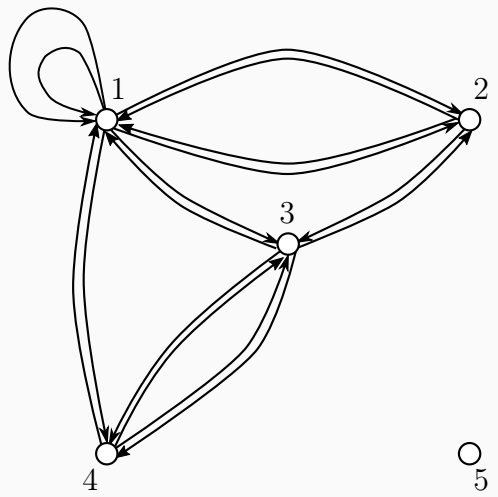
ориентиран мултиграф



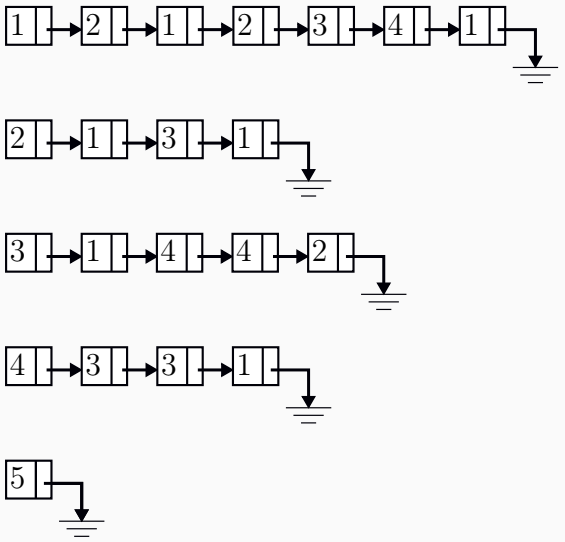
неговите списъци на съседство

Да разгледаме ориентирания мултиграф, показан на Фигура 4.10, заедно със списъците му на съседство.

Фигура 4.10 : Списъците на съседство на друг ориентиран мултиграф.



ориентиран мултиграф



неговите списъци на съседство

Списъците на ориентирания мултиграф от Фигура 4.10 съвпадат със списъците на неориентирания мултиграф от Фигура 4.7. Причината е ясна: ориентирания мултиграф от Фигура 4.10 се явява съответният ориентиран граф на неориентирания мултиграф от Фигура 4.7 (Определение 85). Излиза, че, ако елементите на списъците не съдържат имената на ребрата—каквито са списъците на Фигура 4.7 и Фигура 4.10—не можем да кажем дали става дума за неориентиран граф или за ориентиран граф, който се явява съответен.

В някакъв смисъл, списъците на неориентиран граф, които нямат имена на ребра, всъщност представят (съответния) ориентиран граф.

4.1.4 Представяния на ограничени класове графи

Коренови дървета и масиви от родителите. Нека е дадено кореново дърво $\mathsf{T} = (\mathsf{V}, \mathsf{E})$ с корен $\mathtt{r}$. Може да представим T като като антиарборесценция (Определение 97) чрези списъци на съседство. Знаем, че всеки връх, различен от корена, има точно един родител в кореново дърво, а коренът няма родител. Следователно, за всеки връх $\mathtt{i} \in \mathsf{V}$:

- ако $\mathtt{i} \neq \mathtt{r}$, то списъкът на $\mathtt{i}$ има един единствен елемент, а именно този, който отговаря на родителя на $\mathtt{i}$;
- ако $\mathtt{i} = \mathtt{r}$, то списъкът на $\mathtt{i}$ е празен.

Щом списъците са с дължини едно или нула, още по-ефикасно от практическа гледна точка е те да не са свързани списъци, а клетки от масив. Така стигаме до идеята за *масив от родителите*. Това е масив $\mathtt{P}[1, \ldots, \mathtt{n}]$, където за всяко $\mathtt{i} \in \{1, \ldots, \mathtt{n}\}$:

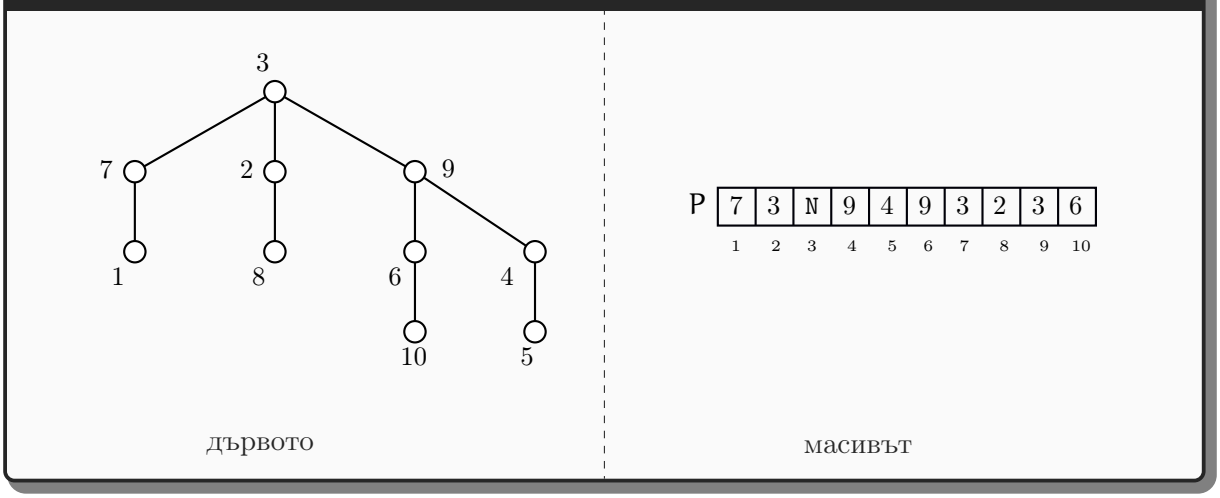
- ако $\mathtt{i} \neq \mathtt{r}$, то $\mathtt{P}[\mathtt{i}] = \mathtt{j}$, където $\mathtt{j}$ е родителят на $\mathtt{i}$;
- ако $\mathtt{i} = \mathtt{r}$, то $\mathtt{P}[\mathtt{i}] = \mathtt{Nil}$.

`Nil` е специална стойност, която индикира, че няма родител.

Със същия успех P[r] можеше да съдържа r, и по този начин коренът би бил единственият връх, чиято клекта съдържа своя индекс. Двете възможности са еднакво добри на практика, просто човек трябва да си избере предварително коя от тях да използва.

Фигура 4.11 показва кореново дърво и масивът на родителите, който го представя.

Фигура 4.11 : Кореново дърво и неговият масив от родители.



На Фигура 4.11 дървото нарочно е показано не като ориентирано, а като обикновено кореново дърво (вижте разсъжденията за това, дали кореновите дървета са непременно ориентирани или не, на стр. 113). Масивът го представя като антиарборесценция (Определение 97), но това е друго нещо! Какво представя масивът и какво **е** дървото са различни неща. За да се убедим в това: дървото на обхождането на BFS или DFS (Секция 4.2) е арборесценция, а го представяме с масив на родителите.

Интервални графи и множества от интервали. Интервалните графи (Подсекция 2.17.1) може да се представят с матрици на съседство, матрици на инцидентност или списъци на съседство. Това е съвсем естествено, понеже те са графи. За някои задачи обаче, като например задачата за разполагането на изпитите в залите на стр. 215, това не е ефикасно. Тази задача има алгоритмично решение, което е с почти линейно бързодействие, ако графът не се представя със списъци или матрици, а с интервалите. В този смисъл, множеството от интервалите **е** графът.

За да докажем някакви свойства на алгоритъма, ние можем да се позоваваме на съответния интервален граф, но за самия алгоритъм интервалите **представляват пълноценно** графа. Всеки опит да построим "истинско" представяне на графа с матрица или списъци и да използваме това представяне в задачата с изпитите води, в най-лошия случай, до алгоритъм с квадратична сложност по време. За някои други задачи може да се налага да построим списъците – примерно, ако искаме да обходим (по някаква причина...) интервалния граф в ширина (Секция 4.2). Но за задачата за изпитите това не се налага. За да я решим с алгоритъм е достатъчно да използваме интервалите – те са напълно адекватно представяне за целта.

4.2 Обхождания на графи

Най-базовата задача върху графи е да бъде обходен дадения граф. На английски "да обходя граф" е *to traverse a graph*, откъдето името на задачата на английски е *graph traversal*. В миналото е бил ползван терминът *graph search*, откъдето, без съмнение, идват имената "BFS" и "DFS", но днес "търсене в граф" (на английски, *graph search*) означава съвсем друго нещо: преследване и залавяне на беглец, движещ се в графа[†], така че ще ползваме "обхождане", а не "търсене/претърсване".

Когато разглеждаме някоя задача върху графи—която и да е тя—трябва да е кристално ясно какъв е видът графи, върху който е задачата. Задачата за обхождане на графи е дефинирана върху **ориентирани мултиграфи с възможни примки**. Ако я разглеждаме върху неориентирани граф или мултиграф G, то това е частен случай, в смисъл, че всъщност, на ниско ниво, решаваме задачата върху ориентирания граф или мултиграф, съответен на G (Определение 85).

Задачата е, да се обходят ребрата и върховете на даден граф, без да се пропуска нищо и без да се "зацикля". Не е коректно да се каже "без да се повтаря" – читателят ще забележи, че при тези обхождания попадаме много пъти в един и същи връх, а, ако графът е неориентиран, ребрата се обхождат по два пъти всяко. Ако няма изолирани върхове, достатъчно е да се обходят ребрата – това гарантира обхождане и на върховете. На доста места в Нета се казва, че се обхождат върховете. Това не е коректно и некоректността се вижда много лесно, ако мислим за обхождане на мултиграфи: ако има сноп паралелни ребра, всяко ребро от снопа трябва да се обходи отделно от другите, което е различно нещо от това да бъдат обходени само върховете-краища на ребрата от снопа.

Името на задачата идва от нагледното (а не формалното) разбиране за граф като колекция от точки и стрелки или криви: представяме си същество, което живее в графа, и което го обхожда. Или, по-реалистично, представяме си лабиринт, от който трябва да намерим изход или да намерим оставено някъде съкровище или *да убием Минотавъра*, при условие, че Минотавърът не се движи. За да сме сигурни, че не сме изпуснали нищо, трябва да минем през целия лабиринт; от друга страна, трябва да не зацикляме. С тази аналогия обаче трябва да се внимава: в графа имаме възможност да "скачаме" директно от връх в друг връх (при определени условия), на което в лабиринта съответствието би било да можем да се телепортираме от място в място (при определени условия), а не само да ходим из него.

Лабиринтът, който имаме предвид, не е точно пещерна система или плетеницата от коридори под двореца на цар Минос. Този лабиринт се състои от добре обособени номерирани стаи с номерирани врати, водещи към коридори, през които се достига до други стаи (или същата стая, ако са допустими примки). Ако искаме да обхождаме ориентиран граф, коридорите на съответния лабиринт трябва да са еднопосочни; ако искаме да обхождаме неориентиран граф, коридорите на съответния лабиринт трябва да са двупосочни.

По отношение на формалното, теоретико-множествено разбиране за графа, никакво обхождане не се извършва в истинския смисъл на думата. Алгоритмите, които решават задачата, обработват ребрата изчерпателно, без да пропускат и без да зациклят, и това е всичко. В базовата версия на алгоритмите BFS и DFS, тази обработка е почти никаква. В някакви по-изтънчени алгоритми, построени върху BFS или DFS, може да има истинска обработка на ребрата, особено ако последните съдържат повече информация.

При истинско, физическо обхождане на лабиринт има два начина да сбъркаме:

[†]Задачата за залавянето на беглеца има много разновидности: графът може да е ориентиран или не, преследвачите може да виждат беглеца или не, те може да имат способност "скачат" произволно от връх на връх, или да са ограничени да се "плъзгат" по ребрата, и така нататък.

1. да пропуснем;
2. да зациклим.

За да избегнем и двете можем да отбелязваме къде вече сме били, и да напредваме по някакъв систематичен начин в частта, в която още не сме били. Отбелязването на това, къде вече сме били, във физическия лабиринт най-вероятно би станало с някакво маркиране на коридорите. *Нишката на Ариадна* е именно средство за маркиране, само че само на някои коридори.

В алгоритмичната реализация на обхождането обаче не маркираме коридорите, а само стаите. Иначе казано, не маркираме ребрата, а само върховете. Причината е проста: **за да пестим памет**. За да обходим графа успешно, достатъчно е да ползваме един бит памет за всеки връх, който бит има смисъл на

- "не сме били в този връх" при стойност 0;
- "вече сме били в този връх" при стойност 1.

И така, за цялото маркиране ни е достатъчен един битов масив с n клетки. Ако искаме да маркираме ребрата, дори само с един бит на ребро, ще ни трябва масив с m клетки. А, както знаем, m може да е много по-голямо от n: ако става дума за обикновен граф, m може да е квадратично по-голямо от n, докато при мултиграф m може да е произволно голямо спрямо n.

На практика е полезно да различаваме не две, а три състояния на всеки връх. Две състояния са напълно достатъчни, за да решим задачата за обхождането, но ако искаме да решаваме по-сложни задачи от само обхождане, може да се налага да различаваме три състояния. Какви са точно тези три състояния и какъв смисъл имат, ще видим нататък.

Читателят ще забележи, че алгоритмичните реализации на обхождането, освен масива за обхождането, ползват още памет, в която отбелязват за всеки връх "докъде сме стигнали", тоест, кои излизащи ребра сме ползвали вече и кои, още не. Това е особено важно за DFS. В аналогията с лабиринта, вратите, водещи навън от дадена стая, са номерирани и ние ги отваряме в този ред; това означава да имаме по един брояч[†] за всяка стая. Допускаме, че броячът иска само константна по големина памет, независимо от това какви числа ще съхранява. Това е стандартно допускане в теорията на алгоритмите. При това допускане е ясно, че паметта за всички броячи е пропорционална на n и не зависи от m. Като цяло, паметта за състоянията на върховете плюс паметта за броячите/указателите е пропорционална на n, тоест, линейна функция на n, и **не зависи от** m.

Обхождането винаги започва от един връх, който ще наричаме *стартов връх*. Този връх може да е част от входа на алгоритъма, но може да бъде избран от алгоритъма по някакъв начин: примерно, произволен връх или връх 1 (което е същото, защото идентификаторите на върховете са раздадени произволно). Да кажем, че стартовият връх е 1.

Общата схема на обхожданията, които ще разгледаме, е следната. Това не е конкретен алгоритъм, а нещо по-абстрактно: именно схема, по която за изградени няколко алгоритъма. За изход от схемата не говорим; изход имат конкретните алгоритми BFS и DFS, които ще видим нататък.

[†]Ако графът е представен със списъци на съседство, това не е брояч, а е указател в списъка на дадения връх. Примери ще видим, когато разгледаме реализациите на BFS и DFS.

Алгоритмична схема 1: Схема за обхождане на графи

Вход: ориентиран мултиграф G.
Променливи: множество от върхове S, връх $\mathfrak{x}$, булев масив $\mathrm{visited}[1, \ldots, \mathfrak{n}]$.
Инициализирай visited с False.

$\mathsf{S} \leftarrow \{1\}$, $\mathrm{visited}[1] \leftarrow \mathrm{True}$.

❶ Ако $\mathsf{S} = \varnothing$, прекрати алгоритъма.

❷ В противен случай, извади елемент от S и го сложи в $\mathfrak{x}$.

❸ За всяко ребро $(\mathfrak{x}, \mathfrak{y})$, ако $\mathrm{visited}[\mathfrak{y}] = \mathrm{False}$:
① то слагаме $\mathfrak{y}$ в S и правим $\mathrm{visited}[\mathfrak{y}] \leftarrow \mathrm{True}$.
② иначе, прескачаме $\mathfrak{y}$.

❹ Отиваме на ❶.

Ще покажем, че тази алгоритмична схема е коректна в смисъл, че алгоритъм, изграден по нея, не зацикля и обхожда всичко. Но първо да видим както означава "всичко".

Разсъждаваме над това, каква част от графа ще обходи алгоритъм, изграден по Схема 1. Дали непременно ще обходи целия граф? В метафората с лабиринта: дали ще обходим целия лабиринт, ако тръгнем от стая 1 и само минаваме по коридори от стая в стая, съблюдавайки посоките на коридорите, и евентуално се телепортираме обратно в стая, в която вече сме били? Отговорът е: ***не непременно****. Да се върнем на разглеждането на графи. Ако графът G е неориентиран (това е частен случай), ще го обходим целия, започвайки от връх 1, тстк G е свързан. В общия случай обаче графът не е свързан и ще обходим само свързаната компонента, в която е връх 1. Ако разглеждаме ориентирани графи, въпросът какво ще обходим, стартирайки във връх 1, е по-сложен и "според зависи". Ако графът е силно свързан, ще го обходим целия задължително. Ако не е силно свързан, със сигурност ще обходим силно свързаната компонента, в която се намира връх 1, но* ***може*** *и да излезем извън нея и да обходим и други силно свързани компоненти; а може да не излезем от нея изобщо. Като пример да разгледаме Фигура 3.9:*

- *ако обхождането започне във връх $\mathfrak{u}$[†], ще обходим целия граф, ще "излезем" от $\mathfrak{u}$ (и от червената силно свързана компонента) през някое от ребрата e_1, e_2 или e_3 и ще обходим и синята силно свързана компонента; тоест, ще обходим целия граф.*
- *ако започне в някой от другите върхове, няма да може да излезе от синята силно свързаната компонента, защото няма да може да прекоси никое от ребрата e_1, e_2 или e_3 заради посоката, така че обходена ще се окаже само синята силно свързана компонента.*

Със сигурност не може да се прехвърляме от една слабо свързана компонента в друга поради липсата на ребра от кой да е връх на едната до кой да е връх на другата.

[†]С други думи, ако връх 1 е връх $\mathfrak{u}$.

Накратко: ако говорим за неориентиран граф, ще обходим точно свързаната компонента, в която е връх 1*; ако говорим за ориентиран граф, ще обходим точно тази част на графа, която е достижима от връх* 1*.*

Да се върнем на изследването на коректността на алгоритмите, изградени по Схема 1. Ето доказателство.

Теорема 50: Коректност на алгоритмите по Схема 1

Всеки алгоритъм за обхождане, изграден стриктно по Схема 1, е коректен.

Доказателство: Нека AlgX е произволен алгоритъм, изграден стриктно по Схема 1. Ще покажем, че AlgX не зацикля и обхожда всичко, което един коректен алгоритъм за обхождане трябва да обходи.

Това, че зацикляне е невъзможно, е очевидно: в S "влизат" само непосетени върхове и, веднъж влезли, те стават посетени; освен това е невъзможно посетен връх да стане отново непосетен. Тъй като на всяко достигане на ред ❷ един връх бива изваден от S, а S е крайно множество във всеки момент, то рано или късно алгоритъмът ще прекрати работата си на ред ❶.

Ще покажем, че алгоритъмът обхожда всичко, което е достижимо от връх 1. Както видяхме, това "всичко" зависи доста от конкретиката в ориентирания случай, така че за простота на аргумента ще разгледаме частния случай, в който G е неориентиран. Нещо повече, БОО допускаме, че G е свързан. Тогава това, което трябва да покажем е, че алгоритъм, изграден по Схема 1, ще обходи целия граф.

Да допуснем противното: в края на алгоритъма съществува връх $\mathtt{i}$, който е непосетен в смисъл, че visited[$\mathtt{i}$] = False. Но очевидно visited[1] = True в края на алгоритъма, така че $\mathtt{i} \neq 1$. Тъй като G е свързан, то съществува път $\mathtt{p}$ между върхове 1 и $\mathtt{i}$. Забележете, че за всеки връх $\mathtt{j} \in V(\mathtt{p})$ е вярно, че visited[$\mathtt{j}$] $\in$ {False, True}: това е вярно за всеки момент от работата на алгоритъма, а не само за края. Съгласно Наблюдение 9, в края на алгоритъма в $\mathtt{p}$ има ребро $(\mathtt{k}, \ell)$, такова че visited[$\mathtt{k}$] = True, а visited[ℓ] = False.

В самото начало на алгоритъма всеки връх има visited стойност False. След това, връх получава visited стойност True тстк влиза в S – това е елементарно да се докаже по индукция по изпълнението на цикъла на редове ❶–❹. При махане на връх от S, алгоритъмът вкарва в S всички негови съседи, които не са били в S досега, и присвоява True на техните visited стойности (редове ❸ и ①).

Но в края на алгоритъма множеството S е празно, така че връх k е бил сложен в S и после е бил изваден от S, като при изваждането му от S всички негови съседи, които са имали visited стойност False, са получили visited стойност True. Следователно, след изваждането на k от S и преди да се прави нещо друго, всички съседи на k се оказват с visited стойност True. Но ℓ е съсед на k в графа, щом има ребро $(\mathtt{k}, \ell)$ в пътя. Следователно, visited[ℓ] е True при изваждането на k от S, и остава True до края на алгоритъма, в противоречие с допускането, че visited[ℓ] = False в края на алгоритъма. □

4.2.1 BFS

BFS е алгоритъмът за обхождане на графи в ширина. Той е изграден съгласно Схема 1, като множеството S е реализирано чрез абстрактен тип данни опашка (FIFO). Името идва от **B**readth-**F**irst **S**earch[†].

[†]А не Breadth-First Traversal, което би било коректно, ако говорим за graph traversal. Терминът "search" се ползва по исторически причини.

Много общо казано, той започва от дадения стартов връх, обхожда неговите съседи, като при това обхожда ребрата, с които ги достига, после обхожда съседите на съседите (различни от стартовия връх), и така нататък, обхождайки върховете по нарастване на разстоянията от стартовия връх. Версията на BFS, която ще разгледаме, ползва не две, а три състояния за всеки връх $\mathtt{i}$:

1. $\mathtt{i}$ е непосетен. Всички върхове в началото са непосетени. Условно казваме, че непосетените върхове са *бели*.

2. $\mathtt{i}$ е посетен, но още не сме приключили с него. В метафората с лабиринта, такова е състоянието на стая, в която вече сме били, но все още не сме използвали всички врати, водещи навън от нея; с други думи, не сме обходили коридорите, излизащи от нея. Условно казваме, че върховете в това състояние са *сиви*.

3. $\mathtt{i}$ е посетен и вече сме приключили с него. В метафората с лабиринта, това е стая, в която сме били и освен това сме обходили всички коридори, излизащи от нея. Условно казваме, че върховете в това състояние са *черни*.

И така, BFS, който ще разгледаме, ползва масив $\mathrm{colour}[1, \ldots, \mathtt{n}]$, всеки елемент от който има точно една от стойностите `white`, `grey` и `black`.

BFS изгражда и така нареченото *дърво на обхождането*. Ако графът е неориентиран, това е покриващо дърво за свързаната компонента, съдържаща стартовия връх. То е кореново дърво с корен стартовия връх, а ребро $\mathtt{e}$ от графа влиза в дървото тстк BFS открива бял връх чрез $\mathtt{e}$. С други думи, дървото на обхождането показва как BFS е откривал непосетени върхове. Забележете, че това дърво касае само обхождането на върховете! Всяко ребро (от свързаната компонента, съдържаща стартовия връх) бива обходено от BFS, но не всяко ребро влиза в дървото на обхождането.

Тук се вижда защо за кореновите дървета казахме на стр. 113, че имат неявна ориентация – хем са ориентирани графи, хем не са. Дървото на обхождането на неориентиран граф $\mathtt{G}$ е подграф на $\mathtt{G}$ и като такъв трябва да е от същия вид, тоест, неориентиран граф. Ако все пак се опитаме да дадем ориентация на ребрата на покриващото дърво, естествено е тя да е навън от корена, защото това е посоката, в която откриваме нови върхове; с други думи, дървото на обхождането трябва да е арборесценция. Както ще видим обаче, BFS го изгражда като ориентиран граф, но с точно обратната ориентация, а именно от листата към корена; с други думи, BFS строи антиарборесценция.

За ориентираните графи не сме въвели понятие "покриващо дърво". В теорията на графите има понятие за *покриващ граф на ориентиран граф* (на английски, *spanning subgraph of a directed graph*), но със смисъл, който е несъвместим с арборесценция или антиарборесценция. Доколкото е известно на автора на записките, това понятие се ползва само в контекста на силно свързани ориентирани графи, което е аналогично на това, че "покриващо дърво" се ползва само в контекста на свързани неориентирани графи.

- "Покриващ граф" за свързан **неориентиран** граф $\mathtt{G}$ е свързан **неориентиран** подграф $\mathtt{G}'$ с върхове $\mathtt{V}(\mathtt{G})$. Използвайки $\mathtt{G}'$, можем да отидем, образно казано, от който връх искаме в който друг връх искаме. $\mathtt{G}'$ да е дърво (и по-точно, покриващо дърво) означава покриващият подграф да е **минимален като брой ребра** и все пак да свързва всички върхове от $\mathtt{V}(\mathtt{G})$, тъй като по него (покриващото дърво) можем да отидем от който връх искаме в който връх искаме.

- Ако се опитаме да направим нещо аналогично за ориентирани графи, то графът $\mathtt{G}$, в който "се развива действието" трябва да е **силно свързан**, за да може да отидем откъдето искаме, където другаде искаме, в ориентирания смисъл. Всеки покриващ подграф

G', който разглеждаме, трябва също да е ориентиран, върховете му да са $V(G)$ и да е **силно свързан**. Но лесно се вижда, нито арборесценциите, нито антиарборесценциите са силно свързани графи, така че G' да е **минимален като брой ребра** изобщо не е същото като G' да е арборесценция или антиарборесценция.

Нещо повече. Намирането на минимален покриващ ориентиран силно свързан подграф на ориентиран силно свързан граф е **NP**-трудна задача, поне в тегловния вариант, което е друга важна разлика неориентираните графи, при които намирането на минимално покриващо дърво е алгоритмично лесно, поне в тегловния вариант (Секция 4.3).

Въпреки всички тези съображения, BFS (а също и DFS) върху ориентирани графи изгражда арборесценция, която показва как алгоритъмът е откривал непосетени досега върхове, но тази арборесценция има само тази роля: да показва как е напредвал алгоритъмът. Тук не твърдим, че изградената арборесценция може да се ползва като покриващ подграф, за да можем по нея да отидем откъдето искаме, където другаде искаме. Подобно на BFS върху неориентирани графи, алгоритъмът всъщност изгражда антиарборесценция.

Накратко: независимо от това, дали графът е неориентиран или ориентиран, BFS изгражда антиарборесценция чрез масив на родителите $\pi[1, \ldots, n]$, която е репрезентация или на кореново покриващо дърво с корен стартовия връх, ако графът е неориентиран, или на арборесценция с корен стартовия връх, ако графът е ориентиран.

Освен масивите $\text{colour}[1, \ldots, n]$ и $\pi[1, \ldots, n]$, BFS попълва и масив $d[1, \ldots, n]$. В Теорема 51 доказваме, че в края на алгоритъма, $d[1, \ldots, n]$ масив от разстоянията по отношение на избрания начален връх 1. Ако G е неориентиран, това са разстоянията между връх 1 и останалите върхове, съгласно Определение 22. Ако G е ориентиран, това са разстоянията от връх 1 до останалите върхове, съгласно Определение 95.

