ДС Семинар №13
Алгоритми върху графи /
Задачи върху графи

1) Обхождане на графи

- задачата е дефинирана върху ориентирани мултиграфи с възможни примки
- обхождаме върховете и ребрата
- не трябва да пропускаме (връх / ребро)
- не трябва да зацикляме
- обхождането започва от някой стартов връх v.

• Всеки връх ще има три състояния:
~~v кол~~

1) Неоткрит
2) Открит
3) Финализиран

Обработката на върховете ще става в посока от 1) към 3)
Нека v е връх в някой граф G. Върхът v е неоткрит, ако след пускането на някой алгоритъм за обхождане, v не е бил "видян" / "достигнат".
Върхът v е открит, ако алгоритъмът е "достигнал" v, но не е "достигнал" всички негови съседи/деца.
Върхът v е финализиран, ако алгоритъмът е "достигнал" v и е "достигнал" всички негови съседи (такива може и да няма).

• Ще поддържаме някаква структура от данни S, която ще съдържа откритите (не финализирани) върхове.
В началото S съдържа само стартовият връх.

· За да финализираме открит връх v, ще ~~полаз~~ "минаваме"/дете по всяко ребро, излизащо от v, за да открием всеки съсед и на v,

Ако графът има неориентирано ребро (u, v), то това ребро ще се "обходи" два пъти - веднъж, намирайки се във върха u, откривайки неговите съседи, и веднъж, намирайки се във върха v, откривайки неговите съседи.

Ако реброто е ~~не~~ориентирано, през него ще се "мине" точно веднъж.
По тази схема ще открием свързаната/с.св-та компонента, съдържаща стартовия връх.

1) BFS

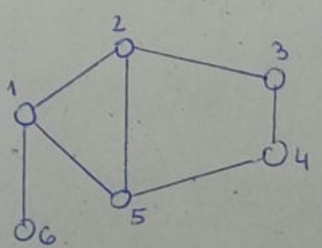
- обхождане по ширина
- "строи" дърво на свързаната компонента, съдържаща стартовия връх, на най-късите пътища (в нетегловни графи) от корена (стартовия връх) до всички останали
- тук структурата данни/множеството S, споменато на предната страница, се реализира чрез АТД опашка.
- неоткритите/непосетени върхове условно са бели
- откритите/посетени върхове условно са сиви
- финализираните върхове условно са черни.
- BFS "строи" антиарборесценция. Всяко ребро бива обходено от BFS, но не всяко ребро влиза в дървото на обхождане. Влизат само ребрата, с които BFS открива непосетени/бели върхове.

- BFS обработва върховете по нарастващо разстояние от избрания начален връх

Псевдокод на BFS

```
BFS(G - ор. мултиграф, V={1,...,n})
for i←1 to n
    colour[i]←white; d[i]←∞, π[i]←Nil
colour[1]←grey, d[1]←0
Enqueue(Q,1)
while !Q.empty()
    x←Dequeue(Q)
    for y∈adj[x]
        if colour[y]=white
            colour[y]=grey
            d[y]←d[x]+1
            π[y]←x
            Enqueue(Q,y)
    colour[x]←black
return colour, π, d.
```

Примерна работа на BFS на граф G, представен:

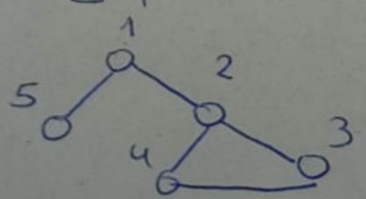


и списъци на съседство:

1: 2, 5, 6
2: 1, 3, 5
5: 1, 2, 4
6: 1
3: 2, 4
4: 5, 3

↳ за упражнение

// за упражнение:



1: 5, 2
2: 4, 3
3: 2, 4
4: 2, 3
5: 1

Стартов връх 1.

