ДОМАШНО № 2 ПО ДИЗАЙН И АНАЛИЗ НА АЛГОРИТМИ

А Л Г О Р И Т М И В Ъ Р Х У Г Р А Ф И

Указания:

1) Всички решения да бъдат обосновани подробно.
2) Алгоритмите, изучени на лекции, могат да се използват наготово.
3) При всяко обхождане на граф да се уточнява видът на обхождането.
4) Названията на всички алгоритми и структури от данни трябва да бъдат на български език.

Задача 1. Нека G е неориентиран граф без кратни ребра и без примки. Нека s и t са върхове на G, $s \neq t$. По ребрата на G се движат двама пешеходци. Единият пешеходец тръгва от s и трябва да пристигне в t, а другият тръгва от t и трябва да пристигне в s. На всеки ход се придвижва само единият пешеходец, като се премества в съседен връх на графа. Двамата не са длъжни да се редуват: всеки от тях може да направи няколко хода последователно.

Да се състави алгоритъм, който намира редица от ходове, с чиято помощ двамата пешеходци могат да се разминат, без да се срещнат. Тоест:

— Пешеходецът, който тръгва от s, трябва да пристигне в t.
След първото си попадане в t той спира да се движи.
— Пешеходецът, който тръгва от t, трябва да пристигне в s.
След първото си попадане в s той спира да се движи.
— Пешеходците не бива да се намират едновременно в един и същи връх.

Предполага се, че графът е представен чрез списъци на съседствата. Алгоритъмът трябва да има линейна времева сложност $\Theta(m + n)$ при най-лоши входни данни (n и m са съответно броят на върховете и на ребрата на графа G).

Опишете алгоритъма с думи възможно най-ясно. Дайте по един пример за двата случая — когато пешеходците могат да се разминат и когато не могат.

Анализирайте времевата сложност и докажете коректността на алгоритъма с максимално строги разсъждения.

Задача 2. Неориентиран свързан граф без кратни ребра и без примки е зададен чрез списъци на съседствата. Графът притежава n върха и m ребра. Всяко ребро на графа е оцветено в бяло или в черно (оцветяването е дадено). Съставете алгоритъм с времева сложност $\Theta(m + n)$ при най-лоши входни данни, който намира покриващо дърво с възможно най-много бели ребра. Предложете словесно описание, псевдокод и анализ на алгоритъма и структурите от данни.

ТОЧКУВАНЕ

Представените решения на задачите от домашното се оценяват в проценти. Пълни решения на двете задачи носят общо 100 % — по 50 % за всяка задача.

РЕШЕНИЯ

Задача 1. Намираме мостовете на графа G чрез ***алгоритъма на Шмит.*** Първия етап (обхождането в дълбочина) пускаме от върха s, а в края на етапа проверяваме дали върхът t е бил обходен. Ако не — то няма път между s и t, затова пешеходците не могат да се разминат и алгоритъмът приключва работа.

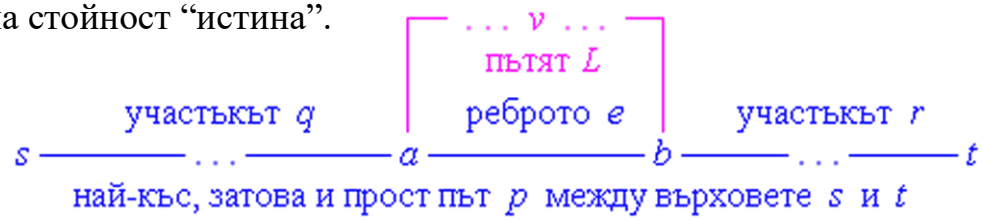
В противен случай има път между s и t. Чрез обхождане в ширина намираме най-къс път p между s и t. Следователно p е прост път, тоест не повтаря нито върхове, нито ребра. От изхода на алгоритъма на Шмит знаем всички мостове в компонентата на свързаност на G, съдържаща s и t.

Първи случай: съществува ребро $e = \{a\ ;\ b\}$ от пътя p, което не е мост. Тъй като графът G не съдържа примки, то a и b са различни върхове. Нека сме именували върховете a и b така, че при движение по p от s към t преминаваме по e от a към b. Да означим с q частта от p между s и a вкл., а пък с r — частта от p между b и t вкл. Понеже всеки участък от най-къс път също е най-къс път, то всеки участък от p е най-къс път, затова няма ребро между връх от q и връх от r освен реброто e между върховете a и b, а то е единствено, понеже G не съдържа кратни ребра. Тъй като p е прост път, то $t \notin q$, $b \notin q$, $s \notin r$ и $a \notin r$.