Масивите $\text{colour}[1, \ldots, n]$, $\pi[1, \ldots, n]$ и $d[1, \ldots, n]$ са изходът на BFS. Те предоставят полезна информация за графа и конкретното извършено обхождане. Масивът $\text{colour}[1, \ldots, n]$ съдържа информация за достижимостта от връх 1. В края на BFS, точно тези върхове i, за които $\text{colour}[i] = \text{black}$, са върховете, които са достижими от връх 1, а за останалите върхове i е вярно, че $\text{colour}[i] = \text{white}$. Следователно, ако G е неориентиран, "черните върхове" при приключване на алгоритъма са върховете на свързаната компонента, на която принадлежи стартовия връх 1. Ако обаче е известно предварително, че G е неориентиран и свързан, или ориентиран и силно свързан, би било излишно да връщаме $\text{colour}[1, \ldots, n]$, защото знаем какво ще съдържа: само стойности black.

Допускаме, че графът-вход G е представен чрез списъци на съседство. Не е невъзможно да е представен и чрез матрица на съседство, но в примерите за работата на BFS, които ще видим нататък, ще смятаме, че е представен със списъци. Матрицата на инцидентност не е подходящо средство за представяне на графа, защото в нея не виждаме бързо съседите на даден връх.

Както ще видим в примерите, конкретиката на списъците на съседство е важна за това, как именно ще работи BFS върху даден граф. Точната форма на дървото на обхождането зависи от списъците, тоест, от представянето на ниско ниво. Само от рисунката на графа не можем да кажем какво е дървото.

Ето описание на BFS чрез така наречения *псевдокод*. Псевдокод е описание на алгоритъм, което хем е достатъчно ясно и недвусмислено, така щото да можем да програмираме с лекота този алгоритъм на някакъв подходящ конкретен език за програмиране, хем не съдържа конкретни особености на някакъв език за програмиране, което ни позволява да се съсредоточим само върху алгоритъма. В този смисъл, описанието на псевдокод е междинно ниво между най-общото описание на естествен език и съвсем конкретната програмна реализация на език

като C, Java или Python. Псевдокодът, който авторът на записките предпочита, е близък до този от учебника по алгоритми на Cormen, Leiserson, Rivest и Stein [16]. Той (стилът на псевдокода) се отличава с икономичност, яснота и естественост (примерно, броим от 1 като на Pascal, а не от 0 като на C).

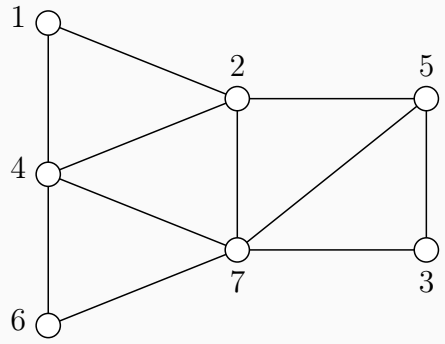
Псевдокодът на BFS ползва променлива Q от тип опашка (FIFO). Q реализира множеството S от Схема 1. Q е абстрактен тип данни, който се характеризира с три примитива Enqueue, Dequeue и IsEmpty, чиито имена трябва да са достатъчно индикативни за това какво правят. Имплементацията остава скрита, както човек би очаквал от абстрактен тип данни; тя не ни интересува в случая. "adj[x]" на ред 9 означава списъка на съседите на x.

BFS($G = (V, E)$): ориентиран мултиграф с възможни примки, като $V = \{1, \ldots, n\}$)

```
 1  for i ← 1 to n
 2      colour[i] ← white, d[i] ← ∞, π[i] ← Nil
 3  colour[1] ← grey
 4  d[1] ← 0
 5  създай празна опашка Q
 6  Enqueue(Q, 1)
 7  while not IsEmpty(Q) do
 8      x ← Dequeue(Q)
 9      for y ∈ adj[x]
10          if colour[y] = white
11              colour[y] ← grey
12              d[y] ← d[x] + 1
13              π[y] ← x
14              Enqueue(Q, y)
15      colour[x] ← black
16  return colour, π, d
```

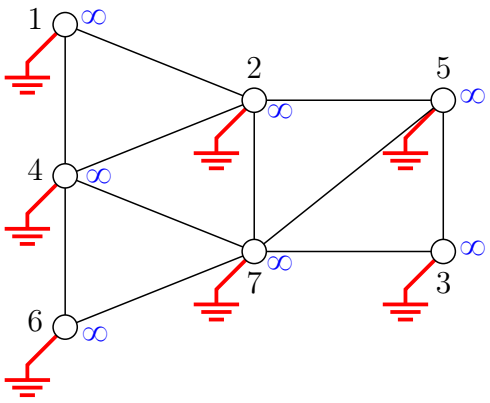
Да разгледаме подробно работата на BFS върху графа от Фигура 4.12. За простота, графът е неориентиран и свързан. Вдясно са показани списъците на съседство. Подредбата на върховете във всеки от тях е напълно произволна, но списъците са важни, за да видим **точно** как работи BFS; без тях (тоест, само от рисунката на графа) не може да определим точно дървото на обхождането.

Фигура 4.12 : Графът, върху който илюстрираме BFS.



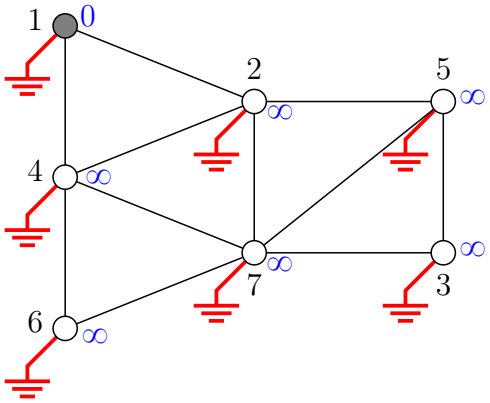
1 : 4, 2
2 : 5, 1, 7, 4
3 : 5, 7
4 : 6, 1, 2, 7
5 : 2, 7, 3
6 : 7, 4
7 : 5, 2, 4, 3, 6

Първо BFS инициализира масивите colour, π и $\mathtt{d}$ на редове 1–2. Цветовете на върховете са бели, също както в предишната фигура, предшествениците са Nil, което е показано чрез знака за заземяване от електротехниката (в червено) до всеки връх, а $\mathtt{d}$ стойностите са ∞ (в синьо).



1 : 4, 2
2 : 5, 1, 7, 4
3 : 5, 7
4 : 6, 1, 2, 7
5 : 2, 7, 3
6 : 7, 4
7 : 5, 2, 4, 3, 6

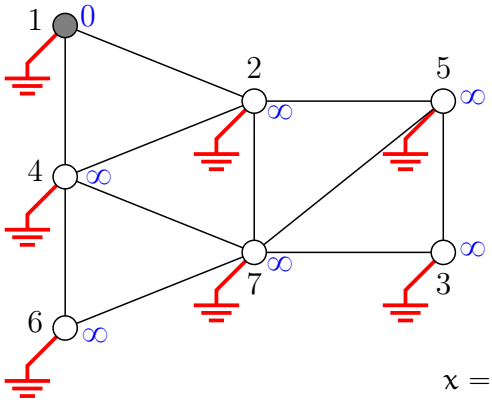
След това BFS променя colour $\mathtt{d}$ стойностите на връх 1 на ред 3, създава опашка Q на ред 5 и слага 1 в нея на ред 6.



1 : 4, 2
2 : 5, 1, 7, 4
3 : 5, 7
4 : 6, 1, 2, 7
5 : 2, 7, 3
6 : 7, 4
7 : 5, 2, 4, 3, 6

Q: $\boxed{1}$ ⟵

На ред 7 булевото условие е истина (опашката не е празна) и тялото на **while**-цикъла (редове 7–15) се изпълнява. На ред 8 връх 1 "излиза" от опашката (при което тя става празна, но до следващото достигане на ред 7 в нея ще влязат върхове).

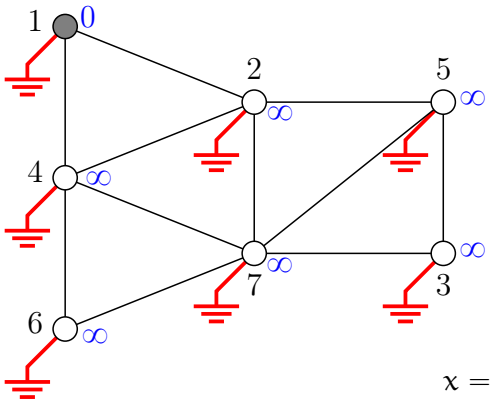


$x = 1$

1 : 4, 2
2 : 5, 1, 7, 4
3 : 5, 7
4 : 6, 1, 2, 7
5 : 2, 7, 3
6 : 7, 4
7 : 5, 2, 4, 3, 6

Q: $\square$ ⟵

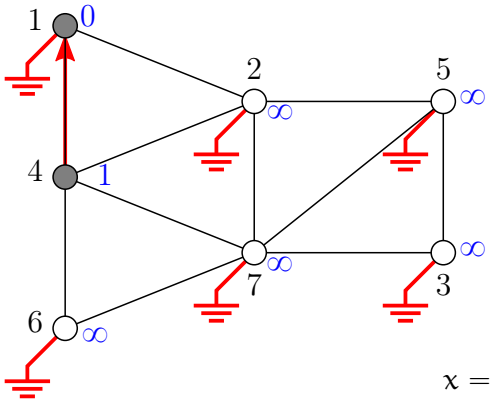
Променливата x съдържа 1. Променливата y получава последователно стойностите на съседите на x, а именно 4 и 2, в **този** ред. Първо y става 4.



1 : 4, 2
2 : 5, 1, 7, 4
3 : 5, 7
4 : 6, 1, 2, 7
5 : 2, 7, 3
6 : 7, 4
7 : 5, 2, 4, 3, 6

Q: ▯

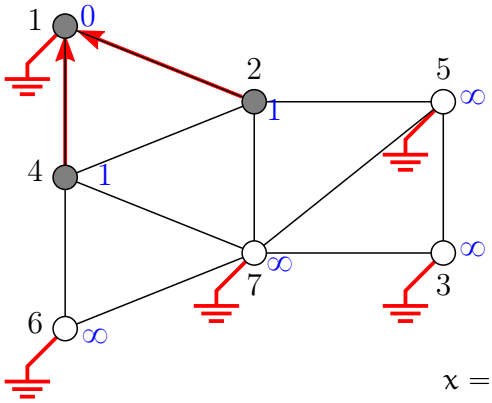
Булевото условие на ред 10 е истина, понеже връх 4 е бял, така че редове 11–14 се изпълняват, в резултат на което цветът на 4 става сив, d стойността му става 1, предшественикът му става връх 1, и 4 "влиза" в Q.



1 : 4, 2
2 : 5, 1, 7, 4
3 : 5, 7
4 : 6, 1, 2, 7
5 : 2, 7, 3
6 : 7, 4
7 : 5, 2, 4, 3, 6

Q: 4

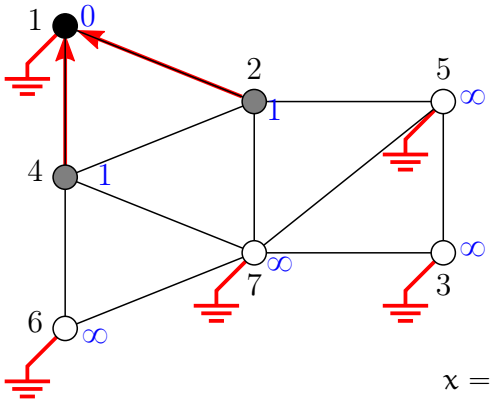
Изпълнението отново е на ред 10 с y = 2. Булевото условие отново е истина, понеже връх 2 е бял, така че редове 11–14 се изпълняват, в резултат на което цветът на 2 става сив, d стойността му става 1, предшественикът му става връх 1, и 2 "влиза" в Q.



1 : 4, 2
2 : 5, 1, 7, 4
3 : 5, 7
4 : 6, 1, 2, 7
5 : 2, 7, 3
6 : 7, 4
7 : 5, 2, 4, 3, 6

Q: 2 4

Изпълнението отива на ред 15, където връх 1 става черен.

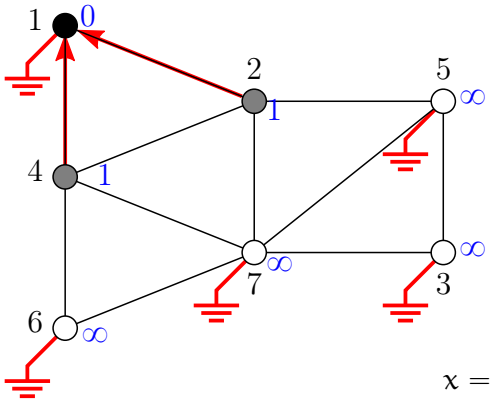


x = 1 y = 2

1 : 4, 2
2 : 5, 1, 7, 4
3 : 5, 7
4 : 6, 1, 2, 7
5 : 2, 7, 3
6 : 7, 4
7 : 5, 2, 4, 3, 6

Q: 2 4

Изпълнението отива на ред 7. Q не е празна, така че тялото на **while**-цикъла (редове 7–15) се изпълнява пак. x става 4, като 4 "излиза" от Q.

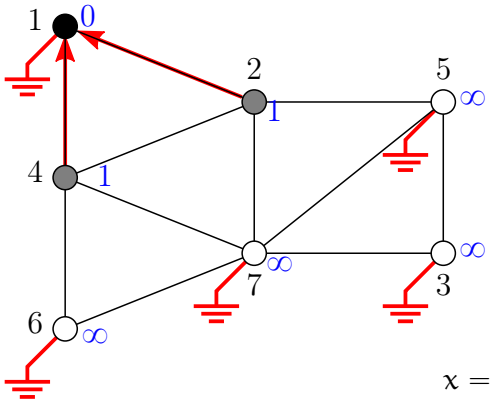


x = 4

1 : 4, 2
2 : 5, 1, 7, 4
3 : 5, 7
4 : 6, 1, 2, 7
5 : 2, 7, 3
6 : 7, 4
7 : 5, 2, 4, 3, 6

Q: 2

Променливата y получава последователно стойностите на съседите на x, а именно 6, 1, 2 и 7, в **този** ред. Първо y става 6.

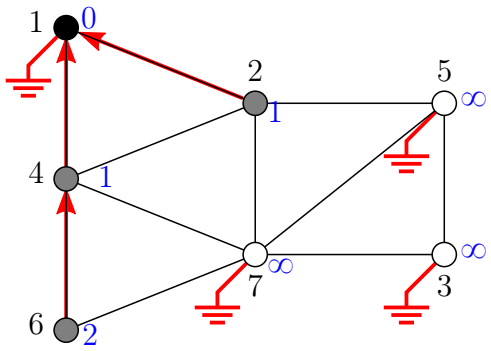


x = 4 y = 6

1 : 4, 2
2 : 5, 1, 7, 4
3 : 5, 7
4 : 6, 1, 2, 7
5 : 2, 7, 3
6 : 7, 4
7 : 5, 2, 4, 3, 6

Q: 2

Булевото условие на ред 10 е истина, понеже връх 6 е бял, така че редове 11–14 се изпълняват, в резултат на което цветът на 6 става сив, d стойността му става 2, предшественикът му става връх 4, и 6 "влиза" в Q.

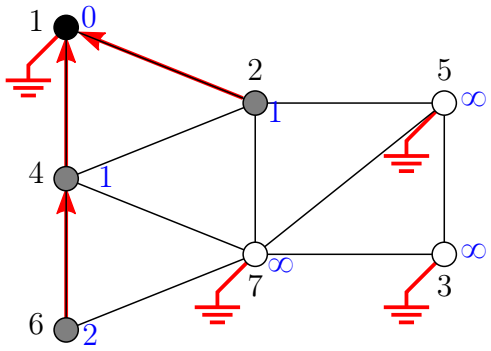


$x = 4$ $y = 6$

1 : 4, 2
2 : 5, 1, 7, 4
3 : 5, 7
4 : 6, 1, 2, 7
5 : 2, 7, 3
6 : 7, 4
7 : 5, 2, 4, 3, 6

Q: 6 | 2 →

При следващото достигане на ред 10, булевото условие е лъжа, понеже връх 1 не е бял, така че при $y = 1$ тялото на **for**-цикъла на редове 11–14 не се изпълнява.

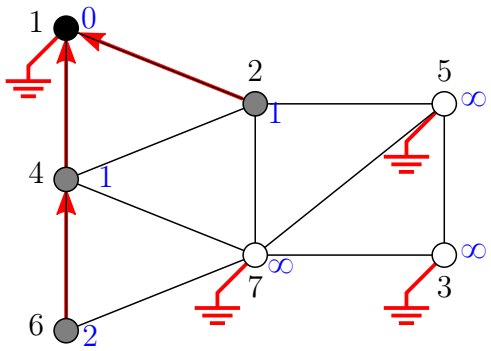


$x = 4$ $y = 1$

1 : 4, 2
2 : 5, 1, 7, 4
3 : 5, 7
4 : 6, 1, 2, 7
5 : 2, 7, 3
6 : 7, 4
7 : 5, 2, 4, 3, 6

Q: 6 | 2 →

При следващото достигане на ред 10, булевото условие е лъжа, понеже връх 2 не е бял, така че при $y = 2$ тялото на **for**-цикъла на редове 11–14 пак не се изпълнява.

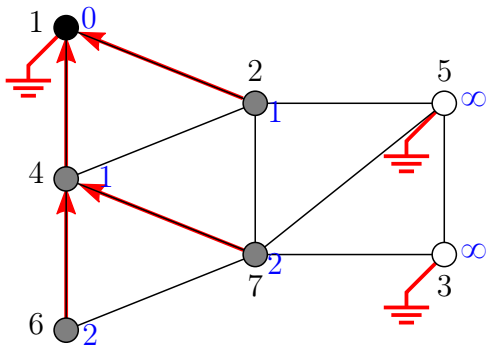


$x = 4$ $y = 2$

1 : 4, 2
2 : 5, 1, 7, 4
3 : 5, 7
4 : 6, 1, 2, 7
5 : 2, 7, 3
6 : 7, 4
7 : 5, 2, 4, 3, 6

Q: 6 | 2 →

При следващото достигане на ред 10, булевото условие е истина, понеже връх 7 е бял, така че при $y = 7$ тялото на **for**-цикъла на редове 11–14 се изпълнява.

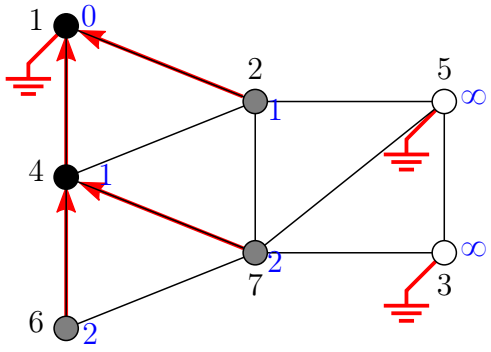


x = 4 y = 7

1 : 4, 2
2 : 5, 1, 7, 4
3 : 5, 7
4 : 6, 1, 2, 7
5 : 2, 7, 3
6 : 7, 4
7 : 5, 2, 4, 3, 6

Q: 7 6 2

След което връх 4 става черен и BFS “приключва” с него.

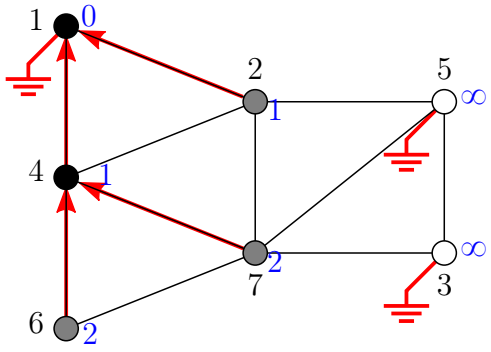


x = 4 y = 7

1 : 4, 2
2 : 5, 1, 7, 4
3 : 5, 7
4 : 6, 1, 2, 7
5 : 2, 7, 3
6 : 7, 4
7 : 5, 2, 4, 3, 6

Q: 7 6 2

След това BFS изважда връх 2 от опашката.

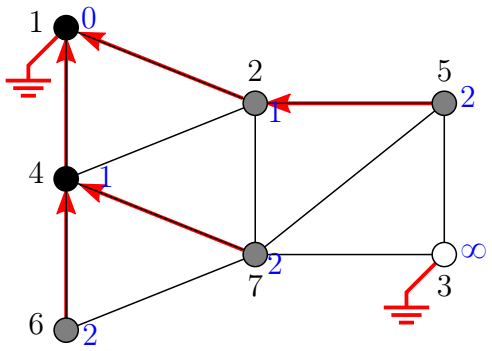


x = 2

1 : 4, 2
2 : 5, 1, 7, 4
3 : 5, 7
4 : 6, 1, 2, 7
5 : 2, 7, 3
6 : 7, 4
7 : 5, 2, 4, 3, 6

Q: 7 6

y взема стойностите 5, 1, 7 и 4, в този ред. Първо y става 5, който е бял връх и **for**-цикълът на редове 11–14 се изпълнява.

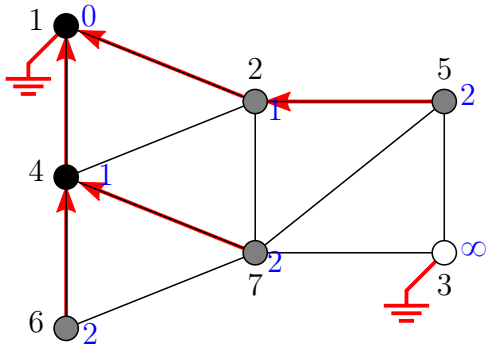


$x = 2$ $y = 5$

1 : 4, 2
2 : 5, 1, 7, 4
3 : 5, 7
4 : 6, 1, 2, 7
5 : 2, 7, 3
6 : 7, 4
7 : 5, 2, 4, 3, 6

Q: 5 7 6

Върховете 1, 7 и 4 не са бели, така че те се прескачат.

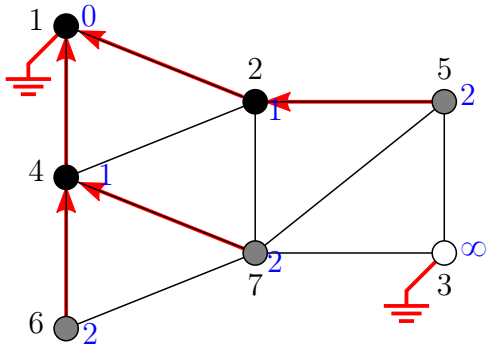


$x = 2$ $y = 4$

1 : 4, 2
2 : 5, 1, 7, 4
3 : 5, 7
4 : 6, 1, 2, 7
5 : 2, 7, 3
6 : 7, 4
7 : 5, 2, 4, 3, 6

Q: 5 7 6

После 2 става черен.

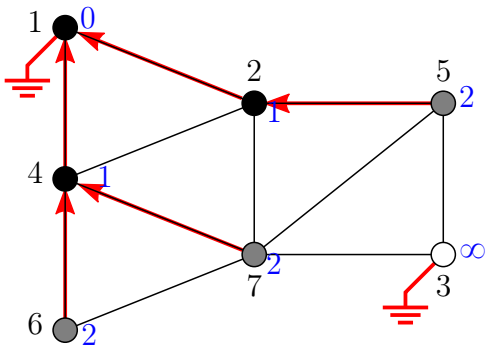


$x = 2$

1 : 4, 2
2 : 5, 1, 7, 4
3 : 5, 7
4 : 6, 1, 2, 7
5 : 2, 7, 3
6 : 7, 4
7 : 5, 2, 4, 3, 6

Q: 5 7 6

После **while**-цикълът се изпълнява с $x = 6$.

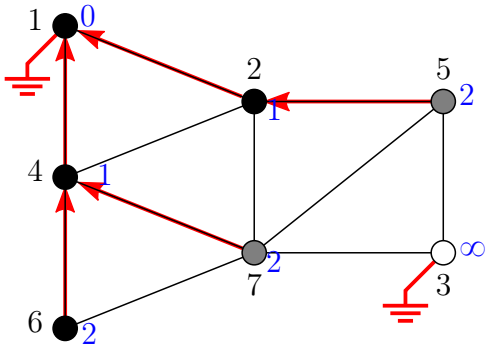


x = 6

1 : 4, 2
2 : 5, 1, 7, 4
3 : 5, 7
4 : 6, 1, 2, 7
5 : 2, 7, 3
6 : 7, 4
7 : 5, 2, 4, 3, 6

Q: 5 | 7 →

Тъй като и двата съседа на връх 6 не са бели, нищо не става, освен че връх 6 става черен.

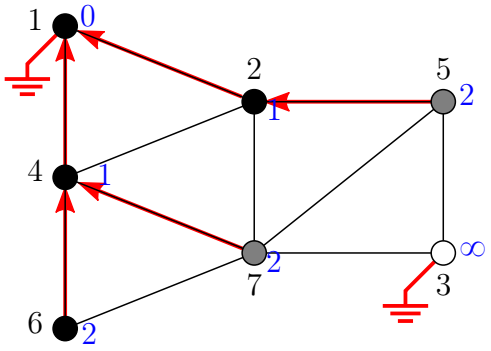


x = 6

1 : 4, 2
2 : 5, 1, 7, 4
3 : 5, 7
4 : 6, 1, 2, 7
5 : 2, 7, 3
6 : 7, 4
7 : 5, 2, 4, 3, 6

Q: 5 | 7 →

После **while**-цикълът се изпълнява с x = 7.

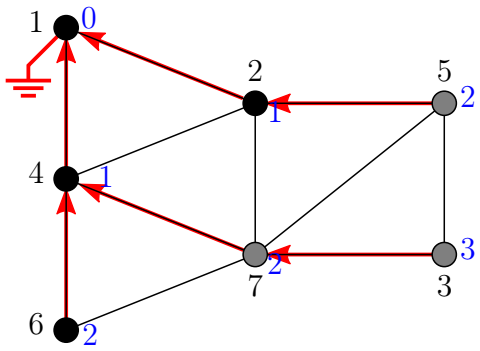


x = 7

1 : 4, 2
2 : 5, 1, 7, 4
3 : 5, 7
4 : 6, 1, 2, 7
5 : 2, 7, 3
6 : 7, 4
7 : 5, 2, 4, 3, 6

Q: 5 →

Съседите на 7 се “обработват” в реда 5, 2, 4, 3 и 6, като само 3 е бял и не се прескача.

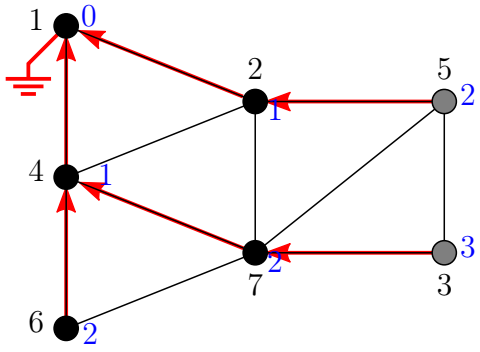


$x = 7$ $y = 3$

1 : 4, 2
2 : 5, 1, 7, 4
3 : 5, 7
4 : 6, 1, 2, 7
5 : 2, 7, 3
6 : 7, 4
7 : 5, 2, 4, 3, 6

Q: 3 5

После BFS приключва с връх 7.