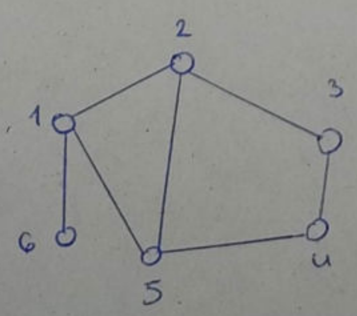
2) DFS

- обхождане в дълбочина
- **не** намира най-късите разстояния от началния връх до всички, достижими от него
- "оцветяването" на върховете е същото като при BFS
- разликата между BFS и DFS се изразява в реда на обхождане на върховете. Този ред зависи от структурата данни, която сме избрали за S.
- обхождането чрез DFS „бързо" се отдалечава от стартовия връх.

Псевдокод на DFS:

```
DFS(..)...
                              стартов връх за текущото пускане на DFS
DFS-VISIT (G=(V,E), x∈V)
colour[x] ← grey
foreach y∈adj[x]
    if colour[y] = while
        π[y] ← x
        DFS-VISIT(G,y)
colour[x] ← black
```

Примерна работа на DFS на граф G, представен и списъци на съседство:



1 : 2, 5, 6
2 : 1, 3, 5
5 : 1, 2, 4
6 : 1
3 : 2, 4
4 : 5, 3

↳ за упражнение

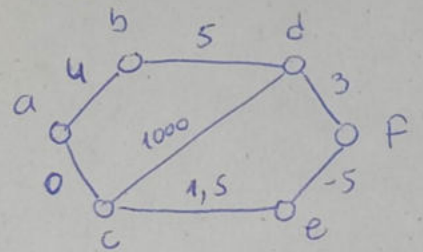
// Да се сравни дървото, представено чрез масива от родителите π, на DFS и BFS върху примера

• Тегловен ориентиран мултиграф е наредена четворка $G = (V, E, f_G, w)$, където $V \neq \emptyset$ е м-во от върхове, E е м-во от ребра, $V \cap E = \emptyset$,

$f_G : E \rightarrow V \times V$ е свързващата функция и

$w : E \rightarrow \mathbb{R}$ е тегловната функция

// Примерно представяне на някакъв тегловен граф:



def: Нека $G = (V, E, f_G, w)$ е тегловен граф и T е покриващо дърво на G:

$$w(T) = \sum_{e \in T} w(e)$$

// Минимални покриващи дървета

. Неориентирани свързани тегловни графи.

- гледайки само броя ребра, всяко покриващо дърво на G е минимален свързан подграф на G, докато минимално покриващо дърво на граф G е такова покриващо дърво на G, чиято сума от ребрата е минимална

Ако разглеждаме нетегловен граф G, то всяко покриващо дърво е минимално (тъй като критерият е броят ребра) и такова може да се открие чрез DFS или BFS.

· МПД Теорема

Нека $G = (V, E)$ е неориентиран свързан теглови граф, с теглова функция $w: E \to R$. Нека $\{U, W\}$ е произволен срез в G. Тогава, за всяко ребро от срез множеството, което е с минимално тегло, съществува МПД, което го съдържа.

1) Алгоритъм на Prim

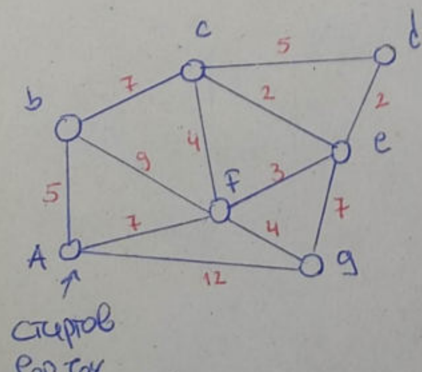
Алгоритъмът започва от даден стартов връх и „разширява" останалата част от дървото ребро по ребро докато включи всички върхове на свързания неориентиран теглови граф.

На най-високо ниво алгоритъмът следва схемата:

Prim-MST(G)
~~Select~~
Избери произволен стартов връх от G.
Докато (съществува не-дървесен връх)
Избери ребро с минимално тегло от срез множеството.
Добави избраното ребро към дървото.

Коректността на алгоритъма се основава на МПД теоремата.