Премахваме реброто e от графа G. Щом e не е мост, то между a и b все още има път. Обхождайки новия граф в ширина, намираме най-къс път L между върховете a и b.

Да допуснем, че всички върхове на L са от p. Обхождаме L от a към b; понеже $a \in q$ и $b \in r$, поне веднъж преминаваме от връх на q към връх на r. Единственото такова ребро е реброто e, а то беше изтрито, така че не може да участва в пътя L. Това противоречие показва, че допускането не е вярно. Следователно L съдържа поне един връх $v \notin p$. Тъй като p съдържа a и b, то върхът v е различен от a и b.

Намирането на върха v става с идеята на ***сортирането чрез броене:*** Нека върховете на графа са номерирани с целите числа от 1 до n вкл. Инициализираме един логически масив $C[1 \ldots n]$ със стойности “истина”. Обхождаме пътя p, например в посока от s към t, и за всеки връх $u \in p$ присвояваме стойност “лъжа” на елемента $C[u]$. После обхождаме пътя L, например в посока от a към b, и спираме при първия срещнат връх $v \in L$, за който $C[v]$ има стойност “истина”.



Сега двамата пешеходци могат да се разминат по следния начин:

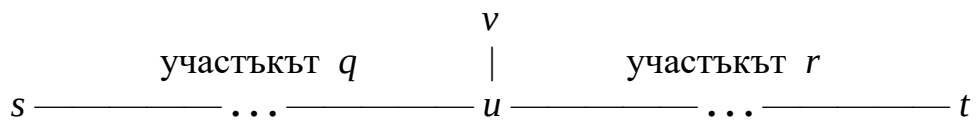
1) Пешеходецът от s отива във върха a по пътя q.
2) Пешеходецът от t отива във върха b по пътя r. При тези движения пешеходците не се срещат, защото $t \notin q$ и $a \notin r$ (доказано по-горе).
3) Пешеходецът от s отива от a във v по съответния участък от пътя L. Тъй като $a \neq b$, $v \neq b$ и L е най-къс, следователно прост път от a до b, то въпросният участък не минава през b, т.е. пешеходците не се срещат.
4) Пешеходецът от t отива от b в a по реброто e. Понеже $v \neq a$, то и сега пешеходците не се срещат.
5) Пешеходецът от s отива от v в b по следващия участък от L. Не се среща с другия пешеходец, защото споменатият участък не минава през a (доказва се като твърдението от стъпка № 3).
6) Пешеходецът от t отива от a в s по пътя q.
7) Пешеходецът от s отива от b в t по пътя r. На стъпки № 6 и № 7 пешеходците не се срещат, защото $b \notin q$ и $s \notin r$ (доказано по-горе).

Сортирането чрез броене се изпълнява за време $\Theta(n)$, а другите етапи, тоест обхожданията на графа, включително тези от алгоритъма на Шмит — за време $\Theta(m + n)$. Следователно общото време на предложения алгоритъм е от асимптотичен порядък $\Theta(m + n)$ при всякакви входни данни.

В седемте стъпки по-горе има пропуск: не е отчетено, че всеки пешеходец е длъжен да спре, когато достигне крайната си цел. Това не създава проблем за пешеходеца, отиващ от t в s: той се движи само по p (стъпки № 2, № 4 и № 6) и понеже p е прост път, пешеходецът се озовава в s едва в края на стъпка № 6. В по-различно положение е пешеходецът, който отива от s в t: той се движи по пътя qLr, който може и да не е прост, при все че всяка от частите му q, L и r сама по себе си е прост път. Не е проблем, ако този пешеходец повтори връх; проблем е, че може да се озове в t предсрочно (възможно е L да минава през t), а тогава предписаните ходове не могат да бъдат изпълнени. Но в такъв случай той просто спира да се движи, тоест не се изпълняват цялата стъпка № 7, цялата или част от стъпка № 5 и евентуално някаква част от стъпка № 3. Това не е пречка за движението на другия пешеходец (стъпки № 4 и № 6), защото съответните участъци от p не съдържат върха t, тъй като p е прост път. Така в крайна сметка пешеходците успяват да се разминат.

Втори случай: Всяко ребро от пътя p е мост и съществува връх u от p със следните свойства: $u \neq s$, $u \neq t$, степента на u е поне 3. Нека v е съсед на u, различен от върха непосредствено преди u и върха непосредствено след u при обхождане на p. В състояние сме да изберем върха v по указания начин, защото графът не съдържа кратни ребра; и $v \neq u$, понеже в графа няма примки. Освен това $v \notin p$, иначе p не би бил най-къс път.

Нека q е участъкът от p между s и u, а r е участъкът от p между u и t.