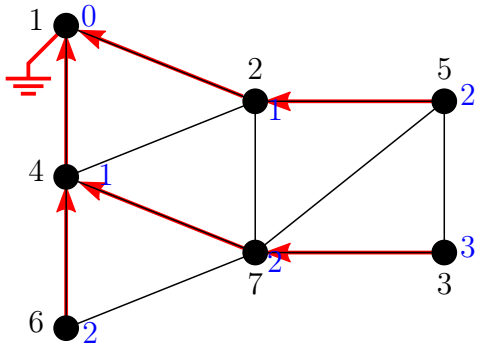


$x = 7$ $y = 6$

1 : 4, 2
2 : 5, 1, 7, 4
3 : 5, 7
4 : 6, 1, 2, 7
5 : 2, 7, 3
6 : 7, 4
7 : 5, 2, 4, 3, 6

Q: 3 5

После BFS изважда връх 5 от опашката, нищо не става със съседите му, после изважда връх 3, и с неговите съседи не прави нищо, и после опашката е празна и алгоритъмът приключва.

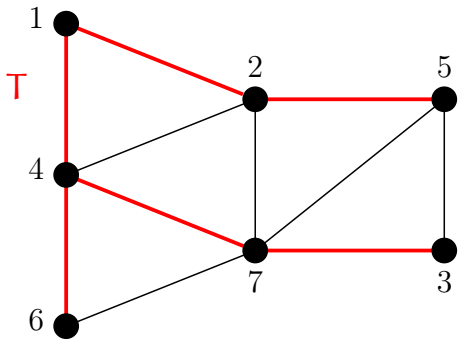


1 : 4, 2
2 : 5, 1, 7, 4
3 : 5, 7
4 : 6, 1, 2, 7
5 : 2, 7, 3
6 : 7, 4
7 : 5, 2, 4, 3, 6

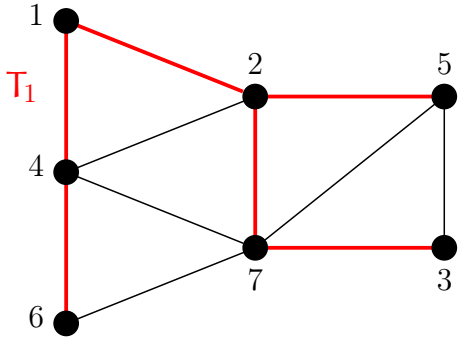
Q:

Дървото на обхождането T, което масивът π реализира, изглежда така, ако мислим за него като за подграф[†]:

[†]Щом е подграф на неориентиран граф, дървото трябва също да е неориентиран граф. Това, че го представяме като антиарборесценция, е друго нещо.



Дървото обаче можеше да бъде следното T_1, ако списъкът на връх 1 беше $\boxed{1:\ 2,\ 4}$:

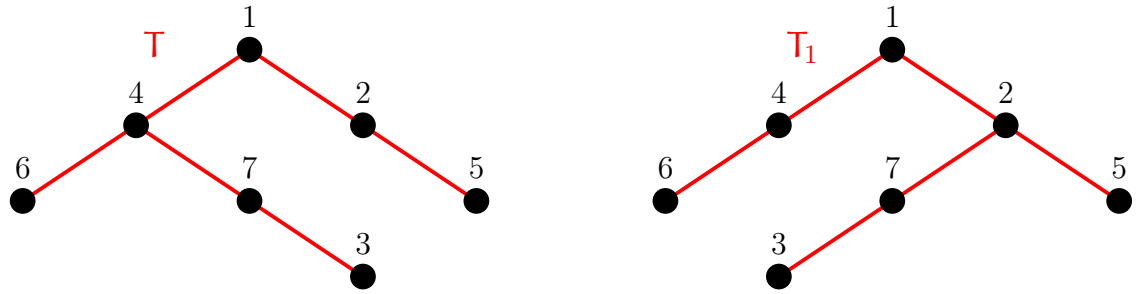


Причината е ясна: ако в списъка на 1 връх 2 се появява преди 4, то в опашката 2 щеше да влезе преди 4, така че и да излезе преди 4, следователно и двата бели съседа[†] 7 и 5 на 2 щяха да станат сиви и при това и двете ребра $(2,7)$ и $(2,5)$ да влязат в дървото. После, когато x стане 4, връх 7 вече би бил сив, така че $(4,7)$ не би влязло в дървото.

И така, точната форма на дървото зависи от конкретиката на списъците, защото е възможно BFS да "тръгне" първо насам или първо натам, което, естествено, се отразява в дървото.

Да разгледаме тези две дървета на обхождането, които са построени от BFS при различни списъци на съседство, едно до друго:

[†]Става дума за момента от работата на BFS, в който x = 2; в този момент и 7, и 5 са бели.



Игнорирайте това, че дърветата са изоморфни и като коренови дървета (вижте Определение 58), това се е получило случайно, понеже примерът е малък. Съществото е друго: дърветата са различни, но с една и съща височина. Това вече не е случайно, а следва от Теорема 51.

Да си представим опашката Q от BFS по друг начин. Както знаем от Теорема 50, BFS обхожда всички върхове (щом G е неориентиран и свързан), така че всеки връх от $\mathsf{V}(\mathsf{G})$ в някакъв момент влиза в Q и в някакъв следващ момент излиза от Q. Да си представим редицата от върховете на графа в реда, в който влизат в Q, само че написана **отдясно наляво**. За примера на BFS, който разгледахме преди малко, тази редица е

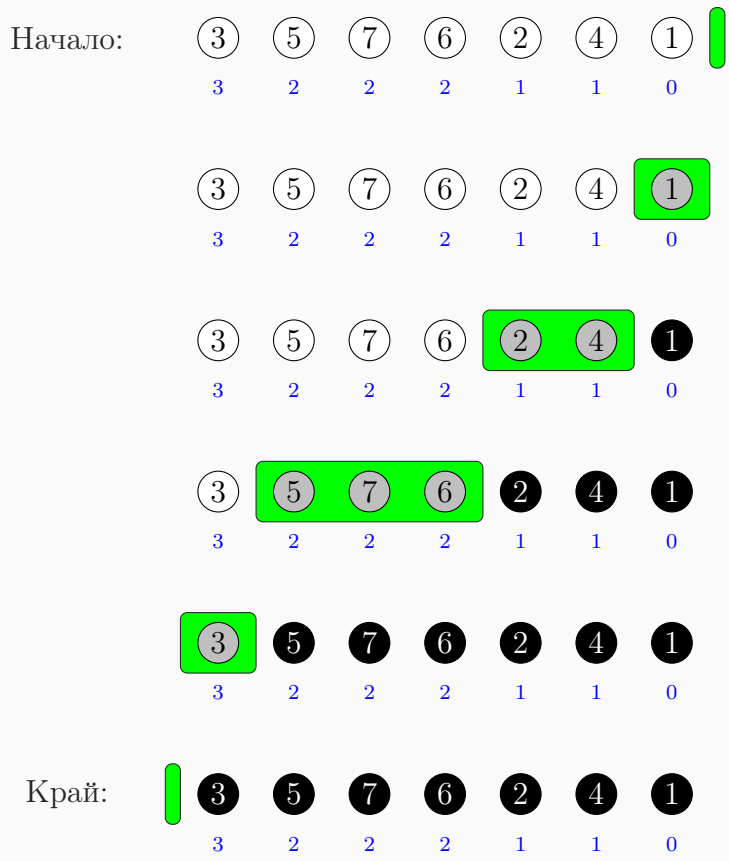
$$\underleftarrow{3,5,7,6,2,4,1} \tag{4.6}$$

Примерът за работата на BFS, който видяхме преди малко, показваше—това е доста неформално казано—опашката като тунел, в който движението на върховете е **отляво надясно**. От друга страна можем да си представим, че върховете са в неподвижна редица, а тунелът се движи над тях в посока **отдясно наляво**; (4.6) илюстрира точно тази представа.

Фигура 4.13 показва опашката, представена чрез зелена правоъгълна кутия, в избрани моменти от работата на BFS върху графа от Фигура 4.12, като опашката се движи над върховете отдясно наляво. Под името на всеки връх е написано разстоянието между него и връх 1 в синьо (Теорема 51 казва, че това разстояние е равно на d стойността на върха в края на алгоритъма). Цветовете на върховете във всеки момент точно съответстват на състоянията им:

- в началото всички върхове са бели,
- после белите върхове са тези, които не са били в опашката, сивите са тези в опашката, а черните са тези, които са излезли от опашката
- в края всички върхове са черни.

Фигура 4.13 : Опашката по време на BFS(G), където G е графът от Фиг. 4.12.

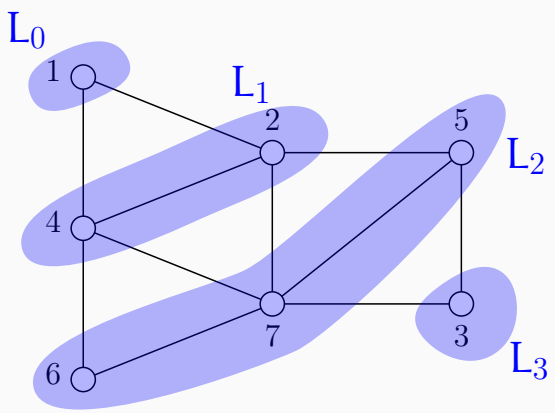


Фигура 4.13 навежда на мисълта, че BFS обработва върховете по нарастващи разстояния от избрания начален връх. Лема 23 доказва това формално. Трябват ни няколко дефиниции. Нека входът G е неориентиран и свързан. Нека $d_{max} = \max\{\mathrm{dist}(1,i) \mid 1 \leqslant i \leqslant n\}$. Очевидно е, че за всяко $k \in \{0,\ldots,d_{max}\}$, в G има поне един връх на разстояние k от връх 1. Тогава можем да дефинираме разбиване $\{L_0, L_1, \ldots, L_{d_{max}}\}$ на $V(G)$ по разстоянията от връх 1 така:

$$\forall k \in \{0,\ldots,d_{max}\} : L_k = \{j \in V(G) \mid \mathrm{dist}(1,j) = k\}$$

Ще казваме, че L_k е k-*ият слой по отношение на връх* 1. Фигура 4.14 показва слоевете на графа от Фигура 4.12.

Фигура 4.14 : Слоевете на графа от Фигура 4.12.



Очевидно е, че за всеки връх i, $d[i]$ се инициализира с ∞, а впоследствие $d[i]$ получава окончателната си стойност, която е естествено число, или на ред 3, или на ред 12; веднъж получена, тя не се променя до края на работата на алгоритъма. За всяко $k \in \{0, \ldots, d_{max}\}$, нека t_k е първият момент от работата на BFS, при който всички върхове от L_k са получили своята окончателна d стойност и изпълнението е на ред 7. Ясно е, че t_0, t_1, $\ldots$, $t_{d_{max}}$ са добре дефинирани. Също така е ясно, че $t_0 < t_1 < \cdots < t_{d_{max}}$, но това вече няма да приемаме за очевидно, а ще докажем. Нека $\mathcal{T} = \{t_0, t_1, \ldots, t_{d_{max}}\}$.

Още дефинираме, че във всеки момент от работата на алгоритъма, $V(Q)$ е множеството от върховете, които се съдържат в Q. Още дефинираме, че $T(\pi, k)$ означава кореновото дърво с множество от върхове $L_0 \cup \cdots \cup L_k$, което бива реализирано в момент t_k от масива π; тъй като в общия случай $L_0 \cup \cdots \cup L_k \subset V(G)$, става дума само за тези елементи на π, чиито индекси са върхове от $L_0 \cup \cdots \cup L_k$. Тъй като $L_0 \cup \cdots \cup L_{d_{max}} = V(G)$, вярно е, че $T(\pi, d_{max})$ е окончателното дърво на обхождането.

Лема 23: За доказателството на Теорема 51

За всяко $k \in \{0, \ldots, d_{max}\}$, в момента t_k:

1. $\bigcup_{0 \leqslant i \leqslant k-1} L_i$ съдържа точно черните върхове, L_k съдържа точно сивите върхове и $L_k = V(Q)$, а $\bigcup_{k+1 \leqslant i \leqslant d_{max}} L_i$ съдържа точно белите върхове.
2. $\forall i \in L_0 \cup \cdots \cup L_k : d[i] = dist_G(1, i) = dist_{T(\pi,k)}(1, i)$.
3. Моментите от $\mathcal{T}$, които са се случили, са t_0, t_1, $\ldots$, t_k, и то в този ред.

Доказателство: Ще докажем твърдението с индукция по k, за $k \in \{0, \ldots, d_{max}\}$. Забележете, че правим доказателство по индукция върху крайно множество.

База $k = 0$. L_k е L_0, а очевидно $L_0 = \{1\}$. Моментът t_k е t_0: моментът, в който изпълнението за пръв път е на ред 7. Очевидно това е единственият момент от $\mathcal{T}$, който се е случил, така че 3 е вярно.

Ще докажем 1. От една страна, множеството $\bigcup_{0 \leqslant i \leqslant -1} L_i$ е празното множество, а от друга страна от псевдокода ясно се вижда, че в този момент черни върхове няма. От една страна, L_0 е $\{1\}$, а от друга страна, от псевдокода ясно се вижда, че в този момент единственият сив връх е 1, а също така $V(Q) = \{1\}$. От една страна, $\bigcup_{1 \leqslant i \leqslant d_{max}} L_i$ е $V(G) \backslash \{1\}$, а от друга страна, в този момент от псевдокода ясно се вижда, че в този момент всички върхове без 1 са бели.

Ще докажем 2. Тъй като $L_0 \cup \cdots \cup L_k$ е L_0, което е $\{1\}$, трябва да се покаже, че $d[1] = dist_G(1, 1) = dist_{T(\pi,0)}(1, 1)$. От една страна, $d[1]$ заради присвояването на ред 4, а от друга страна, $dist_G(1, 1) = 0$ (Наблюдение 11); следователно, $d[1] = dist_G(1, 1)$. Да разгледаме $T(\pi, 0)$. Това е кореновото дърво с единствен връх 1, който се явява и корен, така че масивът π го представлява чрез първата си клетка, в която има Nil заради първия **for**-цикъл (редове 1–2). Вярно е, че $dist_{T(\pi,0)}(1, 1) = 0$ (Наблюдение 11). Базата "излиза". ✓

Индуктивно предположение Да допуснем, че конюнкцията от 1, 2 и 3 е истина за някое $k \in \{0, \ldots, d_{max} - 1\}$.

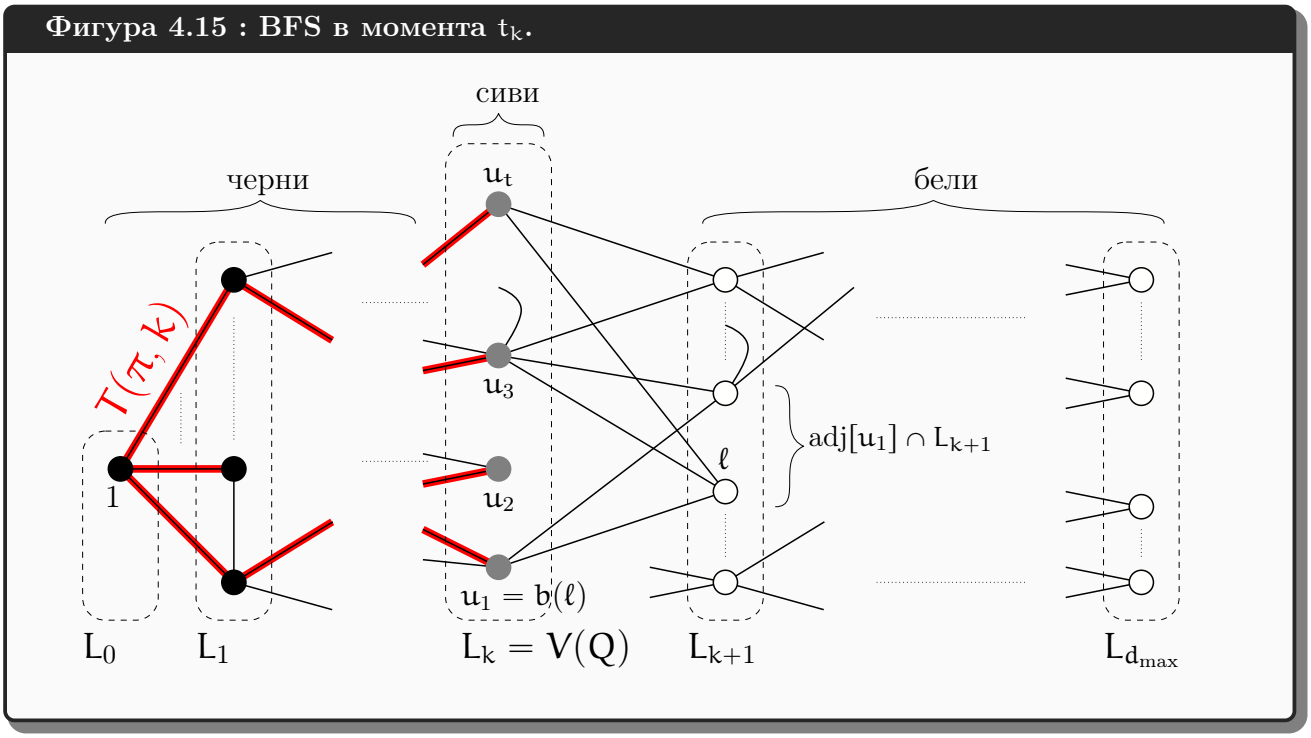
Индуктивна стъпка Ще разгледаме работата на алгоритъма от момента t_k нататък. Ще покажем, че в момент t_{k+1} твърденията 1, 2 и 3 са верни за стойност на аргумента $k + 1$. Подробно формално доказателство тук би използвало втора, вложена индукция, за тази цел, заради вложеността на цикли. Това би направило цялото доказателство прекалено тежко за четене, така че ще има известна непрецизност в изложението.

Разглеждаме момента t_k. Моментите t_0, t_1, …, t_{k-1}, а t_{k+1}, t_{k+2}, …, $t_{d_{max}}$ още не са се случили вече са се случили съгласно част 3 от индуктивното предположение.

Фигура 4.15 илюстрира състоянието на нещата в момента t_k. L_0, …, L_{k-1} са точно черните върхове, L_k са сивите върхове и L_{k+1}, …, $L_{d_{max}}$ са белите върхове. Щом $k < d_{max}$, то $L_{k+1} \neq \varnothing$; ерго, бели върхове има. Върховете в Q са точно върховете от L_k и за удобство са именувани u_1, …, u_t. Да кажем, че в Q те са наредени именно в този ред, така че първо ще бъде изваден u_1, после u_2, и така нататък, после u_t.

$T(\pi, k)$ е очертано в червено. Неговите листа не са непременно върховете от L_k, въпреки че всеки връх от L_k се явява негово листо; може в L_0, …, L_{k-1} да има върхове, които нямат деца по отношение на $T(\pi, k)$. На фигурата е показан такъв връх в L_1.

Фигура 4.15 : BFS в момента t_k.



Ключово наблюдение е, че всеки връх от L_{k+1} има съсед в L_k, инак не би бил в L_{k+1}. Обратното не е вярно – в L_k може да има върхове, които нямат съседи в L_{k+1}, като u_2 на фигурата. Щом всеки връх в L_{k+1} има съсед в L_k, всеки връх в L_{k+1} е в поне едно—но може и в повече от едно—множество $\mathrm{adj}[u_i]$. Тогава множеството $\{\mathrm{adj}[u_i] \cap L_{k+1} \mid 1 \leqslant i \leqslant t\}$ е покриване на L_{k+1}.

Друго ключово наблюдение е, че нито един връх от L_k не е съсед на връх от $L_{k+2} \cup \cdots \cup L_{d_{max}}$, така че множествата $\{\mathrm{adj}[u_i] \cap L_s \mid 1 \leqslant i \leqslant t\}$, за $k+2 \leqslant s \leqslant d_{max}$, са празни.

От тези две наблюдения следва, че **while**-цикълът (редове 7–15), изваждайки върховете u_1, …, u_t един по един от опашката, слагайки ги в променливата x (ред 8), чрез вложения **for**-цикъл (редове 9–14) слага всеки връх ℓ от L_{k+1} (но не и от L_{k+2}, …, $L_{d_{max}}$) в променливата y (ред 9) в някакъв момент t_ℓ, като очевидно $t_k < t_\ell < t_{k+1}$. Тъй като ℓ има бял цвят в момента t_ℓ, условието на ред 10 е истина и тялото на **if**-а (редове 11–14) се изпълнява. Тогава в сила са следните твърдения.

- ℓ става сив връх на ред 11.
- $d[\ell]$ става $d[b(\ell)] + 1$ на ред 12, но от индуктивното допускане знаем, че $d[b(\ell)] = k$, така че $d[\ell]$ става $k+1$. Но $\mathrm{dist}_G(1, \ell)$ е именно $k+1$, защото $\ell \in L_{k+1}$ по конструкция.

- $\pi(\ell)$ става $b(\ell)$ на ред 13, при което масивът π вече реализира дърво, в което разстоянието между 1 и ℓ е $k+1$, съвпадайки с $\mathrm{dist}_G(1,\ell)$.
- ℓ влиза в опашката на ред 14.

Когато и последният връх от L_{k+1} "мине" през y, стане сив и получи d стойност $k+1$ и родител-връх от L_k чрез π, изпълнението отива на ред 7 и това е моментът t_{k+1}. Заключаваме, че 3 е изпълнено за аргумент $k+1$.

Всички върхове от L_k са черни в t_{k+1}, така че черните са точно върховете от $\bigcup_{0\leqslant i\leqslant k} L_i$. Върховете от L_{k+1} са точно сивите върхове и те са точно върховете в Q, така че $V(Q) = L_{k+1}$. Белите върхове са точно върховете от $\bigcup_{k+2\leqslant i\leqslant d_{max}} L_i$. Заключаваме, че 1 е изпълнено за аргумент $k+1$.

И накрая, забелязваме, че $\forall i \in L_{k+1} : d[i] = \mathrm{dist}_G(1,i) = \mathrm{dist}_{T(\pi,k+1)}(1,i)$. Нещо повече, алгоритъмът не е променил нито d стойността, нито π стойността на връх от $L_0 \cup \cdots \cup L_k$ между моментите t_k и t_{k+1}, защото тези върхове не са бели по това време, така че **if**-ът ги прескача. Тогава $\forall i \in L_0 \cup \cdots \cup L_{k+1} : d[i] = \mathrm{dist}_G(1,i) = \mathrm{dist}_{T(\pi,k+1)}(1,i)$. Заключаваме, че 2 е изпълнено за аргумент $k+1$. □

Следствие 16

При приключване на работата на BFS, в сила е $t_0 < t_1 < \cdots < t_{d_{max}}$.

Теорема 51: BFS намира разстоянията в графа

Нека $G = (V,E)$ е свързан неориентиран граф. Нека е изпълнен BFS(G), като T е дървото на обхождането. След приключване на алгоритъма, за всеки връх $i \in V$ е вярно, че $d[i] = \mathrm{dist}_G(1,i)$ и $\mathrm{dist}_G(1,i) = \mathrm{dist}_T(1,i)$.

Доказателство: Ползваме Лема 23. Разглеждаме момента $t_{d_{max}}$. Съгласно нея, в този момент е вярно, че $\forall i \in L_0 \cup \cdots \cup L_{d_{max}}$ е вярно, че $d[i] = \mathrm{dist}_G(1,i)$ и $\mathrm{dist}_G(1,i) = \mathrm{dist}_T(1,i)$. Но $L_0 \cup \cdots \cup L_{d_{max}} = V$. □

4.2.2 DFS

DFS е алгоритъмът за обхождане на графи в дълбочина. Името идва от **D**epth-**F**irst **S**earch. За разлика от BFS, който е построен директно върху Схема 1 с реализация на множеството S чрез опашка, DFS не се получава от Схема 1 директно – факт, който обосноваваме на стр. 297. Въпреки че DFS не се получава директно от Схема 1, ние ще ползваме нейното доказателство за коректност и за него.

DFS наистина е свързан със стекова структура (LIFO), но имплементацията, която ще разгледаме, е рекурсивен алгоритъм, който ползва неявно системния стек.

Интуитивно казано, DFS обхожда графа по начин, обратен на BFS. BFS е "предпазливото" обхождане, при което обхождаме слой по слой навън от избрания начален връх. DFS е "смелото" обхождане, при което—ако ползваме аналогията с лабиринта—попадайки в коя да е стая,

- излизаме през първата неползвана врата, ако има такава, и отиваме в стая, в която или не сме били, или вече сме били:
  - ако не сме били, маркираме я като посетена и продължаваме по същия начин,

  - а ако сме били, се връщаме обратно

- ако вече няма неизползвана врата, се връщаме колкото е възможно по-малко и опитваме същото нещо, ако е възможно; когато сме отново в началната стая и вече няма неизползвани врати, прекратяваме алгоритъма.

Реализацията на DFS, която ще разгледаме, ползва масив от цветове за върховете със същия смисъл като при BFS и също изгражда дърво на обхождането. За разлика от BFS, DFS не изчислява разстояния в графа; напълно възможно е да въведем и при него някакви слоеве от върхове, но те не биха били от равноотдалечени от началния връх върхове, за разлика от BFS. По правило, върху един и същи граф, BFS и DFS изграждат различни дървета, като това на BFS е по-разклонено и ниско. Нещо повече. Ако графът е представен със списъци, DFS е по-чувствителен към разликите в наредбите на върховете в списъците, като е възможно при различни такива наредби получените дървета да имат различни височини (което при BFS е невъзможно).

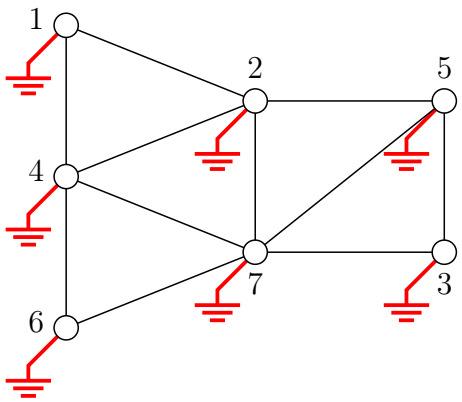
По отношение на въпроса, каква част от графа обхожда DFS, отговорът точно същият като за BFS.