Примерна работа на алг. на Prim на граф G, представен:



↳ за упражнение

2) Алгоритъм на Kruskal

- по-ефикасен върху разредени графи
- алчен алгоритъм, също като алгоритъма на Prim
- не започва с конкретен стартов връх
- изгражда свързани компоненти от върховете на G, които завършват в покриващо (минимално) дърво на G (ако е свързан)
- в началото всеки връх представлява свързана компонента
- алгоритъмът "поддържа" ребро с минимално тегло и проверява дали краищата му са от различни свързани компоненти A и B, ако да, добавяме реброто в изграждащото дърво и A и B стават една свързана компонента

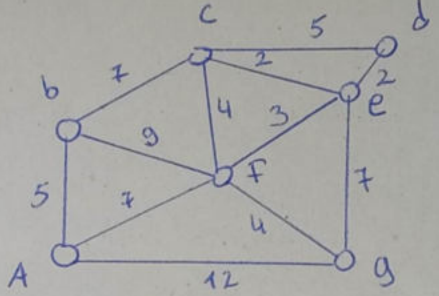
Тъй като всяка свързана компонента, в контекста на алгоритъма, е дърво, добавяйки единствено ребро между две дървета, няма как да получим цикъл.

На най-високо ниво алгоритъмът следва схемата

- Kruskal-MST(G)
  - Построй покриваща гора F на G.
  - Вкарай ребрата в приоритетна опашка Q, сортирани по тегла.
  - Ако F има една свързана компонента, върни F.
  - В противен случай вземи първото ребро e в Q, чийто краища са от различни свързани компоненти A и B, и го сложи в дървото и слей двете свързани компоненти A и B.

- **коректността на алгоритъма се основава на МПД теоремата.**

Примерна работа на алгоритъма на Kruskal на граф G, представен



↳ за упражнение

II) Най-къси пътища в графи

Ориентирани теглови графи

- има се предвид най-леки пътища в графи,
- ако ребрата нямат тегла или всички ребра са с еднакво тегло, BFS върши работа за намиране на най-късите пътища от даден връх до всички останали
- алгоритъмът на Dijkstra не работи коректно при наличие на отрицателни тегла на ребра

1) Алгоритъм на Dijkstra

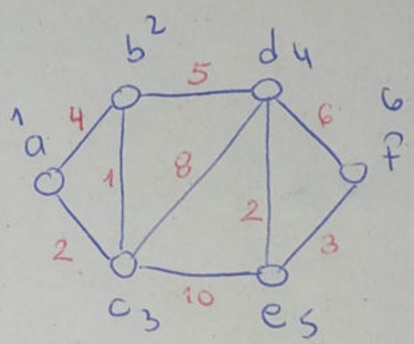
- намира най-къси пътища от даден връх s до всички останали.

На най-високо ниво алгоритъмът на Dijkstra следва схемата:

Shortest-Path-Dijkstra (G, s)

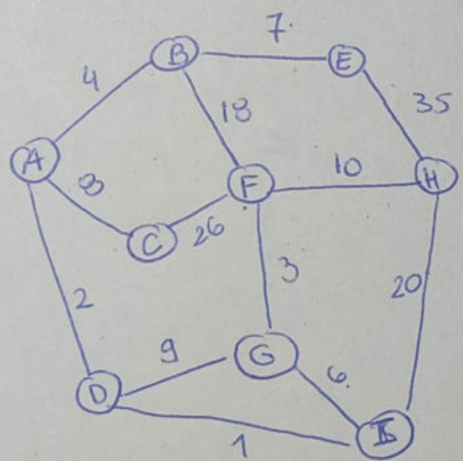
1. За всеки връх $v \in V$: $d[v] \leftarrow \infty$, $\pi[v] \leftarrow Nil$
2. $d[s] \leftarrow 0$
3. $S \leftarrow \emptyset$
4. Ако във $V \setminus S$ няма връх u: $d[u] < \infty$, върни d и π
5. В противен случай, вземи $x \in V \setminus S$: $d[x]$ е минимално
6. $S \leftarrow S \cup \{x\}$
7. За всеки съсед $y \in adj[x]$:
8. Ако $d[y] > d[x] + w(x, y)$, то
9. $d[y] = d[x] + w(x, y)$.
10. Отиди на 4.

Примерна работа на алгоритъма на Dijkstra върху граф G, представен: (стартовият връх е a) (Търсим най-къс път от a до всички останали)



↳ за упражнение.