DFS работи в общия случай върху ориентиран мултиграф с възможни примки, също като BFS.

```
DFS(G = (V, E), като V = {1, ..., n})
1  for i ← 1 to n
2      colour[i] ← white
3      π[i] ← Nil
4  DFS-VISIT(G, 1)
5  return colour, π
```

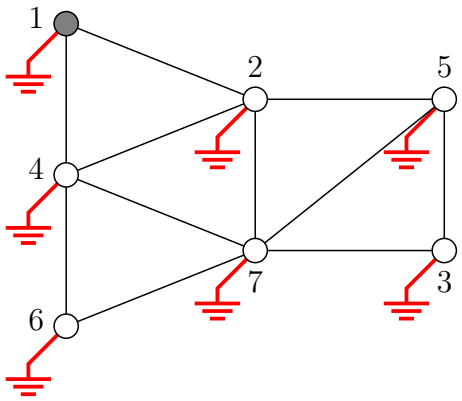
```
DFS-VISIT(G = (V, E); x ∈ V)
1  colour[x] ← grey
2  foreach y ∈ adj[x]
3      if colour[y] = white
4          π[y] ← x
5          DFS-VISIT(G, y)
6  colour[x] ← black
```

Ще разгледаме подробно работата на DFS върху графа от Фигура 4.12, за да направим съпоставка с BFS. Трябва да имаме предвид, че алгоритъмът е рекурсивен, така че променливите x и y на функцията DFS-VISIT, която може да вика себе си, в различните нивá на рекурсията имат различни стойности. Резултатът от инициализацията изглежда по следния начин (подобен на инициализацията на BFS, но без d стойностите):



1 : 4, 2
2 : 5, 1, 7, 4
3 : 5, 7
4 : 6, 1, 2, 7
5 : 2, 7, 3
6 : 7, 4
7 : 5, 2, 4, 3, 6

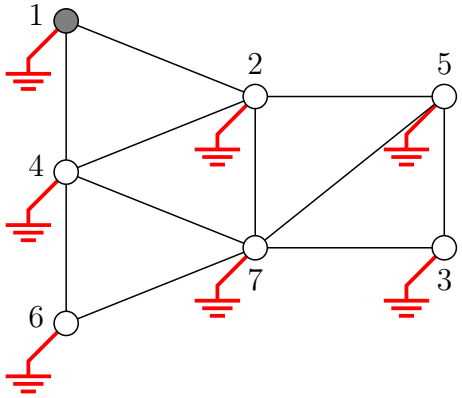
Следва първото викане на рекурсивната функция, при което променливата x съдържа 1. Връх 1 става сив:



1 : 4, 2
2 : 5, 1, 7, 4
3 : 5, 7
4 : 6, 1, 2, 7
5 : 2, 7, 3
6 : 7, 4
7 : 5, 2, 4, 3, 6

$x = 1$

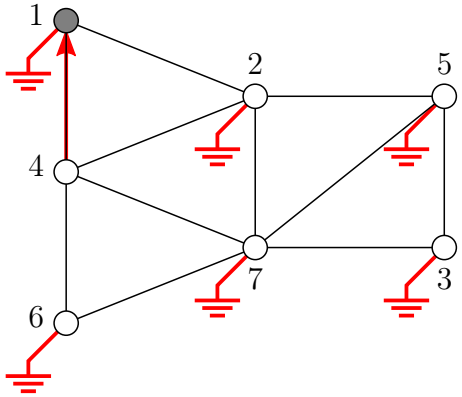
Съседите на x са 4 и 2, в този ред в списъка. Първо y става 4:



1 : 4, 2
2 : 5, 1, 7, 4
3 : 5, 7
4 : 6, 1, 2, 7
5 : 2, 7, 3
6 : 7, 4
7 : 5, 2, 4, 3, 6

$x = 1 \quad y = 4$

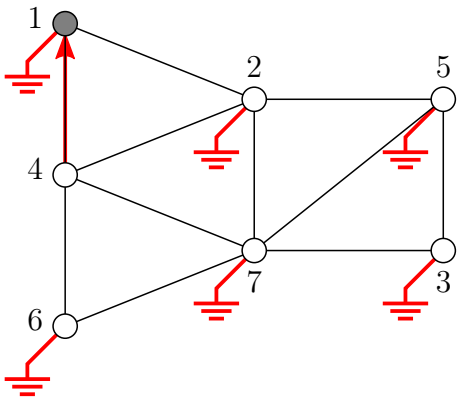
Тъй като цветът на y е бял, изпълнението продължава с промяна на родителя на y на връх 1 на ред 3 на DFS-VISIT:



1 : 4, 2
2 : 5, 1, 7, 4
3 : 5, 7
4 : 6, 1, 2, 7
5 : 2, 7, 3
6 : 7, 4
7 : 5, 2, 4, 3, 6

$x = 1 \quad y = 4$

После DFS-VISIT вика себе си на ред 4, този път с втори аргумент връх 4. Ерго, във второто викане $x = 4$, но първото викане още не е приключило, така че x съществува едновременно в две различни инкарнации, а именно $x = 1$ в първото викане и $x = 4$ във второто викане:

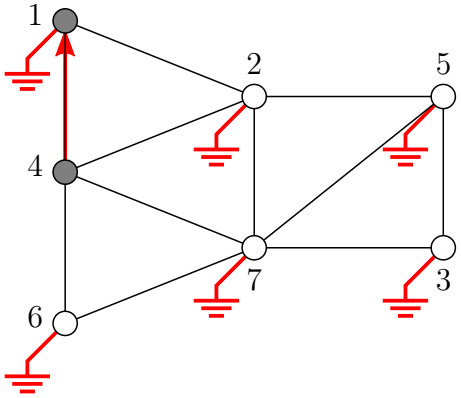


1 : 4, 2
2 : 5, 1, 7, 4
3 : 5, 7
4 : 6, 1, 2, 7
5 : 2, 7, 3
6 : 7, 4
7 : 5, 2, 4, 3, 6

$x = 1 \quad y = 4$

$x = 4$

На ред 1 цветът на 4 става сив:

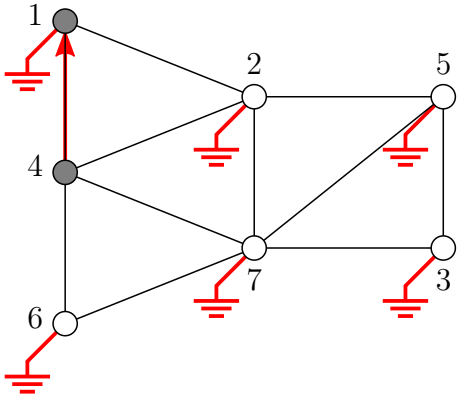


1 : 4, 2
2 : 5, 1, 7, 4
3 : 5, 7
4 : 6, 1, 2, 7
5 : 2, 7, 3
6 : 7, 4
7 : 5, 2, 4, 3, 6

$x = 1 \quad y = 4$

$x = 4$

Пръв в списъка на връх 4 е връх 6. y става 6:

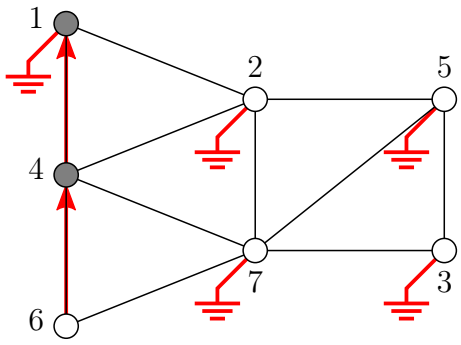


1 : 4, 2
2 : 5, 1, 7, 4
3 : 5, 7
4 : 6, 1, 2, 7
5 : 2, 7, 3
6 : 7, 4
7 : 5, 2, 4, 3, 6

$x = 1 \quad y = 4$

$x = 4 \quad y = 6$

Тъй като 6 е бял, DFS-VISIT слага 4 в $\pi[6]$ и вика себе си с втори аргумент 6:



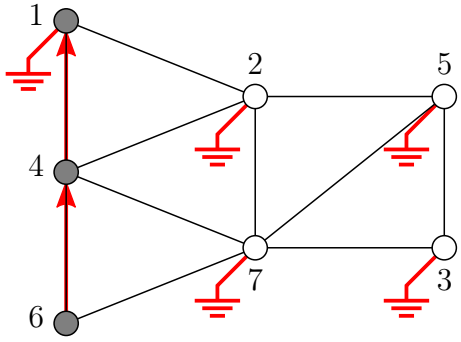
1 : 4, 2
2 : 5, 1, 7, 4
3 : 5, 7
4 : 6, 1, 2, 7
5 : 2, 7, 3
6 : 7, 4
7 : 5, 2, 4, 3, 6

$x = 1 \quad y = 4$

$x = 4 \quad y = 6$

$x = 6$

Вече x съществува в три инкарнации. Връх 6 става сив:



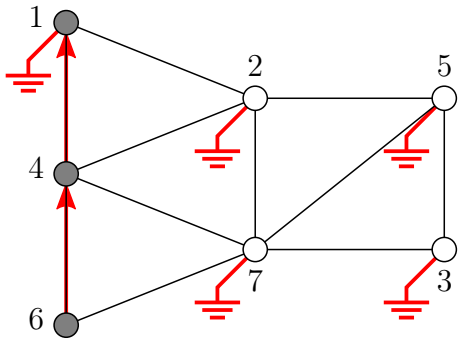
1 : 4, 2
2 : 5, 1, 7, 4
3 : 5, 7
4 : 6, 1, 2, 7
5 : 2, 7, 3
6 : 7, 4
7 : 5, 2, 4, 3, 6

$x = 1 \quad y = 4$

$x = 4 \quad y = 6$

$x = 6$

y става 7:



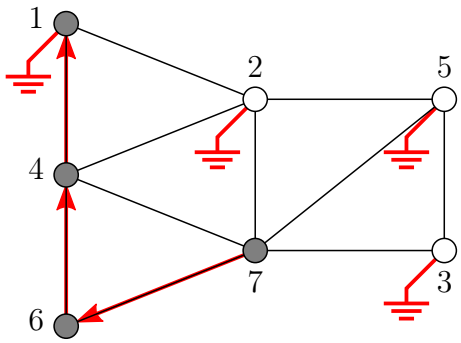
1 : 4, 2
2 : 5, 1, 7, 4
3 : 5, 7
4 : 6, 1, 2, 7
5 : 2, 7, 3
6 : 7, 4
7 : 5, 2, 4, 3, 6

$x = 1 \quad y = 4$

$x = 4 \quad y = 6$

$x = 6 \quad y = 7$

Тъй като 7 е бял, DFS-VISIT слага 6 в $\pi[7]$ и вика себе си с втори аргумент 7:



1 : 4, 2
2 : 5, 1, 7, 4
3 : 5, 7
4 : 6, 1, 2, 7
5 : 2, 7, 3
6 : 7, 4
7 : 5, 2, 4, 3, 6

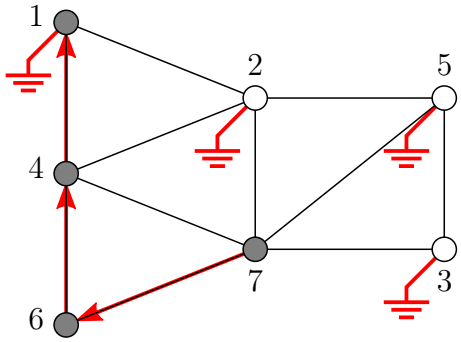
$x = 1 \quad y = 4$

$x = 4 \quad y = 6$

$x = 6 \quad y = 7$

$x = 7$

y става 5:



1 : 4, 2
2 : 5, 1, 7, 4
3 : 5, 7
4 : 6, 1, 2, 7
5 : 2, 7, 3
6 : 7, 4
7 : 5, 2, 4, 3, 6

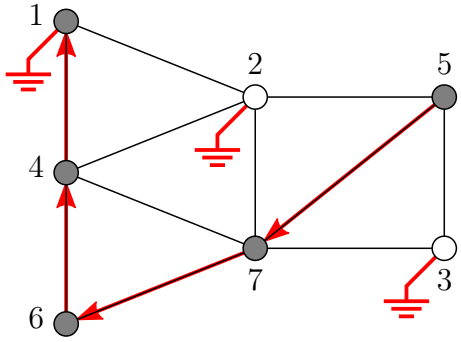
$x = 1 \quad y = 4$

$x = 4 \quad y = 6$

$x = 6 \quad y = 7$

$x = 7 \quad y = 5$

Тъй като 5 е бял, DFS-VISIT слага 7 в $\pi[5]$ и вика себе си с втори аргумент 5:



1 : 4, 2
2 : 5, 1, 7, 4
3 : 5, 7
4 : 6, 1, 2, 7
5 : 2, 7, 3
6 : 7, 4
7 : 5, 2, 4, 3, 6

$x = 1 \quad y = 4$

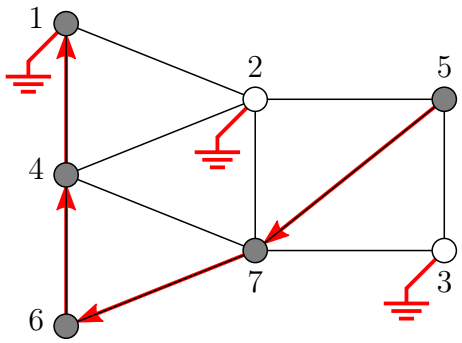
$x = 4 \quad y = 6$

$x = 6 \quad y = 7$

$x = 7 \quad y = 5$

$x = 5$

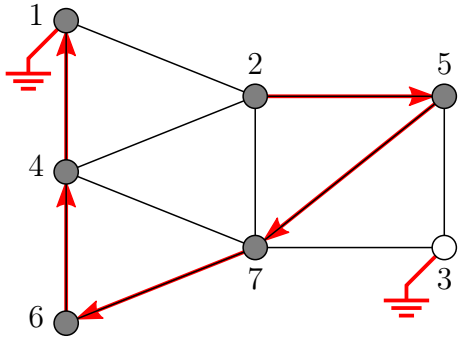
y става 2:



1 : 4, 2
2 : 5, 1, 7, 4
3 : 5, 7
4 : 6, 1, 2, 7
5 : 2, 7, 3
6 : 7, 4
7 : 5, 2, 4, 3, 6

x = 1	y = 4
x = 4	y = 6
x = 6	y = 7
x = 7	y = 5
x = 5	y = 2

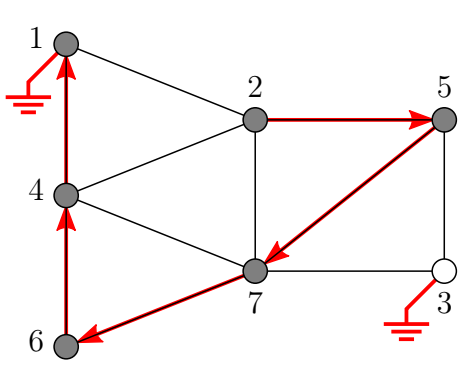
Тъй като 2 е бял, DFS-VISIT слага 5 в $\pi[2]$ и вика себе си с втори аргумент 2:



1 : 4, 2
2 : 5, 1, 7, 4
3 : 5, 7
4 : 6, 1, 2, 7
5 : 2, 7, 3
6 : 7, 4
7 : 5, 2, 4, 3, 6

x = 1	y = 4
x = 4	y = 6
x = 6	y = 7
x = 7	y = 5
x = 5	y = 2
x = 2	

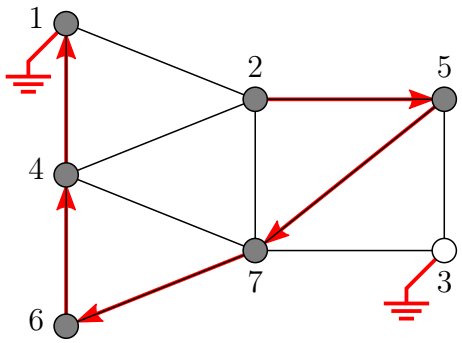
y става 5:



1 : 4, 2
2 : 5, 1, 7, 4
3 : 5, 7
4 : 6, 1, 2, 7
5 : 2, 7, 3
6 : 7, 4
7 : 5, 2, 4, 3, 6

x = 1	y = 4
x = 4	y = 6
x = 6	y = 7
x = 7	y = 5
x = 5	y = 2
x = 2	y = 5

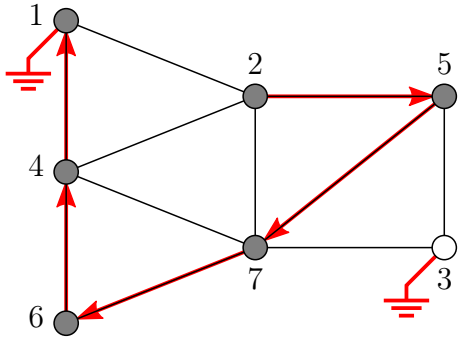
Обаче 5 е сив и условието на **if**-а на ред 3 е лъжа, така че тялото на **if**-а не се изпълнява. След това y става 1:



1 : 4, 2
2 : 5, 1, 7, 4
3 : 5, 7
4 : 6, 1, 2, 7
5 : 2, 7, 3
6 : 7, 4
7 : 5, 2, 4, 3, 6

x = 1	y = 4
x = 4	y = 6
x = 6	y = 7
x = 7	y = 5
x = 5	y = 2
x = 2	y = 1

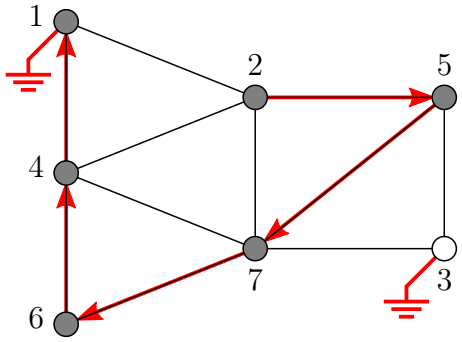
Но 1 също е сив, така че тялото на **if**-а пак не се изпълнява. След това y става 7:



1 : 4, 2
2 : 5, 1, 7, 4
3 : 5, 7
4 : 6, 1, 2, 7
5 : 2, 7, 3
6 : 7, 4
7 : 5, 2, 4, 3, 6

x = 1	y = 4
x = 4	y = 6
x = 6	y = 7
x = 7	y = 5
x = 5	y = 2
x = 2	y = 7

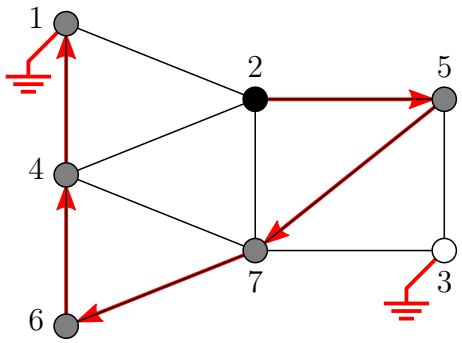
Но 7 също е сив, така че тялото на **if**-а пак не се изпълнява. След това y става 4:



1 : 4, 2
2 : 5, 1, 7, 4
3 : 5, 7
4 : 6, 1, 2, 7
5 : 2, 7, 3
6 : 7, 4
7 : 5, 2, 4, 3, 6

x = 1	y = 4
x = 4	y = 6
x = 6	y = 7
x = 7	y = 5
x = 5	y = 2
x = 2	y = 4

Но 4 също е сив, така че тялото на **if**-а пак не се изпълнява. **for**-цикълът приключва и изпълнението отива на ред 6, където 2 става черен:



1 : 4, 2
2 : 5, 1, 7, 4
3 : 5, 7
4 : 6, 1, 2, 7
5 : 2, 7, 3
6 : 7, 4
7 : 5, 2, 4, 3, 6

$x = 1 \quad y = 4$

$x = 4 \quad y = 6$

$x = 6 \quad y = 7$

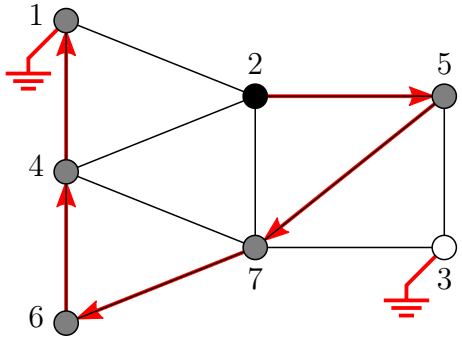
$x = 7 \quad y = 5$

$x = 5 \quad y = 2$

$x = 2$

Забележете, че при DFS първият връх, който става черен, е последният връх, който е бил направен сив!

След това DFS-VISIT излиза от текущото ниво на рекурсията и се връща на предното ниво. Отново $x = 5$, но сега $y = 7$:



1 : 4, 2
2 : 5, 1, 7, 4
3 : 5, 7
4 : 6, 1, 2, 7
5 : 2, 7, 3
6 : 7, 4
7 : 5, 2, 4, 3, 6

$x = 1 \quad y = 4$

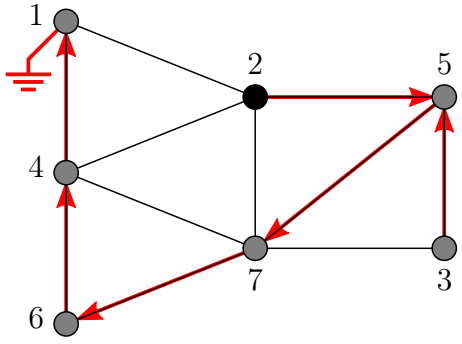
$x = 4 \quad y = 6$

$x = 6 \quad y = 7$

$x = 7 \quad y = 5$

$x = 5 \quad y = 7$

7 е сив, DFS-VISIT го прескача и y става 3. Но 3 е бял, така че тялото на **if**-а се изпълнява. $\pi[3]$ става 5 и имаме ново рекурсивно викане на DFS-VISIT с $x = 3$:



1 : 4, 2
2 : 5, 1, 7, 4
3 : 5, 7
4 : 6, 1, 2, 7
5 : 2, 7, 3
6 : 7, 4
7 : 5, 2, 4, 3, 6

$x = 1 \quad y = 4$

$x = 4 \quad y = 6$

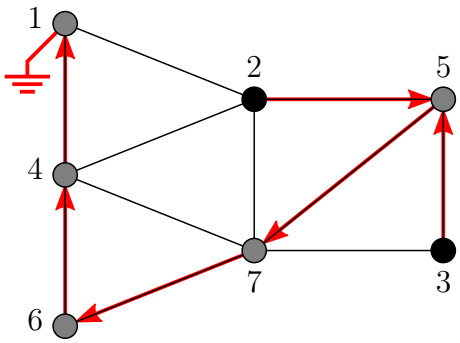
$x = 6 \quad y = 7$

$x = 7 \quad y = 5$

$x = 5 \quad y = 2$

$x = 3$

Тъй като и двата съседа на 3 са сиви, DFS-VISIT ги прескача, 3 става черен и текущото рекурсивно викане приключва. Отново $x = 5$, но сега вече $y = 7$:



1 : 4, 2
2 : 5, 1, 7, 4
3 : 5, 7
4 : 6, 1, 2, 7
5 : 2, 7, 3
6 : 7, 4
7 : 5, 2, 4, 3, 6

$x = 1 \quad y = 4$

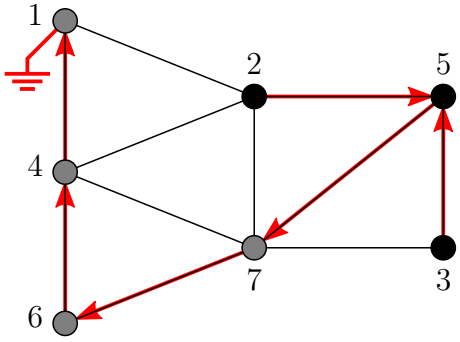
$x = 4 \quad y = 6$

$x = 6 \quad y = 7$

$x = 7 \quad y = 5$

$x = 5 \quad y = 7$

DFS-VISIT прескача и 7, и 3, и връх 5 става черен:



1 : 4, 2
2 : 5, 1, 7, 4
3 : 5, 7
4 : 6, 1, 2, 7
5 : 2, 7, 3
6 : 7, 4
7 : 5, 2, 4, 3, 6

$x = 1 \quad y = 4$

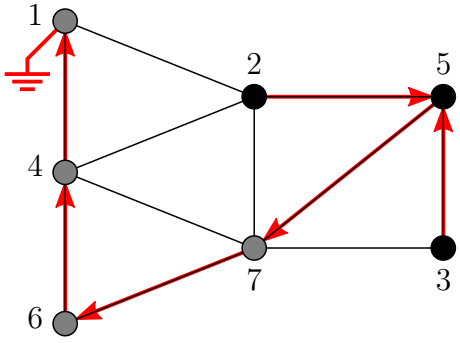
$x = 4 \quad y = 6$

$x = 6 \quad y = 7$

$x = 7 \quad y = 5$

$x = 5$

После DFS-VISIT излиза и от това ниво на рекурсията и отново $x = 7$, като сега $y = 2$:



1 : 4, 2
2 : 5, 1, 7, 4
3 : 5, 7
4 : 6, 1, 2, 7
5 : 2, 7, 3
6 : 7, 4
7 : 5, 2, 4, 3, 6

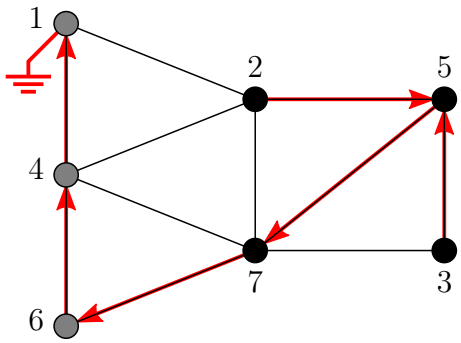
$x = 1 \quad y = 4$

$x = 4 \quad y = 6$

$x = 6 \quad y = 7$

$x = 7 \quad y = 2$

2 се прескача, после 4, 3 и 6 се прескачат и 7 става черен:



1 : 4, 2
2 : 5, 1, 7, 4
3 : 5, 7
4 : 6, 1, 2, 7
5 : 2, 7, 3
6 : 7, 4
7 : 5, 2, 4, 3, 6

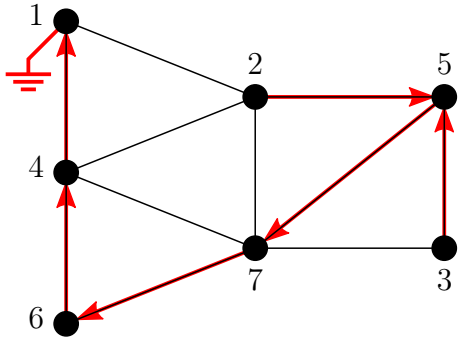
$x = 1 \quad y = 4$

$x = 4 \quad y = 6$

$x = 6 \quad y = 7$

$x = 7$

Лесно се вижда, че нататък DFS-VISIT само прескача y-ите, на всички нивá на рекурсията, и връща нагоре, докато накрая прави връх 1 черен:

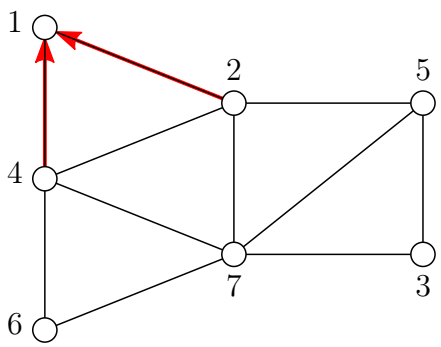


1 : 4, 2
2 : 5, 1, 7, 4
3 : 5, 7
4 : 6, 1, 2, 7
5 : 2, 7, 3
6 : 7, 4
7 : 5, 2, 4, 3, 6

$x = 1$

След това изпълнението се връща в основния алгоритъм DFS, който приключва работата, връщайки π и colour на ред 5.