зад 1) На чертежа е представен неориентиран теглови граф ~ Изпит



а) Намерете най-къс път от A до H. Изберете алгоритъм и опишете стъпките

б) Хамилтонов ли е графът?

в) Ойлеров ли е графът?

Решение:

а) Ще симулираме алгоритъма на Dijkstra с начален връх A.

Резултатът е масив от разстояния d и масив на родителство π.

	A	B	C	D	E	F	G	H	I
d	0	4	8	2	11	12	9	22	3

	A	B	C	D	E	F	G	H	I
π	Nil	A	A	A	B	G	I	F	D

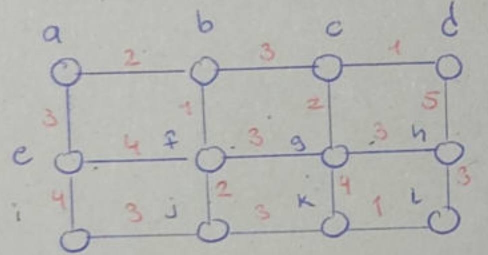
от масива d виждаме, се най-късият път от A до H е с дължина 22 и от масива П може да пресъздадем самия път.

б) графът е Хамилтонов - примерен Х.цикъл е

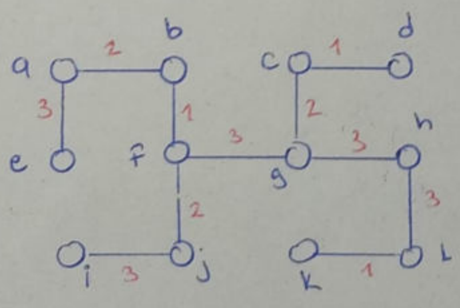
p = ACFBEHIGDA

в) графът не е Ойлеров - не всички върхове са от четна степен

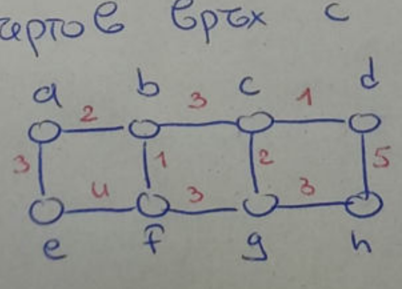
зад(2) Да се намери МПД чрез алгоритъма на Kruskal на графа, представен чрез:



Покриващо мин. дърво:

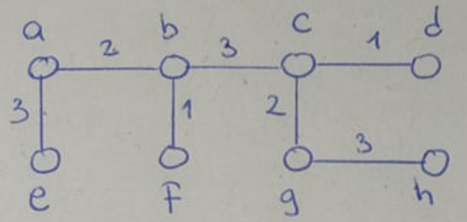


зад(3) Да се намери МПД чрез алгоритъма на Prim със стартов връх c на следния граф:

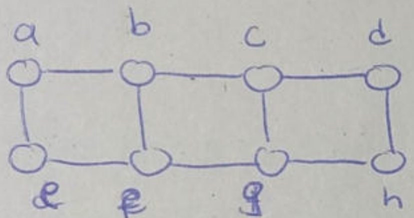


Решение:

Примерно МПД, построено чрез алгоритъма на Prim



зад. ④ Да се намери броя на покриващите дървета на граф G, представен:

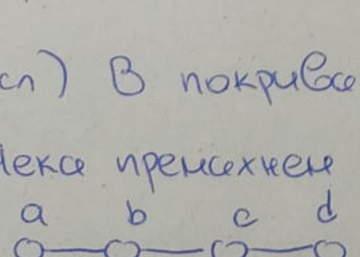


Решение:

Всяко покриващо дърво T на G съдържа всички върхове на G. Знаем, че за всяко дърво е в сила $m=n-1$. Графът от задачата има 8 върха и 10 ребра, следователно трябва да премахнем 3 ребра, така че полученият граф да е дърво. Ще разгледаме следните 4 случая:

1сл) В покриващото дърво T НЕ присъстват ребрата (b,f) и (c,g)

Нека премахнем тези ребра от G:

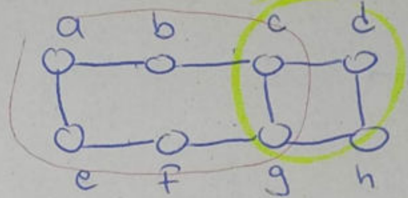


G'

За да получим покриващо дърво T трябва да премахнем едно ребро от цикъла - G'. Това може да направим по 8 начина.