И накрая да се убедим, че DFS не се получава от BFS при замяна на опашката със стек (което влече и замяна на Enqueue и Dequeue съответно с Push и Pop). Да разгледаме графа, върху който илюстрирахме и BFS, и DFS. Нека началният връх пак е 1. Да си представим работата на алгоритъм, получен от замяна на опашката със стек в BFS върху този граф, започвайки от връх 1. На първата итерация на **while**-цикълът върхове 2 и 4 влизат в стека, като родителят и на двата става връх 1:



Това се случва независимо от реда на върховете в списъка на 1. Е, ясно е, че след това начало никога не може да се получи дървото на обхождането, което получихме с DFS. Нещо повече: убедете се сами, се дървото на DFS, което получихме при симулирането на DFS върху този граф, започвайки от връх 1, не може да е такова, че родителят и на 2, и на 4 да е връх 1.

4.3 Минимални покриващи дървета

4.3.1 Въведение и дефиниции

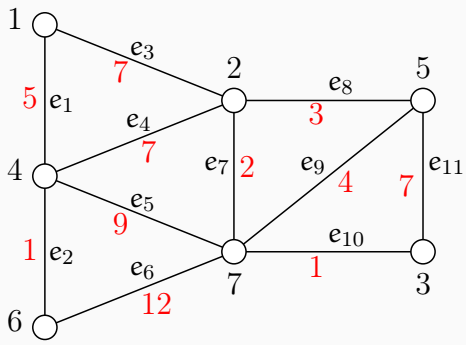
Разглеждаме неориентирани свързани тегловни графи. Наличие на примки или паралелни ребра не променя нищо по отношение на задачата, която разглеждаме, така че допускаме, че няма примки и няма паралелни ребра.

Нека $G = (V, E)$ е неориентиран свързан тегловен граф, като тегловната функция е $w : E \to \mathbb{R}$. Нека $\mathcal{T}$ е множеството от покриващите дървета на G. От Теорема 25 знаем, че, щом G е свързан, то $\mathcal{T} \neq \varnothing$. За всяко $T \in \mathcal{T}$ дефинираме *теглото на* T като

$$w(T) = \sum_{e \in E(T)} w(e)$$

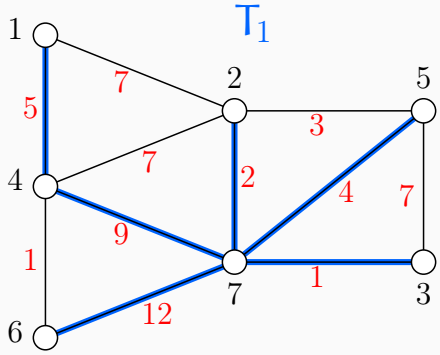
В тази секция ще разглеждаме примери върху графа, показан на Фигура 4.16. Теглата на ребрата са в червено.

Фигура 4.16 : Неориентиран свързан тегловен граф.

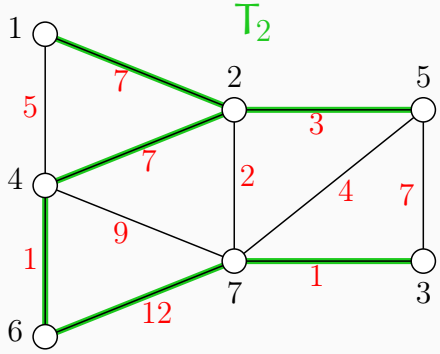


Фигура 4.17 показва две различни покриващи дървета на този граф: T_1 и T_2 с тегла съответно 33 и 31.

Фигура 4.17 : Различни покриващи дървета на графа от Фигура 4.16.



$w(T_1) = 33$



$w(T_2) = 31$

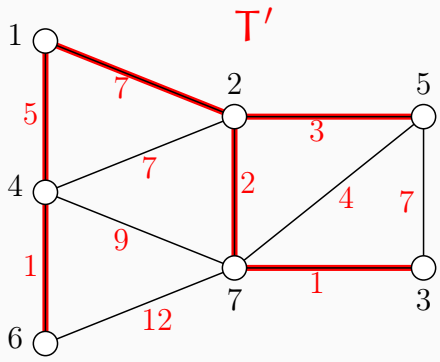
Определение 101: Минимално покриващо дърво (МПД)

Нека $G = (V, E)$ е неориентиран свързан тегловен граф, като тегловната функция е $w : E \to \mathbb{R}$. Нека $\mathcal{T}$ е множеството от покриващите дървета на G. *Минимално покриващо дърво* на G е всяко $T \in \mathcal{T}$, такова че

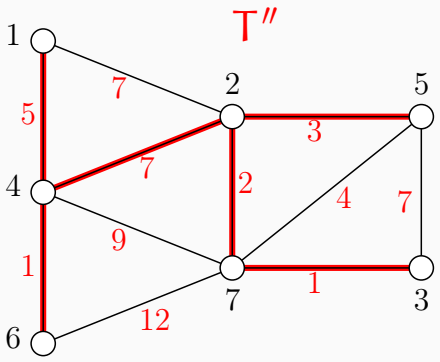
$$w(T) = \min\{w(D) \mid D \in \mathcal{T}\}$$

На английски терминът е *Minimum Spanning Tree*, накратко MST.
Очевидно има поне едно МПД, но може да има няколко МПД-та. В екстремния случай, в който всички ребра са с еднакви тегла, всяко покриващо дърво е МПД. Примерно, графът от Фигура 4.16 има две (различни) МПД-та, всяко с тегло 19, което е илюстрирано на Фигура 4.18.

Фигура 4.18 : Двете МПД-та на графа от Фигура 4.16.



$w(T') = 19$



$w(T'') = 19$

Разглеждаме алгоритмичната задача, при дадени G и w да се построи МПД. Ако графът не беше тегловен, задачата би била да се построи покриващо дърво на граф, което може да направим ефикасно с BFS или DFS. Ако теглата на ребрата са едни и същи, отново BFS или DFS дава ефикасно решение. Но в общия случай алгоритмите за обхождане **не решават** задачата за построяване на МПД и се налага да използваме специализирани алгоритми.

В подсекции 4.3.3 и 4.3.4 ще разгледаме два алгоритъма за построяване на МПД. Те са лесни за разбиране и за реализиране, но коректността им не е съвсем очевидна и и ще докажем чрез Теорема 52.

4.3.2 МПД теоремата

Първо да си припомним какво е "срез в граф", "срез-множество", "страни на среза" и "ребро, прекосяващо среза": вижте Определение 31 и Фигура 2.30 за илюстрация. Да въведем означения за краткост. Ако графът е $G = (V, E)$ и срезът е $\{U, W\}$, то с "$E_{U,W}$" ще означаваме срез-множеството, а с "$E^{\min}_{U,W}$" ще означаваме подмножеството на $E_{U,W}$ от ребрата с минимално тегло. Очевидно, $E_{U,W}$ и $E^{\min}_{U,W}$ са непразни, щом G е свързан и страните на среза са непразни.

Теорема 52: МПД теорема

Нека $G = (V, E)$ е неориентиран свързан тегловен граф, като тегловната функция е $w : E \to \mathbb{R}$. Нека $\{U, W\}$ е произволен срез в G. Тогава, за всяко $e \in E^{\min}_{U,W}$ съществува МПД D, което съдържа e.

Доказателство: Разглеждаме произволно $e \in E^{\min}_{U,W}$. Да допуснем, че няма МПД, което съдържа e. Но G е свързан и поне едно МПД има. Нека T е произволно МПД на G. По допускане, $e \notin E(T)$.

Да добавим e към T. Формално, това означава да построим графа $H = (V, E(T) \cup \{e\})$. H е уницикличен граф (вижте Определение 46). Нека c е името на цикъла в H. Нещо повече: $e \in E(c)$. Всеки връх на c е или в U, или в W, като поне един връх на c е в U и поне един връх на c е в W. Съгласно Наблюдение 9, съществуват две **различни** $e', e'' \in E(c)$, такива че единият край на всяко от тях е в U, а другият е в W. Тези две ребра може да имат или да нямат общ край, това няма значение. Поне едно от тях е различно от e. БОО, нека $e' \neq e$.

Да разгледаме графа $D = H - e'$. D е дърво, защото се получава от свързан граф с точно един цикъл чрез изтриване на ребро от цикъла; очевидно $H - e'$ е ацикличен и е свързан. Нещо повече, D е покриващо дърво на G, защото $V(D) = V$. Нещо повече, $e \in E(D)$.

Да сравним теглата на T и D. Дървото D се получава от T чрез добавяне на e и изтриване на e', така че

$$w(D) = w(T) + w(e) - w(e')$$

Но по конструкция e е ребро с минимално тегло измежду ребрата, прекосяващи среза. Ерго, $w(e') \geqslant w(e)$, така че $w(e) - w(e') \leqslant 0$. Но тогава

$$w(D) \leqslant w(T)$$

Първо да допуснем, че $w(D) < w(T)$. Но това е невъзможно, защото T е МПД на G по конструкция. Остава възможността $w(D) = w(T)$. Но тогава D също е МПД. Нещо повече, D е МПД, което съдържа реброто e, в противоречие с допускането, че нито едно МПД не съдържа e. □

4.3.3 Алгоритъм на Prim

Този алгоритъм е откриван повече от веднъж, от различни хора или групи хора, отдалечени във времето, без да знаят за вече извършената работа. Първоначално алгоритъмът е бил открит от чешките математици Jarník и Borůvka, като оригиналната статия от 1930 г. на чешки може да бъде свалена *от сайта на Czech Digital Mathematics Library*. През 50-те години на 20 век алгоритъмът е преоткрит от американския математик Robert Prim [49]. Тук ще го наричаме "алгоритъм на Prim", защото е добил популярност под това име.

Алгоритъмът на Prim за намиране на МПД на свързан неориентиран тегловен граф G далечно прилича на обхождане. Той

- започва от един стартов връх, да кажем връх 1, намира най-леко ребро $e_1 = (1, i)$ (очевидно e_1 е инцидентно с връх 1) и слага e_1 в дървото,
- после намира най-леко ребро $e_2 = (j, k)$, такова че точно единият от j, k е от $\{1, i\}$ (което означава, че другият е от $V(G) \backslash \{1, i\}$) и слага e_2 в дървото,
- и така нататък, докато не сложи точно $n - 1$ ребра в дървото, след което приключва и връща построеното дърво.

Накратко, МПД-то започва като един единствен връх (стартовият връх) и след това нараства итеративно, като на всяка итерация се сдобива с нов връх и ново ребро, докато не покрие целия граф. Съществената е, че по време на работата на алгоритъма, частта от МПД-то, която е построена до момента, е **едно дърво**; то става покриващо чак накрая, но във всеки момент е **едно дърво**, а не колекция от дървета, за разлика от алгоритъма на Kruskal на стр. 305.

Алгоритъм 8: Алгоритъм на Prim

Вход: неориентиран свързан граф $G = (V, E)$ с тегловна функция $w : E \to \mathbb{R}$.
Изход: МПД D на G.

❶ Конструирай $D = (\{1\}, \varnothing)$.

❷ Ако $V(D) = V$, върни D и край.

❸ В противен случай, нека E' е срез-множеството на среза $\{V(D), V \backslash V(D)\}$. Нека $E'_{\min}$ се състои от ребрата с минимално тегло в E'. Нека $e = (u, v)$ е произволно ребро от $E'_{\min}$, като $u \in V(D)$, което влече $v \notin V(D)$.
Направи $V(D) \leftarrow V(D) \cup \{v\}$, $E(D) \leftarrow E(D) \cup \{e\}$ и отиди на ❷.

Ще покажем, че алгоритъмът е коректен. Първо ще покажем, че е добре дефиниран: множеството E' в ❸ е непразно, иначе G не би бил свързан, така че и $E'_{\min}$ е непразно, откъдето следва, че реброто e е добре дефинирано.

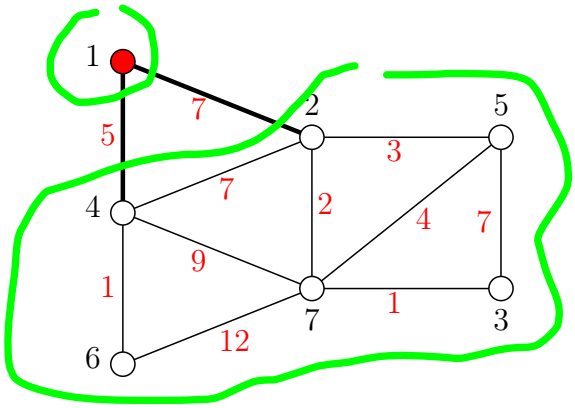
Това, че алгоритъмът строи **покриващ** подграф D, е очевидно от условието за край в ❷. Това, че D е дърво, се доказва тривиално по индукция по броя на достиганията на ❷.

- Базата е първото достигане, при което D, построено в ❶, очевидно е дърво.
- Допускаме, че достигаме ❷ с дърво D. Ако изпълнението продължи на ❸, към това дърво се добавят точно един нов връх и реброто между него и някой прежде добавен връх, и резултатът пак е дърво. Практически същата конструкция имахме в Определение 44, като в Теорема 22 показахме, че тя строи (само) дървета.

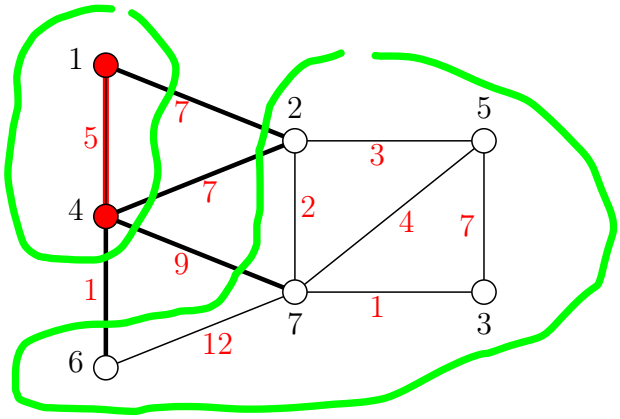
И така, алгоритъмът връща покриващо дърво.

Накрая ще покажем, че върнатото F е **минимално** покриващо дърво. Но това следва веднага от Теорема 52, приложена към избраното e във всяко изпълнение на ❸: срезът е $\{V(D), V \backslash V(D)\}$, срез-множеството е E', най-леките ребра от срез-множеството са точно елементите на $E'_{\min}$, а Теорема 52 казва, че всяко от тях е елемент на някое МПД, така че не може да сбъркаме, което и от тях да вземем.

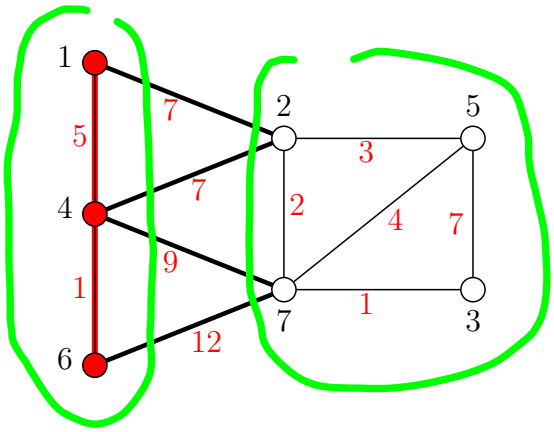
Ето симулация на алгоритъма на Prim върху графа от Фигура 4.16. В началото частично построеното МПД се състои само от връх 1, което задава среза $\{\{1\}, \{2, 3, 4, 5, 6, 7\}\}$. Срез-множеството е нарисувано удебелено.



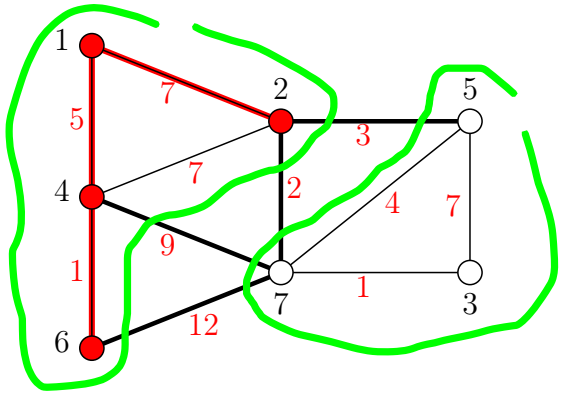
Срез-множеството се състои от $(1,2)$ с тегло 7 и $(1,4)$ с тегло 5. Избираме $(1,4)$, като добавяме връх 4 към $\mathsf{V}(\mathsf{T})$ и $(1,4)$ към A. Нещата изглеждат така:



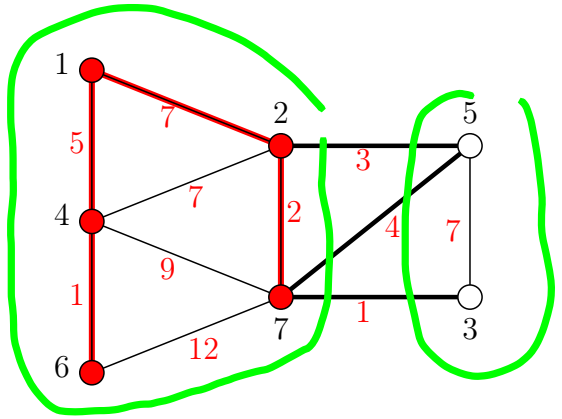
Сега срезът е $\{\{1,4\},\{2,3,5,6,7\}\}$. Най-леко ребро, прекосяващо среза, е $(4,6)$ с тегло 1. Връх 6 се добавя към $\mathsf{V}(\mathsf{T})$, а $(4,6)$ се добавя към A. Нещата изглеждат така:



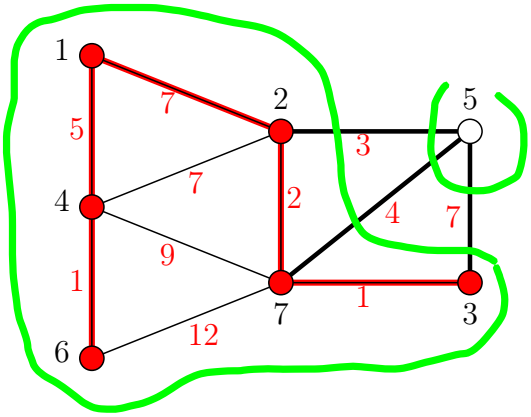
Сега има две най-леки ребра, прекосяващи среза. Измежду тях избираме произволно $(1,2)$. Връх 2 се добавя към $\mathsf{V}(\mathsf{T})$, а $(1,2)$ се добавя към A. Нещата изглеждат така:



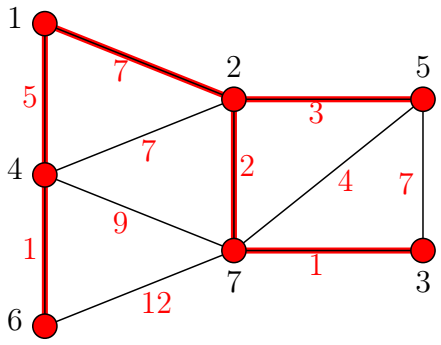
Сега най-лекото ребро, прекосяващо среза, е $(2,7)$ с тегло 2. Връх 7 се добавя към $\mathsf{V}(\mathsf{T})$, а $(2,7)$ се добавя към A. Нещата изглеждат така:



Сега най-лекото ребро, прекосяващо среза, е $(3,7)$ с тегло 1. Връх 3 се добавя към $\mathsf{V}(\mathsf{T})$, а $(3,7)$ се добавя към A. Нещата изглеждат така:



Сега най-лекото ребро, прекосяващо среза, е $(2,5)$ с тегло 3. Връх 5 се добавя към $\mathsf{V}(\mathsf{T})$, а $(2,5)$ се добавя към A. Нещата изглеждат така:



Алгоритъмът приключва.

4.3.4 Алгоритъм на Kruskal

Алгоритъмът на Kruskal е ефикасен алгоритъм за конструиране на МПД, който е основан на съвсем различна идея от тази на алгоритъма на Prim.

Алгоритъм 9: Алгоритъм на Kruskal

Вход: неориентиран свързан граф $G = (V, E)$ с тегловна функция $w : E \to \mathbb{R}$.
Изход: МПД F на G.

❶ Конструирай покриваща гора[a] $F = (V, \varnothing)$ на G.

❷ Ако F има точно една свързана компонента, върни F и край.

❸ В противен случай, нека $E' \subseteq E$ са точно тези ребра, чиито краища са от различни свързани компоненти (дървета) на F. Нека $E'_{\min}$ се състои от ребрата с минимално тегло в E'. Нека $e = (u, v)$ е произволно ребро от $E'_{\min}$. Нека T' и T'' са тези дървета в F, за които $u \in V(T')$ и $v \in V(T'')$.
Конструирай дървото $D = (V(T') \cup V(T''), E(T') \cup E(T'') \cup \{e\})$. Във F, замени дърветата T' и T'' с D и отиди на ❷.

[a] Покриващ подграф на G е всеки подграф на G със същото множество върхове като G. Вижте Определение 10. Покриващият подграф не е непременно дърво. В ❶ е колекция от дървета, тоест, гора, и то такава, която няма ребра.

Ще покажем, че алгоритъмът е коректен. Първо ще покажем, че е добре дефиниран: множеството E' в ❸ е непразно, иначе G не би бил свързан, така че и $E'_{\min}$ е непразно, откъдето следва, че реброто e е добре дефинирано.

След това ще покажем, че F, което алгоритъмът връща, е покриващо дърво. Но това се доказва тривиално по индукция по броя на достиганията на ❷.

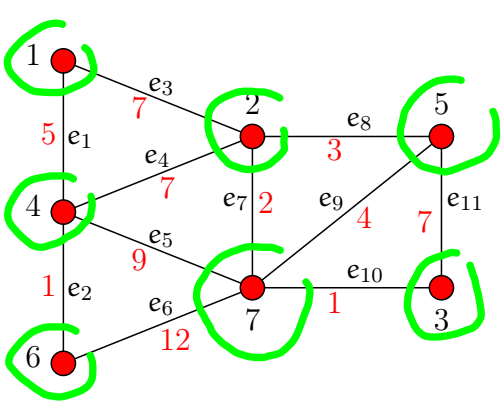
- Базата е първото достигане, при което F е покриваща гора с n дървета от ❶.
- Допускайки, че достигаме ❷ с гора F, която има k дървета, като $k > 1$, веднага виждаме, че D от ❸ е дърво (свързан граф без цикли, което следва от това, че T' и T'' са свързани графи без цикли, "залепени" едно за друго чрез единствено ребро e, което не е от нито едно от тях) и че при следващото достигане на ❷, F е покриваща гора с $k - 1$ дървета.

И така, алгоритъмът връща покриваща гора с 1 дърво, тоест, покриващо дърво.

Накрая ще покажем, че върнатото F е **минимално** покриващо дърво. Но това следва веднага от Теорема 52, приложена към избраното e във всяко изпълнение на ❸: срезът е $\{V(\mathsf{T}'), V \backslash V(\mathsf{T}')\}^\dagger$, срез-множеството е E', най-леките ребра от срез-множеството са точно елементите на $\mathsf{E}'_{\min}$, а Теорема 52 казва, че всяко от тях е елемент на някое МПД, така че не може да сбъркаме, което и от тях да вземем.

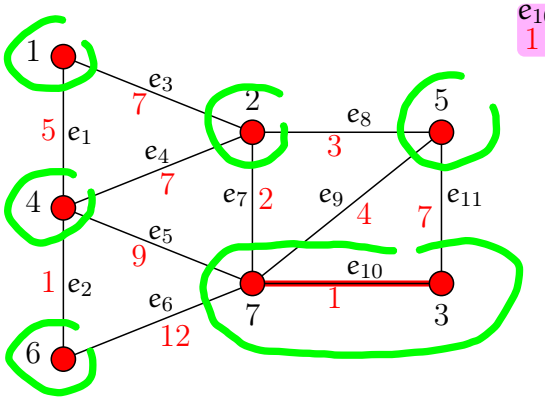
Ефикасна реализацията на алгоритъма на Kruskal се постига, като първо се сортират ребрата по тегло и после **в ненамаляващия ред** на теглата се слагат първите $n-1$ ребра, за всяко от които, **в този ред**, е вярно, че не образува цикъл с вече сложените ребра. Очевидно първите две ребра винаги "влизат", защото две ребра не могат да образуват цикъл, но от третото ребро трябва да се проверява дали образува цикъл с вече сложените ребра.

Ето пример за работата на алгоритъма на Kruskal върху графаот Фигура 4.16. В началото сортираме единадесетте ребра по тегло и правим покриваща гора, която е празният граф; тоест, има седем изолирани върхове.



e_{10}	e_2	e_7	e_8	e_9	e_1	e_3	e_4	e_{11}	e_5	e_6
1	1	2	3	4	5	7	7	7	9	12

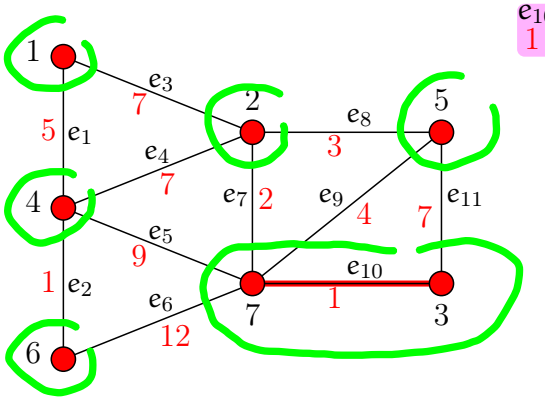
След това започваме да разглеждаме ребрата в сортираната редица, като всяко ребро или "влиза", или се прескача. Първо е e_{10}.



e_{10}	e_2	e_7	e_8	e_9	e_1	e_3	e_4	e_{11}	e_5	e_6
1	1	2	3	4	5	7	7	7	9	12

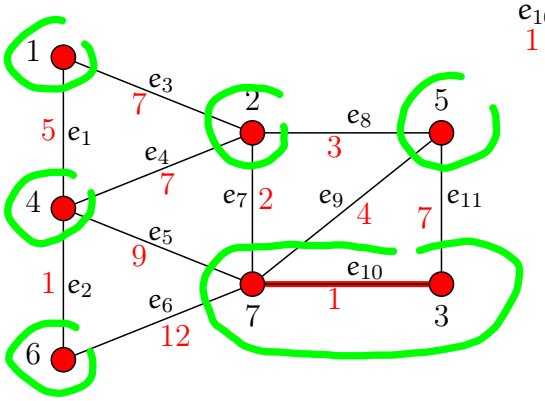
e_{10} влиза, защото краищата му, а именно 7 и 3, са в различни дървета на гората. Сливаме тези дървета плюс реброто e_{10} в едно единствено дърво. Така гората вече има само шест дървета.

†Изборът на T' е произволен. Със същия успех можеше да вземем среза $\{V(\mathsf{T}'), V \backslash V(\mathsf{T}')\}$.



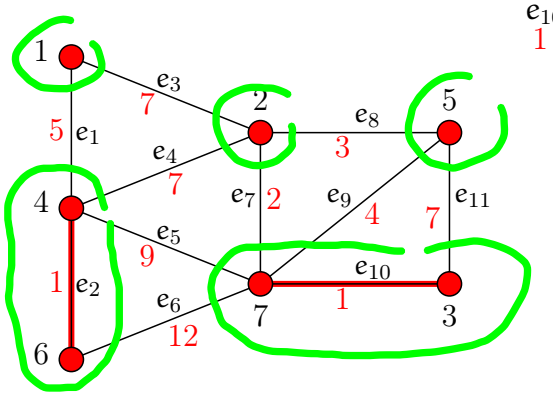
e_{10}	e_2	e_7	e_8	e_9	e_1	e_3	e_4	e_{11}	e_5	e_6
1	1	2	3	4	5	7	7	7	9	12

После е e_2.



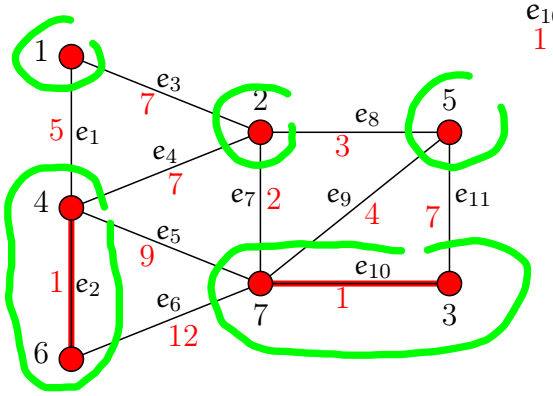
e_{10}	e_2	e_7	e_8	e_9	e_1	e_3	e_4	e_{11}	e_5	e_6
1	1	2	3	4	5	7	7	7	9	12

e_2 влиза. Краищата му, а именно 4 и 6, са в различни дървета, които сливаме заедно с него и получаваме гора със само пет дървета.



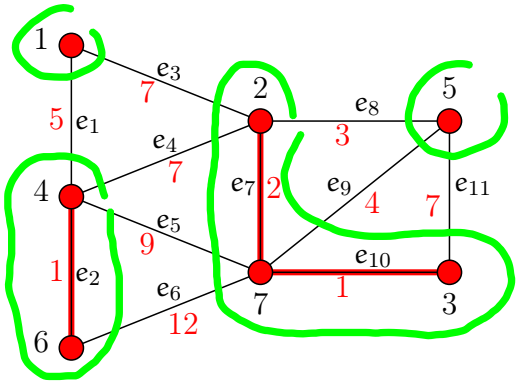
e_{10}	e_2	e_7	e_8	e_9	e_1	e_3	e_4	e_{11}	e_5	e_6
1	1	2	3	4	5	7	7	7	9	12

После е e_7.



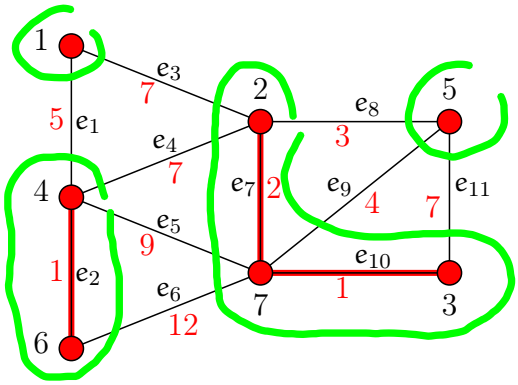
e_{10}	e_2	e_7	e_8	e_9	e_1	e_3	e_4	e_{11}	e_5	e_6
1	1	2	3	4	5	7	7	7	9	12

e_7 влиза, защото краищата му са в различни дървета. Сливаме ги заедно с него и получаваме гора със само четири дървета.



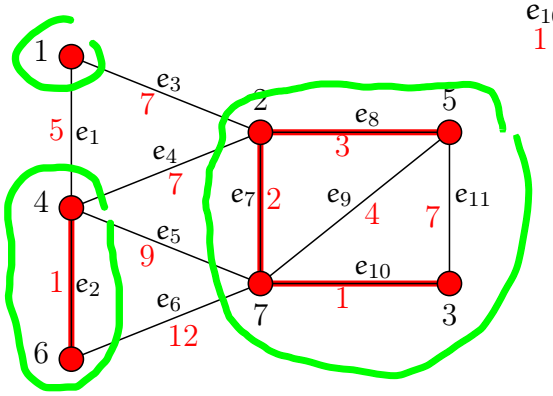
e_{10}	e_2	e_7	e_8	e_9	e_1	e_3	e_4	e_{11}	e_5	e_6
1	1	2	3	4	5	7	7	7	9	12

После е e_8.



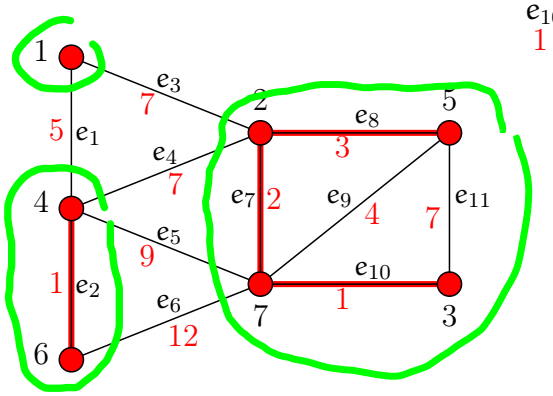
e_{10}	e_2	e_7	e_8	e_9	e_1	e_3	e_4	e_{11}	e_5	e_6
1	1	2	3	4	5	7	7	7	9	12

e_8 влиза, защото краищата му са в различни дървета. Сливаме ги заедно с него и получаваме гора със само три дървета.



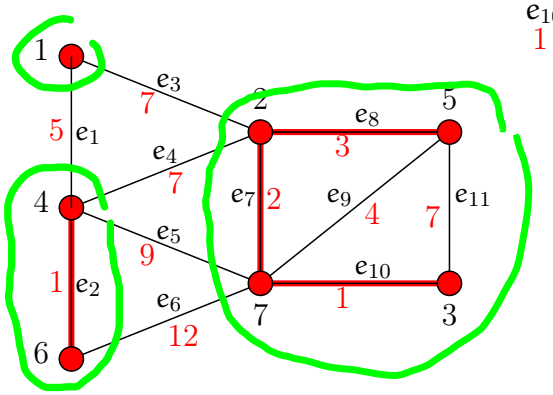
e_{10}	e_2	e_7	e_8	e_9	e_1	e_3	e_4	e_{11}	e_5	e_6
1	1	2	3	4	5	7	7	7	9	12

После е e_9.



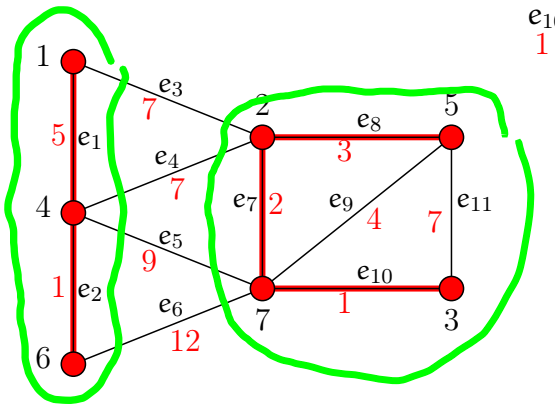
e_{10}	e_2	e_7	e_8	e_9	e_1	e_3	e_4	e_{11}	e_5	e_6
1	1	2	3	4	5	7	7	7	9	12

Неговите краища са в едно и също дърво, поради което e_9 се прескача. После е e_1.



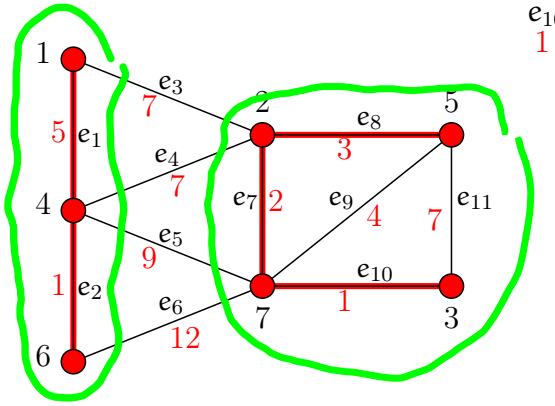
e_{10}	e_2	e_7	e_8	e_9	e_1	e_3	e_4	e_{11}	e_5	e_6
1	1	2	3	4	5	7	7	7	9	12

e_1 влиза, защото краищата му са в различни дървета. Сливаме ги заедно с него и получаваме гора със само две дървета.



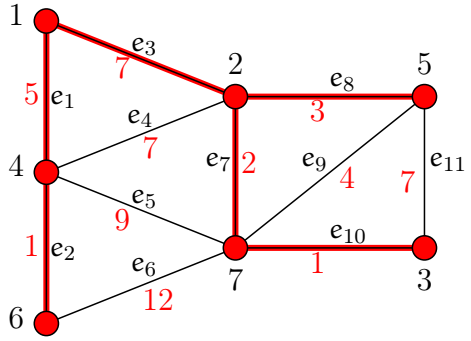
e_{10}	e_2	e_7	e_8	e_9	e_1	e_3	e_4	e_{11}	e_5	e_6
1	1	2	3	4	5	7	7	7	9	12

После е e_3.



e_{10}	e_2	e_7	e_8	e_9	e_1	e_3	e_4	e_{11}	e_5	e_6
1	1	2	3	4	5	7	7	7	9	12

e_3 влиза, защото краищата му са в различни дървета. Сливаме ги заедно с него и получаваме гора със само едно дърво, която е и покриващо дърво за графа.



e_{10}	e_2	e_7	e_8	e_9	e_1	e_3	e_4	e_{11}	e_5	e_6
1	1	2	3	4	5	7	7	7	9	12

Няма смисъл да продължаваме да разглеждаме ребра в сортираната редица: знаем, че графът има седем върха, знаем, че шест ребра вече са влезли, откъдето следва, че покриващата гора вече е (покриващо) дърво. Ако разглеждаме сортираната редица от ребра до края, ще се натъкваме само на ребра (e_4, e_{11}, e_5 и e_6, в този ред), чиито краища са в едно и също дърво, и ще ги прескачаме. Така че това е краят на алгоритъма. Построихме МПД.

4.4 Най-къси пътища в графи

4.4.1 Фундамент

Разглеждаме по подразбиране ориентирани тегловни графи. Ако в даден момент разглеждаме неориентирани графи, това ще се каже изрично. Наличието на примки не променя с нищо най-късите пътища – ако теглото на примката е положително, никой най-къс път няма да мине през нея, ако е отрицателно, то тя е отрицателен цикъл (вижте Определение 103), който обезсмисля задачата за намиране на най-къси пътища съгласно Наблюдение 32. Също така, наличието на снопове от паралелни ребра не променя нищо – ако има сноп с повече от едно ребро, то всеки най-къс път, ако изобщо ползва ребро от този сноп, ще ползва само най-лекото ребро. Така че разглеждаме обикновени ориентирани графи, без примки и без паралелни ребра. По отношение на свързността, графите са произволни; както ще видим, ако има върхове, недостижими от даден начален връх, това се индикира чрез специални стойности и в края на алгоритъма е ясно кои върхове не са достижими.

По подразбиране $G = (V, E)$ е ориентиран тегловен граф с тегловна функция $w : E \to \mathbb{R}$.

Конвенция 13

В Подсекция 4.4, казвайки "път", имаме предвид път, който не е непременно прост.

Нотация 6

Нека $u, v \in V$. Нотацията "$u \rightsquigarrow v$" е кратък запис за "ориентиран път от u до v". Нотацията "$u \overset{p}{\rightsquigarrow} v$" е кратък запис за "$p$ е ориентиран път от u до v".

Определение 102: Тегло на път

Теглото на пътя p е $w(p) = \sum_{e \in E(p)} w(e)$.

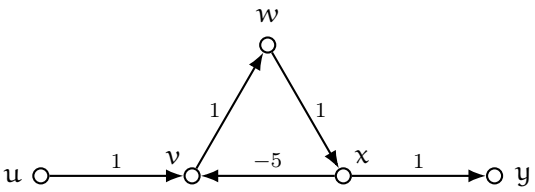
Определение 103: Отрицателен цикъл

Нека G е тегловен граф. Ако G е ориентиран, *отрицателен цикъл* е всеки прост цикъл в G, който има отрицателна сума от теглата на ребрата. Ако G е неориентиран, отрицателен цикъл е всеки прост цикъл в G, който има отрицателна сума от теглата на ребрата, както и всяко ребро с отрицателно тегло.

Наблюдение 32: Проблем при отрицателните цикли.

Ако теглата са само положителни, "най-къс път от u до v" е добре дефинирано понятие. Обаче за всеки отрицателен цикъл c, за всеки път p, който съдържа c, съществува път p', такъв че $w(p') < w(p)$, като p' съдържа c повече пъти от p. Ерго, ако поне един път от връх u до връх v съдържа поне един отрицателен цикъл, не можем да дефинираме "най-къс път от u до v".
Ако няма отрицателни цикли, този проблем не съществува дори при наличие на отрицателни тегла.

Като пример, да разгледаме следния граф:



Ако разглеждаме само прости пътища, най-късият път (той е и единствен) $u \rightsquigarrow y$ е пътят u, v, w, y с тегло 4. Ако обаче разглеждаме пътища, които не са непременно прости, най-къс път $u \rightsquigarrow y$ няма, защото всяко минаване през цикъла v, w, x, v добавя $1 + 1 - 5 = -3$ към теглото на пътя:

u, v, w, x, v, w, x, y има тегло 1
$u, v, w, x, v, w, x, v, w, x, y$ има тегло -2
$u, v, w, x, v, w, x, v, w, x, v, w, x, y$ има тегло -5
и така нататък

Наблюдение 33: Когато няма отрицателни цикли.

При липса на отрицателни цикли, за всеки два върха u и v, всеки най-къс път от u до v задължително е прост път.

Определение 104: Теглото на най-къс път

Теглото на най-къс път от u *до* v е

$$\delta(u,v) = \begin{cases} \min\{w(p) \mid u \overset{p}{\rightsquigarrow} v\}, & \text{ако има поне един } u \rightsquigarrow v \\ \infty, & \text{ако няма такъв път} \end{cases}$$

Забележете, че ако съществува поне един $u \rightsquigarrow v$, който съдържа отрицателен цикъл, то, съгласно Наблюдение 32, $\min\{w(p) \mid u \overset{p}{\rightsquigarrow} v\}$ става $-\infty$. Причината да дефинираме като ∞ теглото на най-къс път при липса на поне един път е очевидна. И така, $\delta(u,v) \in \mathbb{R} \cup \{-\infty, \infty\}$.

Има леко терминологично несъответствие: щом говорим за "тегло на път", би трябвало да казваме "най-лек път" за път с минимално телго, а не "най-къс път". Но както "тегло на път", така и "най-къс път" (в тегловния смисъл) вече са широко приети както на български, така и на английски, като съответно се казва "weight of path" и "shortest path weight", така че в тези лекции ще се съобразим с това. И така, ще се придържаме към следната езикова конвенция.

Конвенция 14

В контекста на задачата, която разглеждаме, "най-къс път" е синоним на "път с най-малко тегло". Ако имаме предвид броя на ребрата в пътя, ще го кажем експлицитно.

Конвенция 15

Тегловните функции, които ще разглеждаме, или имат кодомейн $\mathbb{R}^+$, или имат кодомейн $\mathbb{R}$, но в G няма отрицателни цикли. Поради това никъде няма да дефинираме пътищата като прости пътища. Те ще се окажат прости вследствие на отсъствието на отрицателни цикли, предвид Наблюдение 33.

4.4.2 Приложения

Задачата за най-къси пътища в графи има грамадни приложения. Ако няма тегла, задачата се решава ефикасно с BFS, но наличието на тегла прави задачата интересна и изисква подход, много по-изтънчен от този за решаването на BFS. Както се казва в [1, стр. 6]:

> The shortest path problem is perhaps the simplest of all network flow problems.

Ето само някои примерни приложения на задачата за най-къси пътища в тегловни графи, някои от които са общоизвестни, а други са взеки от [1], [55] и [16].

- В пътуванията от място А до място Б през междинни места, каквито пътувания хората са правили от незапомнени времена. Теглата са дължините на пътищата, свързващи директно двойки места.
- В географските информационни системи (GIS): същата задача, но сега се решава с компютър.
- В маршрутизацията в компютърните мрежи – искаме да намерим маршрут за комуникация между дадени агенти в дадена мрежа, по който маршрут закъснението е минимално. Или искаме да намерим максималното минимално закъснение при комуникация между кои да е два агента (това е задачата за диаметъра на графа, но в тегловен вариант). Теглата са времена.
- В транспортни задачи като намиране на най-евтин полет с прекачвания от дадено летище до дадено друго летище, като цените на директните полети са известни. Теглата са цени.
- В урбанистика при изследване на трафика – стандартно допускане е, че хората обикновено се придвижват по най-късия маршрут. Задачата за оптимизиране на трафика в града съдържа като подзадача задачата за намиране на най-къси пътища, в множество инстанциации за различни двойки локации в града.
- Всеки човек, и изобщо всяко достатъчно сложно живо същество, има вградена представа за отсечката като най-къс път от едно място А до друго място Б във физическия свят, при липса на препятствия между тези места. При наличие на препятствия движение по отсечка може да е невъзможно, но дори тогава хората имат вградена представа за най-ефикасния начин за придвижване от А до Б като придвижване по най-къс път. В този смисъл, движението по най-къс път е най-естественото движение, ако целта е максимум ефикасност. Оттук и приложението на задачата за най-късите пътища в областта на *планиране на движението* (*motion planning*): намирането на редица от пространствени конфигурации, която "придвижва" даден обект от А до Б. Планиране на движението има в компютърната геометрия, компютърната анимация, роботиката и компютърните игри. Във всяка от тези области, алгоритмите за най-къси пътища имат приложение.
- В производственото планиране, например т. нар. *inventory planning* [1, стр. 749] или *dynamic facility location* [1, стр. 764].
- При оптимизиране на производството, примерно на стоманени греди [1, стр. 11].

- В типографията – примерно, типографската система TEX използва задачата за най-къси пътища като подзадача при изчисляването на оптималното (по критерии, които са фиксирани от създателя на TEX Donald Knuth) разбиване на текста на дълъг параграф на отделни редове [1, стр. 21].
- При апроксимиране на функция, която е линейна по части [1, стр. 98]. Търси се друга функция, също линейна по части, но с по-малко *breakpoints*, като се иска приемлив компромис между размера на представянето и точността на апроксимацията.
- В оптимизационни задачи като задачата за раницата (*Knapsack Problem*) [1, стр. 100]. Задачата може да се формулира като задача за най-дълъг път и оттам да се трансформира в задача за най-къс път. Както се казва в [1, стр. 100]:

  > This application illustrates an intimate connection between dynamic programming and shortest path problems on acyclic networks. By making the appropriate identification between the stages and "states" of any dynamic program and the nodes of a network, we can formulate essentially all deterministic dynamic programming problems as equivalent shortest path problems.

  И още [1, стр. 102]:

  > "(deterministic) dynamic programming is a special case of the shortest path problem."

- Системи от диференчни ограничения (*Systems of difference constraints*) [1, стр. 103]. Става дума за линейно оптимизиране, при което ограниченията са от определен вид, на тях съответства някакъв ориентиран граф, и съвкупността от ограниченията е удовлетворима тстк в съответния граф няма отрицателен цикъл. Задачата за установяване или отхвърляне на съществуване на отрицателен цикъл се решава с алгоритъм за най-къси пътища.
- Намиране на транзитивно затваряне на релация [55, стр. 212].
- Разпознаване на естествена реч или автоматизирано намиране на правописни грешки [55, стр. 212].
- Арбитраж (*Arbitrage*) при търговия с валути [16, стр. 679].

4.4.3 Разновидности на задачата

Задачата за най-къс път е известна в няколко варианта. По отношение на началото и края на най-къс път вариантите на задачата са следните.

1. При дадени върхове $s, t \in V$, да се намери най-къс път $s \rightsquigarrow t$.
2. При даден връх s, да се намери най-къс път $s \rightsquigarrow u$ за всеки $u \in V$.
3. При даден връх t, да се намери най-къс път $u \rightsquigarrow t$ за всеки $u \in V$.
4. Да се намери най-къс път $u \rightsquigarrow v$, за всеки $u, v \in V$.

Първият вариант на задачата, а именно от даден връх s до даден връх t, изглежда алгоритмично най-лесен. Както ще се убедим обаче, в най-лошия случай, за да изчислим теглото на най-лек път $s \rightsquigarrow t$, се налага да изчислим и теглото на най-лек път $s \rightsquigarrow u$, за всеки $u \in V$. Ерго, в най-лошия случай, първият вариант е труден колкото втория.

Ние ще разгледаме подробно втория вариант на задачата.

Третият вариант има алгоритмичната трудност на втория, защото третият вариант се превръща във втория при обръщане на посоките на ребрата; тоест, на транспониране на графа.

Четвъртият вариант ще бъде разгледан в курса по алгоритми.

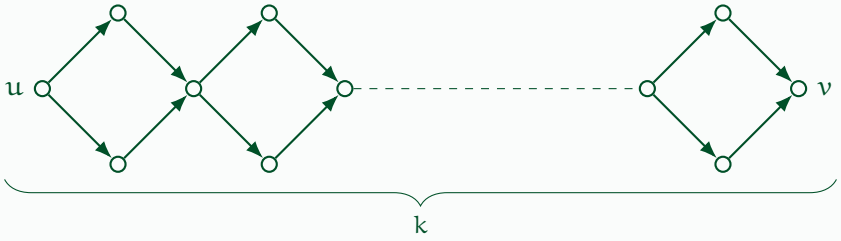
Ще разгледаме и друга класификация на задачата за най-къси пътища по това, какво искаме за най-къс път $u \rightsquigarrow v$.

- Може да искаме само дължината, а не самия път. Това не е много полезно на практика.
- Може да искаме дължината плюс самия път, което в реални приложения е много по-полезно и информативно.
- Може да искаме броя на най-късите пътища $u \rightsquigarrow v$.
- Може да искаме множеството от всички най-къси пътища $u \rightsquigarrow v$.

Ние ще разглеждаме алгоритми, които връщат дължината на най-къс път плюс самия път.

Допълнение 26: Генерирането на всички най-къси пътища е неефикасно

В реални приложения може да е много полезно да бъдат получени всички най-къси пътища от връх до друг връх. Имайки всички най-къси пътища, може да изберем от тях път по някакъв друг критерий. За съжаление, в най-лошия случай, броят на най-късите пътища $u \rightsquigarrow v$ може да е експоненциален в n, в което може да се убедим с този прост пример (приемете, че теглата са единици):



Виждаме k на брой подграфи, да ги наречем *ромбовете*, като всеки ромб е  . Ромбовете са "слепени" в редица. Очевидно на фигурата има 2^k пътя от u до v, всеки от тях с дължина $2k$, защото за всеки ромб, може да минем "отгоре" или "отдолу". И тъй като е възможно $k \asymp n$, в графа може да има $\Omega(2^n)$ пътя от u до v. Следователно, няма ефикасен алгоритъм, който строи всички пътища, защото само записването им отнема време $\Omega(2^n)$ в най-лошия случай.

4.4.4 Същностни характеристики на най-късите пътища

Теорема 53: Най-къс път се състои от най-къси подпътища.

Нека G е ориентиран тегловен граф, s и t са върхове в него и p е най-къс път от s до t. Нека $u, v \in V(p)$, като u предхожда v от s към t в p, ако u и v са различни[a]. Нека подпътят на p от u до v се казва q:

$$p = \underbrace{s \rightsquigarrow u \overset{q}{\rightsquigarrow} v \rightsquigarrow t}_{p}$$

Тогава q е най-къс път от u до v.

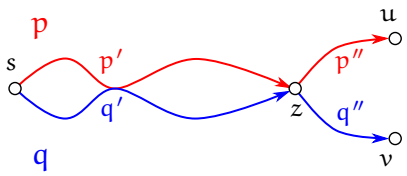
[a] Не е необходимо s, t, u и v да са четири различни върха; в най-екстремния вариант, може $s = t = u = v$ и пак остава вярно, че най-къс път се състои от най-къси подпътища.

Доказателство: Ако допуснем, че има път q', такъв че $u \overset{q}{\rightsquigarrow} v$ и $w(q') < w(q)$, веднага заключаваме, че p не е най-къс път от s до t, понеже замяната на q с q' в p намалява $w(p)$ с $w(q) - w(q')$. □

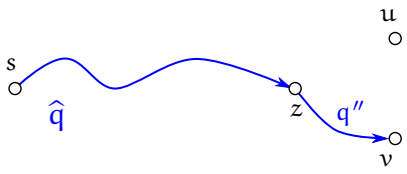
В доказателството на Теорема 53 се възползваме от това, че пътищата не са непременно прости. Ако настоявахме пътищата да са прости, то вмъкването на q' на мястото на q може да е проблематично, понеже няма гаранции, че q' и p нямат други общи върхове освен u и v; ерго, след вмъкването целият път може да не е прост. Ако теглата са положителни, можем "да изрежем" общите части на p и q' и да получим път p'', който е с дори още по-малко тегло, което пак ни дава желаното противоречие. Обаче при отрицателни тегла и по-специално при наличието на отрицателни цикли, това "изрязване" може да увеличи теглото на пътя, което е проблем за доказателството.

Дърво на най-късите пътища. Ако искаме да изчислим най-къс път $s \rightsquigarrow t$, паметта, която е необходима за записването на пътя, има около n клетки в най-лошия случай, защото, в най-лошия случай, дължината на пътя е $n-1$ и не можем да избегнем записването на всеки връх от него. На пръв поглед, ако искаме да запишем по един най-къс път $s \rightsquigarrow v$ за **всеки** $v \in V$, ще ни трябват около n^2 клетки памет.

Всъщност, можем "да минем" само с около n клетки памет за всички пътища, защото можем да ги представим с ориентирано кореново дърво-арборесценция, която може да се представи с масив на предшествията $\pi[1, \ldots, n]$, също както при BFS и DFS. Ще покажем, че най-къси пътища от s могат да се представят с една арборесценция с корен s. Нека p и q са най-къси пътища от s съответно до u и v, където u и v са различни върхове. Ако единственият общ връх на p и q е s, няма какво да се показва. Нека p и q имат поне един общ връх освен s. Нека z е най-отдалеченият в p и в q връх от s, който е общ за p и q. Очевидно не може $z = u = v$, понеже u и v са различни. Ако $z = u \neq v$ или $z = v \neq u$, няма какво повече да показваме; това е случаите, в който съответно p е част от q или q е част от p. Остава да разгледаме случая, в който $z \neq u$ и $z \neq v$. Нека подпътят на p от s до z е p', а подпътят на q от s до z е q':

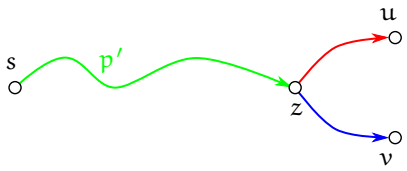


Твърдим, че $w(p') = w(q')$. Да допуснем противното. БОО, нека $w(p') < w(q')$. Тогава в q подменяме q' с p':



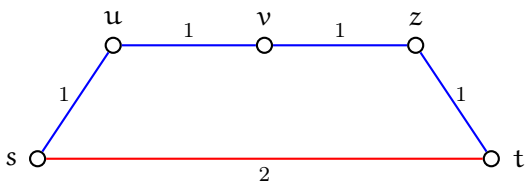
и получаваме път $\hat{q}$ от s до v, такъв че $w(\hat{q}) < w(q)$, в противоречие с допускането, че q е най-къс път от s до v.