Покриващите дървета от 1сл) са 8 на брой.

2сл) В покриващото дърво T НЕ присъства реброто (b,f), но реброто (c,g) присъства. Нека премахнем реброто (b,f) от G:



G'

Трябва да премахнем две ребра от G' така че новополученият граф да е покриващо дърво на G.

За първото ребро има 5 възможности - може да премахнем всяко от ребрата в подграфа, ограден с червено, с изключение на реброто (c,g).

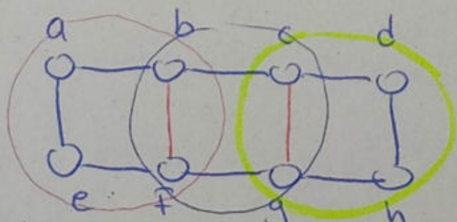
За второто ребро има 3 възможности - може да премахнем всяко от ребрата в зелени подграф, с изключение на (c,g)

Покриващите дървета от 2сл) са 5.3=15 на брой.

3сл) В покриващото дърво T НЕ присъства реброто (c,g), но реброто (b,f) присъства.

Аналогично на 2сл), получаваме, че покриващите дървета от 3сл) са 15 на брой

4сл) В покриващото дърво T присъстват ребрата (c,g) и (b,f)



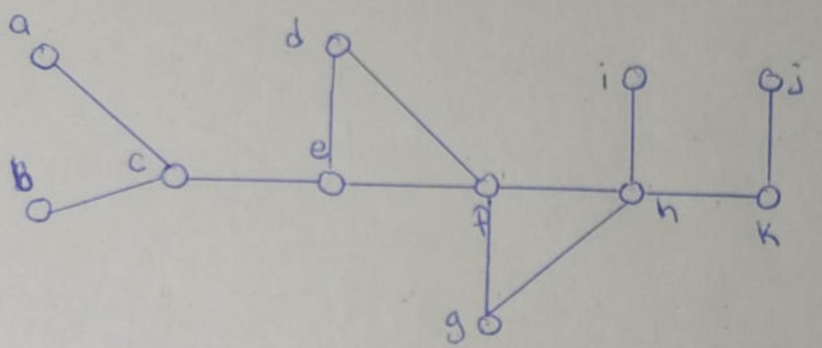
Трябва да премахнем три ребра така че новополученият граф да е покриващо дърво, - трябва да премахнем по едно ребро от подграфите, оградени с червено, жълто и ~~синьо~~ синьо съответно, запазвайки ребрата (b,f) и (c,g).

Последното може да направим по 3.2.3 = 18 начина.

Покриващите дървета от 4сл) са 18 на брой.

Случаите 1, 2, 3 и 4 са изчерпателни. Броят покриващи дървета на G е 8 + 15 + 15 + 18 = 56.

зад 5 Симулирайки обхождане с DFS, да се намери покриващо дърво на граф G, представен: (със стартов връх f)



със списъци на съседство:

f: g, d, h, e

g: h, f

h: k, i, g, f

j: k

d: e, f

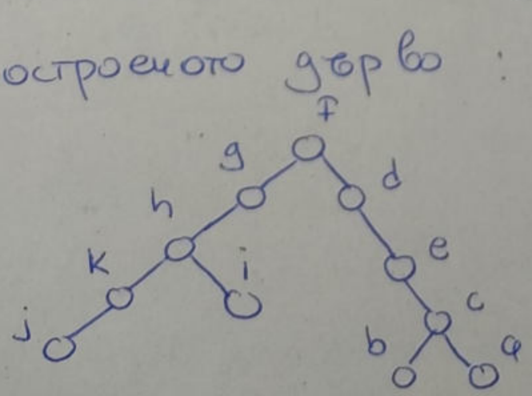
e: c, d, f

c: a, b

c: ab
i: h
k: j, h
a: c
b: c

... (останалите списъци няма да бъдат използвани)

Построеното дърво:



– да се направи същото, но чрез BFS