Щом $w(p') = w(q')$, може и в p, и в q да използваме само единият от тях, да кажем p'; по този начин, получаваме най-къси пътища от s до u и от s до v, които се състоят от един общ подпът от s до z, след което се "разделят" и повече не се събират:



Ако си направим същото нещо за всеки два върха, достижими от s, ще получим арборесценция с корен s, което е дървото на най-късите пътища от s.

Задачата за най-късите пътища и задачата за МПД са различни. Естествено, те са задачи върху различни видове графи, като МПД е върху неориентирани тегловни графи, а най-късите пътища са върху ориентирани тегловни графи, но съществената разлика не е в това. Дори да решаваме задачата за най-късите пътища върху неориентирани тегловни графи, тя остава принципно различна от задачата за МПД. Ето малък пример:



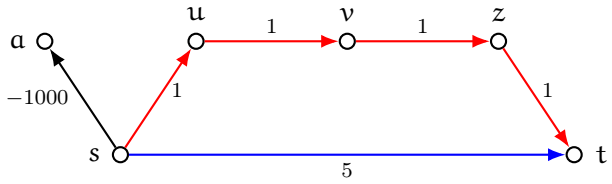
Най-късият път между s и t е реброто (s, t) с тегло 2 (в червено). Но МПД-то (то е само едно) се състои от четирите сини ребра, всяко с тегло 1. Ако си представим работата на алгоритъма на Kruskal върху този граф, сортирайки ребрата, той ще постави в началото четирите ребра с тегло 1 и след това реброто с тегло 2. Когато започне да слага ребра, той ще сложи четирите ребра с тегло 1 и ще спре, без да достигне до реброто с тегло 2, което реализира най-късия път.

Наблюдение 34

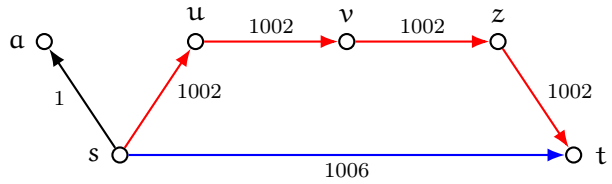
Задачите за намиране на МПД и намиране на дърво на най-късите пътища върху неориентирани тегловни графи са принципно различни. Всяко МПД, за всеки два върха s и t, дава (уникален) път p между s и t, но в общия случай p не е най-къс път в графа между s и t.

При най-късите пътища не може да се "отървем" от отрицателните тегла. Друга важна разлика между задачите за намиране на МПД и на най-къси пътища е възможността да се "отървем" от отрицателни тегла. Когато дефинирахме задачата за МПД, дефинирахме теглата като реални числа, оставяйки възможността те да са отрицателни. Наистина, нищо не пречи теглата да са отрицателни в Теорема 52 или в алгоритмите на Prim и Kruskal. Ако обаче по някаква причина не искаме отрицателни тегла там, лесно може да се отървем от тях, като намерим най-малкото отрицателно тегло x и после добавим $|x| + 1$ към теглата на всички ребра. Те ще станат положителни, а МПД-тата на графа ще останат **същите**.

Този трик не работи при най-късите пътища. Ако "повдигнем" теглата достатъчно, така че всички те да станат положителни, дървото на най-късите пътища може да се промени. Ето малък пример за това, този път с ориентиран граф:



Очевидно най-късият път от s до t се състои от червените ребра и има тегло 4. Ако обаче добавим 1001 към теглото на всяко ребро, така че теглата да станат само положителни:



пътят от червените ребра става с тегло 4008, а този със синьото ребро, само 1006, така че сега той е най-късият път от s до t.

Наблюдение 35

Нека G тегловен неориентиран граф с тегловна функция $w : E \to \mathbb{R}$. Нека $w' : E \to \mathbb{R}$ е друга тегловна функция, като $\forall e \in E : w'(e) = w(e) + c$, където c е произволна реална константа. МПД-тата на G по отношение на w и по отношение на w' са едни и същи. Ако обаче G тегловен ориентиран граф с тегловна функция $w : E \to \mathbb{R}$ и $w' : E \to \mathbb{R}$ е друга тегловна функция, като $\forall e \in E : w'(e) = w(e) + c$, където c е произволна реална константа, дървото на най-късите пътища по отношение на произволен $s \in V(G)$ може да бъде различно за w и за w'.

Следствие от това е фактът, че задачата за намиране на минимално покриващо дърво е алгоритмично същата като задачата за намиране на максимално покриващо дърво:

- ако в Алгоритъм 8 заменим E'_{min} с E'_{max}—множеството от ребрата с **максимално** тегло в E'—ще получим алгоритъм, който намира максимално покриващо дърво;
- ако в Алгоритъм 9 заменим E'_{min} с E'_{max}—множеството от ребрата с **максимално** тегло в E'—също ще получим алгоритъм, който намира максимално покриващо дърво.

За да де убедим в това, достатъчно е да разгледаме Теорема 52. В нея да заменим $E^{min}_{u,w}$ с $E^{max}_{u,w}$: подмножеството на $E_{u,w}$ от ребрата с **максимално** тегло. Получаваме твърдението "за всяко $e \in E^{max}_{u,w}$ съществува максимално покриващо дърво D, което съдържа e", което е вярно и се доказва напълно аналогично на Теорема 52.

От друга страна, никой алгоритъм за най-къси пътища върху общи графи[†] не става алгоритъм за най-дълги пътища при замяна на min с max. Причината е обяснена в Допълнение 27: при най-дългите пътища не е вярно, че оптималната структура се състои от оптимални подструктури.

4.4.5 Пак за отрицателните тегла

Както се убедихме, отрицателните тегла представляват значителен проблем за задачата за най-късите пътища (за разлика от задачата за МПД, където те нямат никакво значение). При наличие на отрицателни тегла ние може дори да не сме в състояние да дефинираме "най-къс път от s до t", ако поне един път от s до t съдържа отрицателен цикъл.

Това се дължи на факта, че разглеждаме пътища, които не са непременно прости, което влече, че всяко "завъртане" през отрицателен цикъл дава още по-къс път. Не може ли да постулираме, че разглеждаме само прости пътища, и по този начин да няма възможност да се "въртим" в циклите поначало? Отговорът е, че теоретично можем да го направим, но алгоритмично не можем да го имплементираме по ефикасен начин. Алгоритмите за най-къси пътища, примерно този на Dijkstra, не правят проверка дали изградените пътища са прости или не. Те се оказват прости (вижте Наблюдение 33; в алгоритъма на Dijkstra се иска теглата да са положителни).

Силна улика за това, че не може да решим проблема с отрицателните цикли с елементарни средства—примерно, постулиране, че не разглеждаме други пътища освен прости—е фактът, че задачата за най-къси пътища е същата като задачата за най-дълги пътища при обръщане на знака на теглата. Иначе казано, ако G е ориентиран тегловен граф, върху който са дефинирани тегловни функции $w, w' : E \to \mathbb{R}$, като $\forall e \in E(G) : w(e) = -w'(e)$, то, за всеки $s, t \in V(G)$, p е най-къс път $s \rightsquigarrow t$ по отношение на w тстк p е най-дълъг път $s \rightsquigarrow t$ по

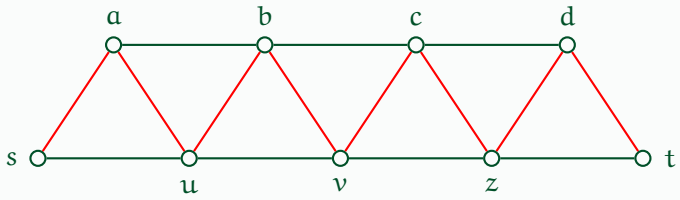
[†]Има алгоиртми за най-къси пътища върху **ограничени класове графи**, примерно дагове, които алгоритми стават алгоритми за най-дълги пътища при замяна на min с max.

отношжение на w'. Задачата за най-дълги пътища е **NP**-пълна, което означава, че почти сигурно за нея няма ефикасен алгоритъм.

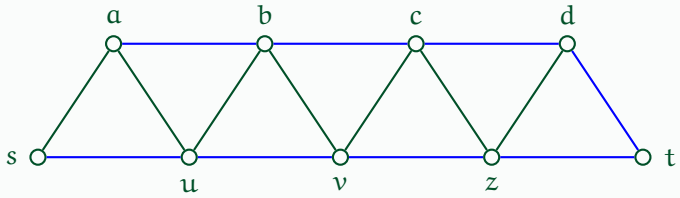
Допълнение 27: За структурата на най-дългите пътища

За задачата за най-дългите пътища не е вярно, че оптималната структура се състои от оптимални подструктури (сравнете с Теорема 53). Това обяснява драстичната разлика между ефикасността на най-добрите известни алгоритми за най-къси пътища и за най-дълги пътища.

Преди всичко, за да дефинираме смислено задачата за най-дългите пътища, трябва да постулираме, че разглеждаме само прости пътища. Иначе, дори в не-тегловния вариант, задачата е лошо дефинирана, понеже съществуването на цикъл в ориентиран граф или на ребро в неориентиран граф влече съществуването на неограничено дълги, като брой ребра, пътища, които не са прости. И така, ако дефинираме задачата за най-дълги пътища по очевидния начин, настоявайки пътищата да са прости, е лесно да намерим пример, в който най-дългият път не се състои от най-дълги подпътища (допускаме, че теглата са единици):



Най-дългият път между s и t е с дължина 8 и е нарисуван с червено. В него има подпът между s и a с дължина 1. Но най-дългият път между s и a е с дължина 8, както се вижда на следната фигура (в синьо):



Очевидно не е вярно, че най-дългите пътища се състоят от най-дълги подпътища.

След всички изложени проблеми, които възникват при отрицателни тегла на ребрата, възниква въпроса, а защо не се откажем от отрицателни тегла поначало? Наистина, ако теглата моделират някакви физически величини, които по природа да положителни, примерно закъснения във времето или разстояния в реалния свят, няма как да се появят отрицателни тегла; отрицателно закъснение, например, означава пътуване назад във времето. Обаче има важни практически задачи, в които е удобно да се работи с абстракцията на отрицателните числа. Примерно, ако моделираме някакви потоци от стоки или пари, можем да кажем, че от сметката на X са прехвърлени 100 лева в сметката на Y, като кажем, че от сметката на Y са прехвърлени -100 лева в сметката на X.

Наблюдение 36

Ако се откажем от отрицателни тегла изобщо, графите ни ще станат със строго по-малка моделираща мощ.

Вярно е, че алгоритъмът на Dijkstra, който е единственият алгоритъм за най-къси пътища, който ще разгледаме в този курс, работи само при положителни тегла. Но има други алгоритми за най-къси пътища, не толкова ефикасни колкото алгоритъма на Dijkstra, които работят при произволни тегла, стига да няма отрицателни цикли. Такъв алгоритъм ще разгледаме в курса по алгоритми.

4.4.6 Алгоритъм на Dijkstra

Ако теглата на ребрата са неотрицателни, можем да използваме следния прост и елегантен алгоритъм за решаване на задачата за най-късите пътища във варианта от един връх до всички останали, като генерираме и самите пътища като дърво на най-късите пътища. Алгоритъмът е открит от великия компютърен учен Edsger Wybe Dijkstra [19]. Този човек е открил много алгоритми, всеки от които може да бъде наречен "Алгоритъм на Dijkstra", така че за избягване на двусмислие може да се казва "Алгоритъм на Dijkstra за най-късите пътища". В този курс не разглеждаме други алгоритми на Dijkstra, така че краткото "Алгоритъм на Dijkstra" е прецизно наименование.

Въпреки че алгоритъмът е прост за възприемане и програмиране, доказателството на коректността му съвсем не е просто и ще го отложим за курса по алгоритми. Тук само ще видим кода, ще направим малък коментар върху него и ще го симулираме върху малък граф.

Алгоритъм 10: Алгоритъм на Dijkstra

Вход: ориентиран граф $G = (V, E)$ с тегловна функция $w : E \to \mathbb{R}^+$, връх s от V.
Изход: масив от дължините на най-късите пътища d, масив π, реализиращ кореново дърво на най-късите пътища с корен s.

❶ За всеки връх $v \in V$: $d[v] \leftarrow \infty$, $\pi[v] \leftarrow \text{Nil}$

❷ $d[s] \leftarrow 0$

❸ $S \leftarrow \varnothing$

❹ Ако във $V \backslash S$ няма връх u, такъв че $d[u] < \infty$, върни d и π и край.

❺ В противен случай, избери $x \in V \backslash S$, такъв че $d[x]$ е минимално

❻ $S \leftarrow S \cup \{x\}$

❼ За всеки $y \in \text{adj}[x]$:

① Ако $d[y] > d[x] + w(x, y)$, то
② $d[y] \leftarrow d[x] + w(x, y)$, $\pi[y] \leftarrow x$

❽ отиди на ❹

Масивите d и π по време на работата на алгоритъма имат следния смисъл. За всеки $v \in V$:

- $d[v]$ е **най-точната апроксимация** на дължината на най-къс път $s \overset{p}{\rightsquigarrow} v$, която можем да направим **въз основа на информацията до момента**,
- а $\pi[v]$ е родителят на v в p.

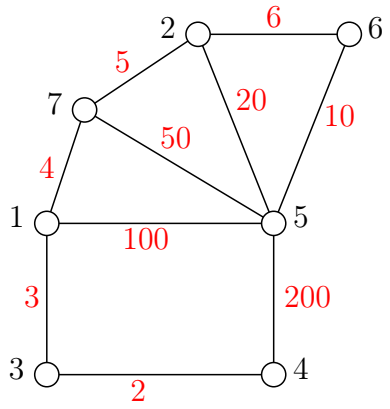
Както лесно се вижда от кода, това апроксимиране е само отгоре и никога отдолу; тоест, $d[v]$ не може да нараства, а само намалява. Вярно е, че в края на алгоритъма, d съдържа точните стойности, а не апроксимации, но това трябва да се докаже прецизно, което доказателство ще отложим за курса по алгоритми.

❶ и ❷ са инициализацията на алгоритъма. Наистина, $d[v]$ има смисъл на най-добра апроксимация на дължината на най-къс $s \rightsquigarrow v$ въз основа на това, което е известно до момента: ако $v = s$, очевидно тази дължина е нула, а ако $v \neq s$, не можем да кажем нищо за нея, което е същото като да кажем, че тя (най-добрата апроксимация) е безкрайност.

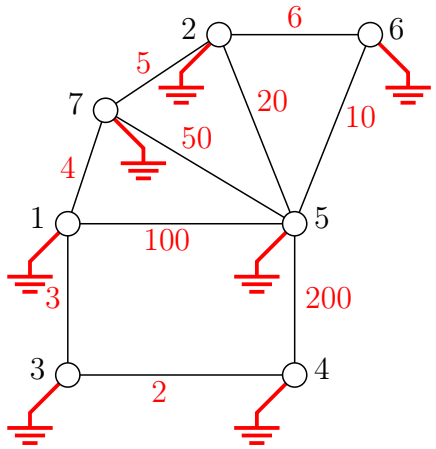
S е множеството от върховете, за които знаем точната стойност; с други думи, в него са всички върхове v, такива че $d[v]$ е точно дължината на най-къс път $s \rightsquigarrow v$.

Човек би казал, че трябва да инициализираме S с $\{s\}$, а не с празното множество, както правим в ❸. Това е напълно възможно, но в такъв случай трябва да променим и d и π-стойностите на децата на s като част от инициализацията. За да бъде кодът прост и елегантен, инициализираме S именно с празното множество. При първата итерация на **while**-а, променливата x задължително ще съдържа s, така че при въпросната първа итерация всяко дете на s ще "влезе" в y и ще получи правилни d и π-стойности.

Да разгледаме следния тегловен граф. За простота, графът е неориентиран и свързан. Теглата са написани с червено до ребрата.



Нека $s = 3$. След ❶, ❷ и ❸ имаме

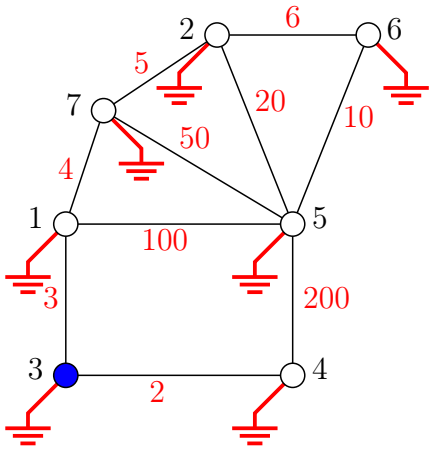


$S = \varnothing$

d:

1	2	3	4	5	6	7
∞	∞	0	∞	∞	∞	∞

Във V\S има връх, а именно 3, чиято d-стойност не е безкрайност, така че изпълнението отива на ❻. S става {3}. На рисунката на графа ще отбелязваме със син цвят на върховете от S:



$S = \{3\}$

d:

∞	∞	0	∞	∞	∞	∞
1	2	3	4	5	6	7

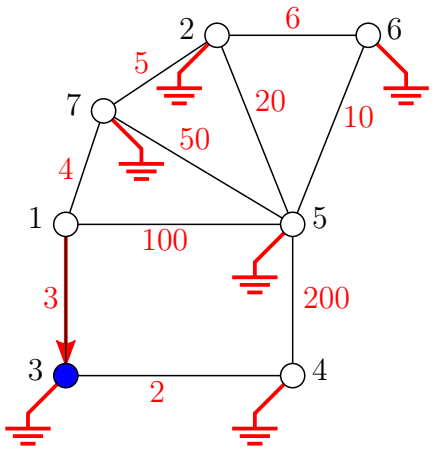
После във **for**-цикъла, y взема последователно стойностите на съседите на 3. Списъците на съседство не са дадени, така че да кажем, че първо y става 1. Булевото условие

$$d[y] > d[x] + w(x, y)$$

всъщност е

$$\infty > 0 + 3$$

което е истина, така че изпълнението отива на ②. Там d[1] става 3, а π[1] става 3.



$S = \{3\}$

d:

3	∞	0	∞	∞	∞	∞
1	2	3	4	5	6	7

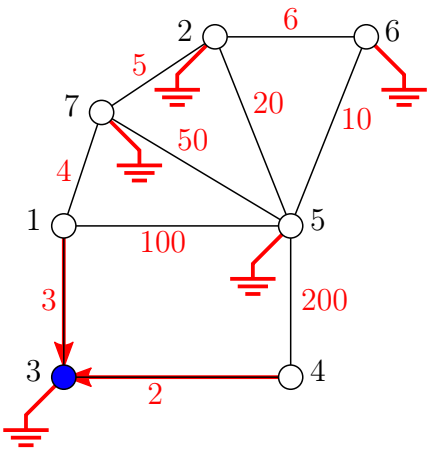
for-цикълът продължава да се изпълнява. y става 4. Булевото условие

$$d[y] > d[x] + w(x, y)$$

всъщност е

$$\infty > 0 + 2$$

което е истина, така че изпълнението отива на ②. Там d[4] става 2, а π[4] става 3.

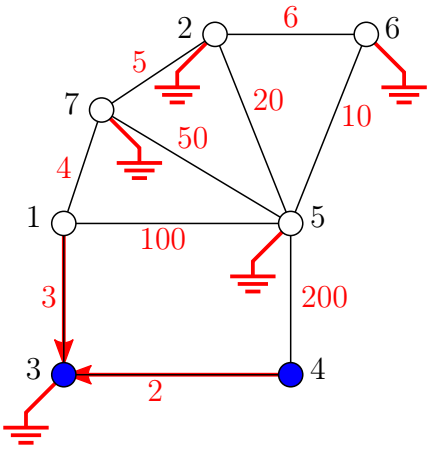


$S = \{3\}$

d:

3	∞	0	2	∞	∞	∞
1	2	3	4	5	6	7

for-цикълът престава да се изпълнява и отново сме на ❹. Има два върха извън S, чиято d-стойност не е безкрайност, така че отиваме на ❺, където x става 4. S става $\{3, 4\}$:



$S = \{3, 4\}$

d:

3	∞	0	2	∞	∞	∞
1	2	3	4	5	6	7

После y става 3 и 5, да кажем в този ред. Когато y е 3, булевото условие

$$d[y] > d[x] + w(x, y)$$

всъщност е

$$0 > 2 + 2$$

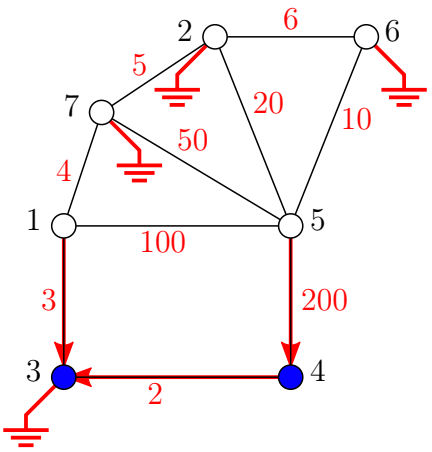
което е лъжа и ② не се изпълнява. После y става 5 и булевото условие

$$d[y] > d[x] + w(x, y)$$

всъщност е

$$\infty > 2 + 200$$

което е истина, така че изпълнението отива на ②. Там $d[5]$ става 202, а $\pi[5]$ става 4:

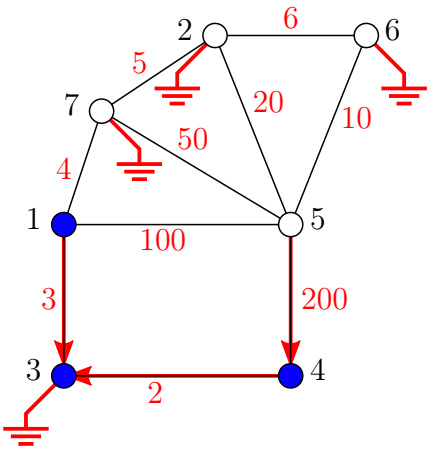


$S = \{3, 4\}$

d:

1	2	3	4	5	6	7
3	∞	0	2	202	∞	∞

for-цикълът престава да се изпълнява и отново сме на ❹. Има два върха извън S, чиято d-стойност не е безкрайност, така че отиваме на ❺, където x става 1. S става $\{1, 3, 4\}$:



$S = \{1, 3, 4\}$

d:

1	2	3	4	5	6	7
3	∞	0	2	202	∞	∞

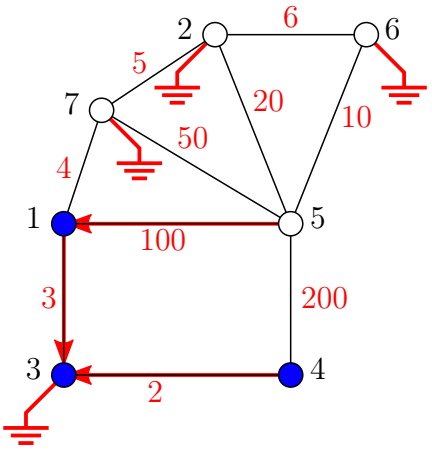
Връх 1 има три съседа: 3, 5 и 7. Да кажем, че първо y става 3. Това е опит за "връщане назад", но условието на ① е лъжа и ② не се изпълнява. Да кажем, че после y става 5. Булевото условие

$$d[y] > d[x] + w(x, y)$$

всъщност е

$$202 > 3 + 100$$

което е истина, така че изпълнението отива на ②. Там $d[5]$ става 103, а $\pi[5]$ става 1:



$S = \{1, 3, 4\}$

d:

1	2	3	4	5	6	7
3	∞	0	2	103	∞	∞

Както виждаме, тази апроксимация на дължината на най-къс път $3 \rightsquigarrow 5$ е по-добра (тоест, по-малка) от предишната, но все още не е истинската стойност.

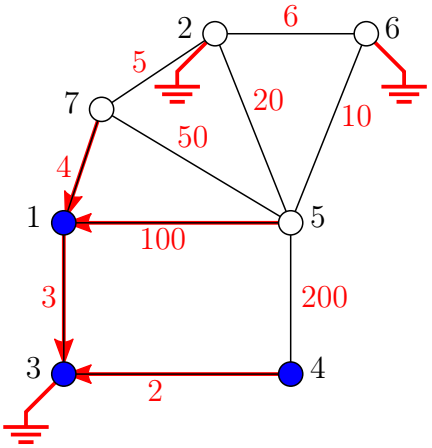
После y става 7. Булевото условие

$$d[y] > d[x] + w(x, y)$$

всъщност е

$$\infty > 3 + 4$$

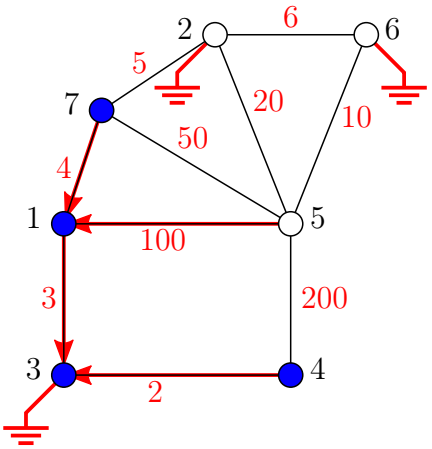
което е истина, така че изпълнението отива на ②. Там $d[7]$ става 7, а $\pi[7]$ става 1:



$S = \{1, 3, 4\}$

d:	3	∞	0	2	103	∞	7
	1	2	3	4	5	6	7

for-цикълът престава да се изпълнява и отново сме на ❹. Има два върха извън S, чиято d-стойност не е безкрайност, така че отиваме на ❺, където x става 7. S става $\{1, 3, 4, 7\}$:



$S = \{1, 3, 4, 7\}$

d:	3	∞	0	2	103	∞	7
	1	2	3	4	5	6	7

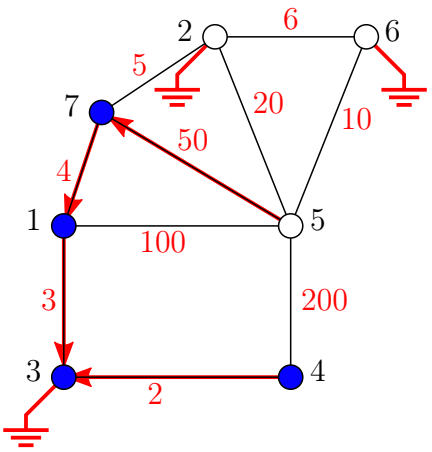
Връх 7 има три съседа: 1, 2 и 5. Да кажем, че първо y става 1. Това е опит за "връщане назад", но условието на ① е лъжа и ② не се изпълнява. Да кажем, че после y става 5. Булевото условие

$$d[y] > d[x] + w(x, y)$$

всъщност е

$$103 > 7 + 50$$

което е истина, така че изпълнението отива на ②. Там $d[5]$ става 57, а $\pi[5]$ става 7:



$S = \{1, 3, 4, 7\}$

d:

1	2	3	4	5	6	7
3	∞	0	2	57	∞	7

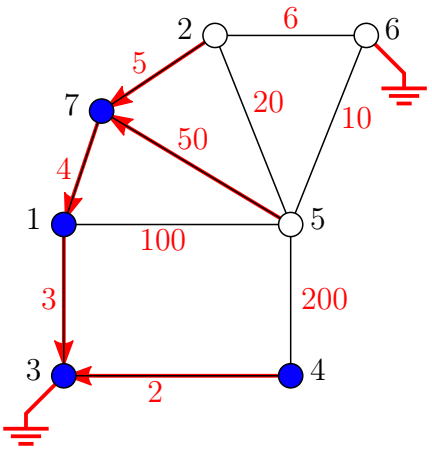
После y става 2. Булевото условие

$$d[y] > d[x] + w(x, y)$$

всъщност е

$$\infty > 7 + 5$$

което е истина, така че изпълнението отива на ②. Там $d[2]$ става 12, а $\pi[2]$ става 7:

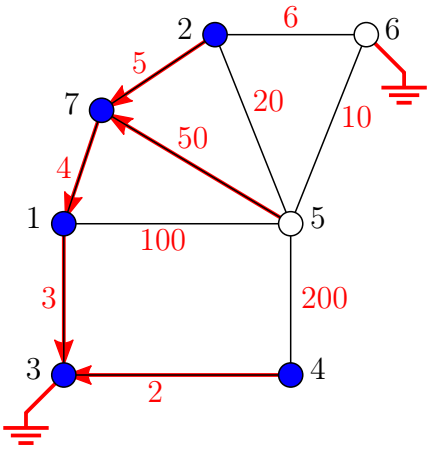


$S = \{1, 3, 4, 7\}$

d:

1	2	3	4	5	6	7
3	12	0	2	57	∞	7

for-цикълът престава да се изпълнява и отново сме на ❹. Има два върха извън S, чиято d-стойност не е безкрайност, така че отиваме на ❺, където x става 2. S става $\{1, 2, 3, 4, 7\}$:



$S = \{1, 2, 3, 4, 7\}$

d:

1	2	3	4	5	6	7
3	12	0	2	57	∞	7

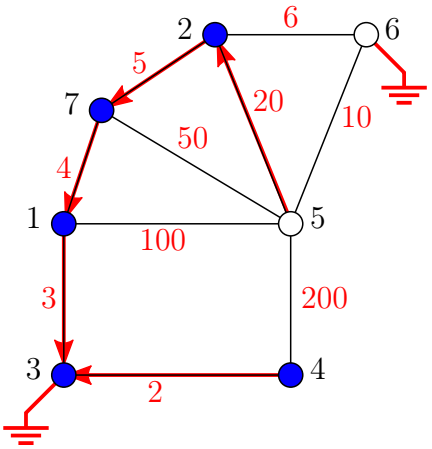
Връх 2 има три съседа: 5, 6 и 7. Да кажем, че първо y става 7. Това е опит за "връщане назад", но условието на ① е лъжа и ② не се изпълнява. Да кажем, че после y става 5. Булевото условие

$$d[y] > d[x] + w(x, y)$$

всъщност е

$$57 > 20 + 12$$

което е истина, така че изпълнението отива на ②. Там $d[5]$ става 32, а $\pi[5]$ става 2:



$S = \{1, 2, 3, 4, 7\}$

d:

3	12	0	2	32	∞	7
1	2	3	4	5	6	7

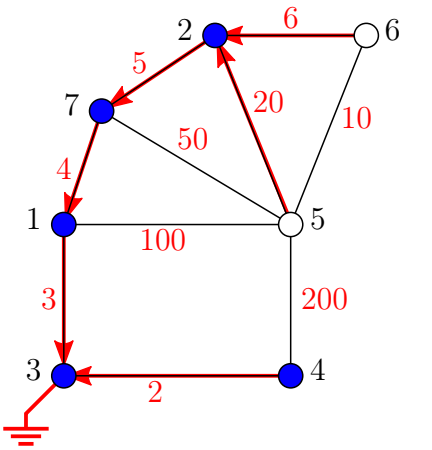
После y става 6. Булевото условие

$$d[y] > d[x] + w(x, y)$$

всъщност е

$$\infty > 12 + 6$$

което е истина, така че изпълнението отива на ②. Там $d[6]$ става 18, а $\pi[6]$ става 2:

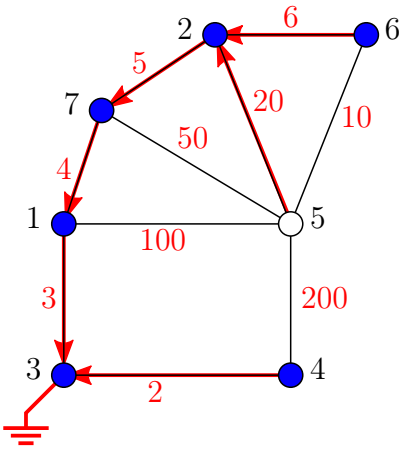


$S = \{1, 2, 3, 4, 7\}$

d:

3	12	0	2	32	18	7
1	2	3	4	5	6	7

for-цикълът престава да се изпълнява и отново сме на ❹. Има два върха извън S, чиято d-стойност не е безкрайност, така че отиваме на ❺, където x става 6. S става $\{1, 2, 3, 4, 6, 7\}$:



$S = \{1, 2, 3, 4, 6, 7\}$

d:

3	12	0	2	32	18	7
1	2	3	4	5	6	7

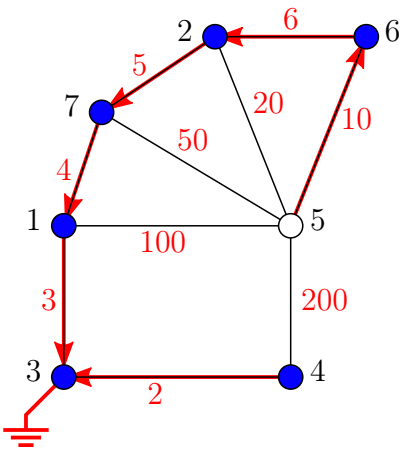
Връх 6 има две съседа: 2 и 5. Да кажем, че първо y става 2. Това е опит за "връщане назад", но условието на ① е лъжа и ② не се изпълнява. После y става 5. Булевото условие

$$d[y] > d[x] + w(x, y)$$

всъщност е

$$32 > 18 + 10$$

което е истина, така че изпълнението отива на ②. Там $d[5]$ става 28, а $\pi[5]$ става 6:

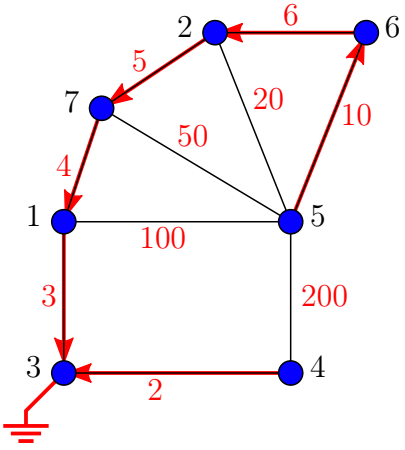


$S = \{1, 2, 3, 4, 6, 7\}$

d:

3	12	0	2	28	18	7
1	2	3	4	5	6	7

for-цикълът престава да се изпълнява и отново сме на ❹. Има един връх извън S, чиято d-стойност не е безкрайност, така че отиваме на ❺, където x става 5. S става $\{1, 2, 3, 4, 5, 6, 7\}$:



$S = \{1, 2, 3, 4, 5, 6, 7\}$

d:

3	12	0	2	28	18	7
1	2	3	4	5	6	7

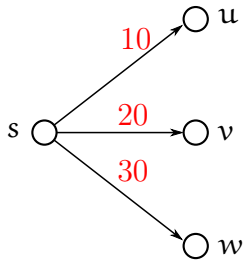
Връх 5 има пет съседа: 1, 2, 4, 6 и 7. За всеки от тях, когато "влезе" в y, булевото условие

$$d[y] > d[x] + w(x, y)$$

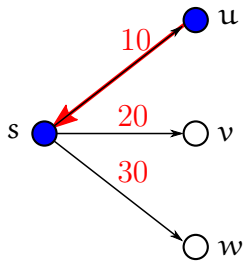
е лъжа, така че **for**-цикълът не променя нищо. Изпълнението отива на ❹, но сега $V = S$, така че $V \backslash S = \varnothing$, така че алгоритъмът връща d и π и спира.

Да се върнем на това, че в алгоритъма на Dijkstra теглата не може да са отрицателни. Дори да няма отрицателни цикли, алгоритъмът на Dijkstra не работи коректно, в общия случай, при наличие на отрицателни тегла. Ето малък пример за това. Примерът е с ориентирани графи, защото при неориентираните графи всяко ребро с отрицателно тегло би било третирано като отрицателен цикъл.

И така, ето фрагмент от ориентиран тегловен граф. Това **не е** целият граф, а само началният връх s и трите му деца u, v и w:



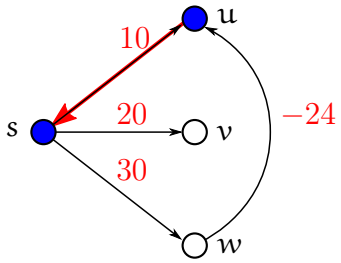
Както вече знаем, алгоритъмът първо слага s в S, като $d[s] = 0$ и после слага u в S, като $d[u] = 10$ и $\pi[u]$ става s.



Оттук насетне, алгоритъмът няма да промени никакъв атрибут на връх u: нито $d[u]$ ще се промени, нито $\pi[u]$ ще се промени. Това е очевидно от кода на Алгоритъм 10: на ред ❹ върховете от S се игнорират, така че няма как променливата x да стане u втори път, след като веднъж вече u е влязъл в S. С други думи, алгоритъмът смята, че със слагането на u в S, за u вече имаме точната информация, а не апроксимация – няма път от s до u с тегло, по-малко от 10.

Ако теглата са положителни, това е точно така. Път от s до u може да "напусне" s само през едно показаните три ребра. Ако такъв път ползва някое от ребрата (s, v) или (s, w), неговото тегло ще стане поне 20 или 30, което значи, че теглото му надхвърля 10. Ако такъв път ползва реброто (s, u), но съдържа и други ребра, пак теглото му ще надхвърля 10.

Ако обаче има отрицателни тегла, възможно е да има път от s до u с тегло, по-малко от 10. Ето как:



Сега има път от s до u с тегло 6, който минава през w. Алгоритъмът обаче няма как да знае за него, когато x съдържа s и y взема стойностите на съседите на $x = s$.

И така, при наличие на отрицателни тегла, убедеността, че няма по-къс до даден връх, която убеденост е основанието този връх да влезе в S, се получава по-трудно и по-бавно, в общия случай. Има алгоритми за най-къси пътища, които работят и при отрицателни тегла, стига да няма отрицателни цикли, но те са по-неефикасни от алгоритъма на Dijkstra.

4.5 Потоци в графи

Изложението в тази секция следва изложението на материала за потоци в графи във фундаменталната книга на Cormen, Leiserson, Rivest и Stein *Introduction to Algorithms* [16, глава 26, стр. 708].

Благодарности

Благодаря на **Иво Стратев** и **Иван Ганев** за откритите и коригирани грешки. Благодаря на **Румен Йордакиев** за откритата грешка в доказателството на Лема 4. Благодаря на **Добромир Кралчев** за многобройните открити и коригирани грешки. Благодаря на **Мартин Дачев** за откритата грешка в Определение 44. Благодаря на **Янита Терзиева** за откритата грешка в дефиницията на "матрица на съседство". Благодаря на **Марио Метушев** за откритията пропуск в Определение 15. Благодаря на **Марио Метушев** за няколко объркани прилагателни и отсъстващ идентификатор на връх в една от фигурите. Благодаря на **Александър Тангълов** за откритата грешка в Определение 28. Благодаря на **Драгомир Бинев Тодоров** за откритото несъответствие в примера за МПД, показан на Фигура 4.16. Благодаря на **Александър Тангълов** за откритите и коригирани многобройни правописни и смислови грешки. Благодарности на **Габриел Шумански** за откритата грешка в номера на една фигура. Благодарности на **Любомир Карагьозов** за откритите грешки в обясненията към Фигура 2.11 и във Фигура 2.46. Благодарности на **Иван Златев** за откритото двусмислие в Определение 29. Благодарности на **Акага Павлова** за откритата и коригирана грешка в примера с матрицата на съседство на стр. 250. Благодарности на **Акага Павлова** за откритата и коригирана неточност в условието на Теорема 46.

Библиография

[1] Ravindra K. Ahuja, Thomas L. Magnanti и James B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice hall, 1993.

[2] Martin Aigner и Günter M. Ziegler. *Proofs from THE BOOK*. 4th. Springer Publishing Company, Incorporated, 2009. ISBN: 3642008550.

[3] B. Andrasfai. *Graph Theory: Flows, Matrices*. Taylor & Francis, 1991. ISBN: 9780852742228. URL: https://books.google.bg/books?id=CYu7e-yap48C.

[4] K. Appel и W. Haken. "Every planar map is four colorable. Part I: Discharging". В: *Illinois J. Math.* 21.3 (септ. 1977). Копие на статията е достъпно онлайн на http://projecteuclid.org/euclid.ijm/1256049011, с. 429—490.

[5] K. Appel, W. Haken и J. Koch. "Every planar map is four colorable. Part II: Reducibility". В: *Illinois J. Math.* 21.3 (септ. 1977). Копие на статията е достъпно онлайн на http://projecteuclid.org/euclid.ijm/1256049012, с. 491—567.

[6] J.A. Baglivo и J.E. Graver. *Incidence and Symmetry in Design and Architecture*. Cambridge Urban and Architectural Studies. Cambridge University Press, 1983. ISBN: 9780521297844. URL: https://books.google.bg/books?id=35phQgAACAAJ.

[7] W.W.R. Ball и H.S.M. Coxeter. *Mathematical Recreations and Essays*. Dover Recreational Math Series. Копие на книгата е достъпно онлайн на http://www.gutenberg.org/files/26839/26839-pdf.pdf. Dover Publications, 1987. ISBN: 9780486253572.

[8] Seymour Benzer. "ON THE TOPOLOGY OF THE GENETIC FINE STRUCTURE". В: *Proceedings of the National Academy of Sciences* 45.11 (1959). Свободно достъпна на https://www.ncbi.nlm.nih.gov/pmc/articles/PMC222769/, с. 1607—1620. ISSN: 0027-8424. DOI: 10.1073/pnas.45.11.1607. eprint: https://www.pnas.org/content/45/11/1607.full.pdf.

[9] C. Berge. *Hypergraphs: Combinatorics of Finite Sets*. North-Holland Mathematical Library. Elsevier Science, 1984. ISBN: 9780080880235. URL: https://books.google.bg/books?id=jEyfse-EKf8C.

[10] C. Berge и E. Minieka. *Graphs and Hypergraphs*. Graphs and Hypergraphs. North-Holland Publishing Company, 1973. ISBN: 9780444103994. URL: https://books.google.bg/books?id=X32GlVfqXjsC.

[11] Claude Berge. "Some classes of perfect graphs". В: с. 155—165.

[12] N. Biggs, E. K. Lloyd и R. J. Wilson. *Graph Theory, 1736-1936*. New York, NY, USA: Clarendon Press, 1986. ISBN: 0-198-53916-9.

[13] J.A. Bondy и U.S.R Murty. *Graph Theory*. 1st. Springer Publishing Company, Incorporated, 2008. ISBN: 1846289696.

[14] A. Bretto. *Hypergraph Theory: An Introduction*. Mathematical Engineering. Springer International Publishing, 2013. ISBN: 9783319000800. URL: https://books.google.bg/books?id=lb5DAAAAQBAJ.

[15] P.J. Cameron. *Combinatorics: Topics, Techniques, Algorithms*. Достъпна онлайн на https://books.google.bg/books?id=_aJIKWcifDwC. Cambridge University Press, 1994. ISBN: 9780521457613.

[16] Thomas H. Cormen и др. *Introduction to Algorithms, Third Edition*. 3rd. The MIT Press, 2009. ISBN: 0262033844, 9780262033848.

[17] N. Deo. *Graph Theory with Applications to Engineering and Computer Science*. Dover Publications, 2017. ISBN: 9780486820811. URL: https://books.google.bg/books?id=DSBMDgAAQBAJ.

[18] R. Diestel. *Graph Theory*. Electronic library of mathematics. Springer, 2006. ISBN: 9783540261834.

[19] E. W. Dijkstra. "A Note on Two Problems in Connexion with Graphs". В: *Numer. Math.* 1.1 (дек. 1959). Свободно достъпна на http://www-m3.ma.tum.de/foswiki/pub/MN0506/WebHome/dijkstra.pdf, с. 269—271. ISSN: 0029-599X. DOI: 10.1007/BF01386390. URL: https://doi.org/10.1007/BF01386390.

[20] G. A. Dirac. "Some Theorems on Abstract Graphs". В: *Proceedings of the London Mathematical Society* s3-2.1 (1952). Достъпна онлайн на http://dx.doi.org/10.1112/plms/s3-2.1.69, с. 69—81. ISSN: 1460-244X. DOI: 10.1112/plms/s3-2.1.69.

[21] Rodney G. Downey и M. R. Fellows. *Parameterized Complexity*. Springer Publishing Company, Incorporated, 2012. ISBN: 1461267986, 9781461267980.

[22] Susanna S. Epp. *Discrete Mathematics with Applications*. 4th. USA: Brooks/Cole Publishing Co., 2010. ISBN: 0495391328.

[23] Leonhard Euler. "Demonstratio nonnullarum insignium proprietatum, quibus solida hedris planis inclusa sunt praedita". В: *Novi Commentarii Academiae Scientiarum Imperialis Petropolitanae* 4 (1758). Достъпна онлайн на http://eulerarchive.maa.org/pages/E053.html, с. 140—160.

[24] Leonhard Euler. "Solutio problematis ad geometriam situs pertinentis". В: *Commentarii Academiae Scientiarum Imperialis Petropolitanae* 8 (1736). Достъпна онлайн на http://eulerarchive.maa.org/pages/E053.html, с. 128—140.

[25] L.R. Foulds. *Graph Theory Applications*. Universitext. Springer New York, 1995. ISBN: 9780387975993. URL: https://books.google.bg/books?id=IK7kreGl3vkC.

[26] M. R. Garey и D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences)*. First Edition. W. H. Freeman, 1979. ISBN: 0716710455.

[27] A. Gibbons. *Algorithmic Graph Theory*. Cambridge University Press, 1985. ISBN: 9780521288811.

[28] M.C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. ISSN. Elsevier Science, 2004. ISBN: 9780080526966. URL: https://books.google.bg/books?id=8xo-VrWo5%5C_QC.

[29] Martin Charles Golumbic. "Interval graphs and related topics". В: *Discrete Mathematics* 55.2 (1985), с. 113—121. ISSN: 0012-365X. DOI: https://doi.org/10.1016/0012-365X(85)90039-1. URL: http://www.sciencedirect.com/science/article/pii/0012365X85900391.

[30] Irina Gribkovskaia, Øyvind Halskau и Gilbert Laporte. "The bridges of Königsberg - A historical perspective." В: *Networks* 49.3 (ноем. 2007), с. 199—203.

[31] J.L. Gross, J. Yellen и M. Anderson. *Graph Theory and Its Applications*. Textbooks in mathematics. CRC Press, Taylor & Francis Group, 2018. ISBN: 9781482249484. URL: https://www.routledge.com/Graph-Theory-and-Its-Applications/Gross-Yellen-Anderson/p/book/9781482249484.

[32] Frank Harary. *Graph Theory*. Reading, MA: Addison-Wesley, 1969.

[33] Frank Harary, ред. *Graph theory and theoretical physics*. 1963, Paris: Academic press, 1967.

[34] D.A. Holton и J. Sheehan. *The Petersen Graph*. Australian Mathematical Society Lecture Series. Cambridge University Press, 1993. ISBN: 9780521435949.

[35] E.C. Kirby и др. "What Kirchhoff Actually did Concerning Spanning Trees in Electrical Networks and its Relationship to Modern Graph-Theoretical Work". В: *Croatica Chemica Acta* 89.4 (2016). Достъпна онлайн на http://hrcak.srce.hr/file/255139. DOI: 10.5562/cca2995.

[36] Edward C. Kirby и др. "What Kirchhoff Actually did Concerning Spanning Trees in Electrical Networks and its Relationship to Modern Graph-Theoretical Work". В: *Croatica Chemica Acta* 89.4 (2016). Достъпна онлайн на https://hrcak.srce.hr/file/255139, с. 403—417. DOI: 10.5562/cca2995.

[37] G. Kirchhoff. "On the Solution of the Equations Obtained from the Investigation of the Linear Distribution of Galvanic Currents". В: *IRE Transactions on Circuit Theory* 5.1 (1958), с. 4—7.

[38] G. Kirchhoff. "Ueber die Auflösung der Gleichungen, auf welche man bei der Untersuchung der linearen Vertheilung galvanischer Ströme geführt wird". В: *Annalen der Physik* 148.12 (1847). Достъпна чрез платен достъп или оторизирана институция на https://onlinelibrary.wiley.com/doi/abs/10.1002/andp.18471481202, с. 497—508. DOI: 10.1002/andp.18471481202.

[39] Casimir Kuratowski. "Sur le problème des courbes gauches en Topologie". fre. В: *Fundamenta Mathematicae* 15.1 (1930). Достъпна онлайн на http://matwbn.icm.edu.pl/ksiazki/fm/fm15/fm15126.pdf, с. 271—283.

[40] Kasimierz Kuratowski. "On the problem of skew curves in topology". В: *Graph theory: proceedings of a conference held in Łagów, Poland, February 10-13, 1981*. Под ред. на M. Borowiecki, J.W. Kennedy и M.M. Systo. Graph theory: proceedings of a conference held in Łagów, Poland, February 10-13, 1981 no. 1018. Преведена на английски от Jan Jaworowski. Достъпна, но не със свободен достъп, онлайн на https://link.springer.com/chapter/10.1007/BFb0071605. Springer Verlag, 1983. ISBN: 9780387126876.

[41] László Lovász. В: *Bulletin of the American Mathematical Society* 43 (2006). Свободно достъпна на http://www.ams.org/bull/2006-43-01/S0273-0979-05-01088-8/S0273-0979-05-01088-8.pdf, с. 75—86. DOI: https://doi.org/10.1090/S0273-0979-05-01088-8.

[42] Brendan D. McKay и Adolfo Piperno. "Practical graph isomorphism, II". В: *Journal of Symbolic Computation* 60 (2014). Свободно достъпна на http://www.sciencedirect.com/science/article/pii/S0747717113001193, с. 94—112. ISSN: 0747-7171. DOI: https://doi.org/10.1016/j.jsc.2013.09.003.

[43] Z. Michalewicz и D.B. Fogel. *How to Solve It: Modern Heuristics*. Springer Berlin Heidelberg, 2004. ISBN: 9783540224945.

[44] Gregory H. Moore. "The evolution of the concept of homeomorphism". В: *Historia Mathematica* 34.3 (2007). Достъпна онлайн на https://www.sciencedirect.com/science/article/pii/S0315086006000863, с. 333—343. ISSN: 0315-0860. DOI: 10.1016/j.hm.2006.07.006.

[45] Øystein Ore. "Note on Hamilton Circuits". В: *The American Mathematical Monthly* 67.1 (1960), с. 55. DOI: 10.2307/2308928.

[46] Xavier Ouvrard. *Hypergraphs: an introduction and review*. Свободно достъпна на https://arxiv.org/pdf/2002.05014. 2020.

[47] Christos H. Papadimitriou. *Computational complexity*. Addison-Wesley, 1994, с. I—XV, 1—523. ISBN: 978-0-201-53082-7.

[48] Nikos Paragios, Yunmei Chen и Olivier D. Faugeras, ред. *Handbook of Mathematical Models in Computer Vision*. Копие на пета глава от книгата *Graph Cuts in Vision and Graphics: Theories and Applications* е свободно достъпно онлайн на http://luthuli.cs.uiuc.edu/~daf/courses/Optimization/Combinatorialpapers/Nikos-Yuri-Olga.pdf. Springer, 2006. ISBN: 978-0-387-26371-7. DOI: 10.1007/0-387-28831-7.

[49] Robert Clay Prim. "Shortest Connection Networks And Some Generalizations". В: *The Bell Systems Technical Journal* 36.6 (1957). Свободно достъпна на https://archive.org/details/bstj36-6-1389, с. 1389—1401.

[50] Heinz Prüfer. "Neuer Beweis eines Satzes über Permutationen". В: *Archiv der Mathematischen Physik* 27 (1918), с. 742—744.

[51] N. Robertson и P.D. Seymour. "Graph Minors. XIII. The Disjoint Paths Problem". В: *Journal of Combinatorial Theory, Series B* 63.1 (1995). Свободно достъпна на https://www.sciencedirect.com/science/article/pii/S0095895685710064, с. 65—110. ISSN: 0095-8956. DOI: https://doi.org/10.1006/jctb.1995.1006.

[52] Neil Robertson и P.D. Seymour. "Graph minors. I. Excluding a forest". В: *Journal of Combinatorial Theory, Series B* 35.1 (1983). Свободно достъпна на https://www.sciencedirect.com/science/article/pii/0095895683900795, с. 39—61. ISSN: 0095-8956. DOI: https://doi.org/10.1016/0095-8956(83)90079-5.

[53] Neil Robertson и P.D. Seymour. "Graph Minors. XX. Wagner's conjecture". В: *Journal of Combinatorial Theory, Series B* 92.2 (2004). Свободно достъпна на https://www.sciencedirect.com/science/article/pii/S0095895604000784, с. 325—357. ISSN: 0095-8956. DOI: https://doi.org/10.1016/j.jctb.2004.08.001.

[54] K.H. Rosen. *Discrete Mathematics and Its Applications*. 7-е изд. McGraw-Hill, 2012. ISBN: 9780077631154. URL: https://books.google.bg/books?id=8sayMgEACAAJ.

[55] Steven S. Skiena. *The Algorithm Design Manual*. Springer, 2008. ISBN: 9781848000704 1848000707 9781848000698 1848000693. DOI: 10.1007/978-1-84800-070-4.

[56] R.P. Stanley. *Algebraic Combinatorics: Walks, Trees, Tableaux, and More*. Undergraduate Texts in Mathematics. Авторът е предоставил свободно копие на книгата, без упражненията, на адрес http://www-math.mit.edu/~rstan/algcomb/algcomb.pdf. Springer New York, 2013. ISBN: 9781461469988.

[57] John Joseph Sylvester. "Chemistry and Algebra". В: *Nature* 17 (1878). Свободно достъпна на https://www.nature.com/articles/017284a0, с. 284. DOI: 10.1038/017284a0.

[58] K. Thulasiraman и др. *Handbook of Graph Theory, Combinatorial Optimization, and Algorithms*. Chapman & Hall/CRC Computer and Information Science Series. Taylor & Francis, 2015. ISBN: 9781584885955.

[59] K. Wagner. “Über eine Eigenschaft der ebenen Komplexe”. В: *Mathematische Annalen* 114 (1937). Пълният текст на немски може да бъде свален свободно през http://eudml.org/doc/159935., с. 570—590.

[60] Klaus Wagner. *Graphentheorie*. Bibliographisches Institut, 1970.

[61] Denny Zhou, Jiayuan Huang и Bernhard Schölkopf. “Learning with Hypergraphs: Clustering, Classification, and Embedding”. В: *Advances in Neural Information Processing Systems 19: Proceedings of the 2006 Conference*. Свободно достъпна на https://www.microsoft.com/en-us/research/publication/learning-hypergraphs-clustering-classification-embedding. MIT Press, септ. 2007, с. 1601—1608.

[62] Вадим Визинг. “Об оценке хроматического класса p-графа”. В: *сб. Дискретный анализ. – Новосибирск: Институт математики СО АН СССР* (1964), с. 25—30.