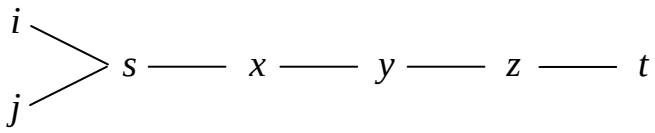
В този случай пешеходците се разминават по следния начин:

1) Пешеходецът от s върви по участъка q; като стигне в u, се отбива във v. Понеже p е прост път и $u \neq t$, то следва, че $t \notin q$. А щом $v \notin p$, то $v \neq t$. Следователно при това движение пешеходецът от s не минава през t, тоест двамата пешеходци не се срещат.
2) Пешеходецът от t отива в s по пътя p, без да се отбива. Тъй като $v \notin p$, то пешеходците не се срещат.
3) Пешеходецът от s отива от върха v във върха u по свързващото ги ребро и оттам по участъка r отива в t. Понеже p е прост път и $u \neq s$, то $s \notin r$. А щом $v \notin p$, то $v \neq s$. Ето защо при това движение пешеходецът от s не минава през s, тоест не се среща с другия пешеходец.

Тъй като p е прост път, p съдържа най-много $n - 2$ върха, различни от s и t, затова обхождането на върховете на p изисква време $O(n)$. Понеже търсенето се прекратява, когато намери u и v, а всички върхове от p, обходени преди тях, са от степен 2, то обходените ребра са не повече от $2(n - 2) + 1$, т.е. също $O(n)$. Ето защо времевата сложност на този етап от алгоритъма е от порядък $O(n)$, а времевата сложност на целия алгоритъм остава линейна при всякакви данни.

Трети случай: Всички върхове от p, различни от s и t, са от втора степен и всяко ребро от p е мост. В този случай пешеходците не могат да се разминат. Проверката на върховете и ребрата на p (след края на алгоритъма на Шмит) се извършва за време $O(n)$, така че времевата сложност на целия алгоритъм е линейна при всякакви входни данни.

Важността на изискването, пешеходците да престанат да се движат, когато достигнат крайната си цел, тук проличава ясно. Без това изискване пешеходците понякога успяват да се разминат даже в последния случай. Показаният граф попада именно в третия разгледан случай, но разминаването все пак е възможно: пешеходецът от s отива в i, пешеходецът от t отива в j, пешеходецът от s отива от i в t, а пешеходецът от t отива от j в s.



Ако отпадне изискването за неподвижност след достигане на крайната цел, ще остане единствен (и тривиален) случай, когато разминаването е невъзможно: компонентата на свързаност на s и t е прост път, т.е. всички нейни върхове имат степен < 3. Търсенето на мостовете отпада и задачата става безинтересна.

Задача 2. Понеже покриващите дървета имат еднакъв брой ребра ($n - 1$), то белите ребра са най-много, когато черните са най-малко. С други думи, търсим минимално покриващо дърво: черните ребра имат тегло 1, белите — 0. За целта използваме алгоритъма на Прим—Ярник или алгоритъма на Крускал. Техните времеви сложности не са линейни, обаче и при двата алгоритъма можем да постигнем значително ускорение.

При алгоритъма на Прим—Ярник няма нужда от приоритетна опашка: стигат ни три списъка от върхове (със стойности нула, единица и безкрайност). Използваме двусвързани списъци, за да можем лесно да трием посочен елемент (можем да минем и с едносвързани списъци, но с цената на изкуствен трик: когато искаме да посочим елемент от списък, трябва да използваме указател не към желания от нас елемент, а към неговия предшественик в списъка). В началото на алгоритъма слагаме всички върхове в списъка "безкрайност" с изключение на един връх (избран произволно) — той отива в списъка "нула" и става начален връх за алгоритъма; списъкът "единица" е празен отначало.

Извличаме произволен връх u от списъка "нула"; в случай че той е празен, извличаме произволен връх u от списъка "единица"; ако и този списък е празен, алгоритъмът приключва работа: успешно, тоест намерил е покриващо дърво с минимален брой черни ребра — ако списъкът "безкрайност" е празен също; неуспешно (няма покриващо дърво) — ако списъкът "безкрайност" не е празен. Последният случай се отнася само за несвързан граф и е изключен по условие.

Извлеченият връх u става текущ връх. Обхождат се всички ребра $\{u\ ;\ v\}$. Ако върхът v е затворен, т.е. присъединен е към дървото, той не се обработва. (Отначало няма затворени върхове.) Иначе правим опит за релаксация на v:
— ако v е в списъка "нула", релаксация не е възможна;
— ако v е в списъка "единица" и реброто $\{u\ ;\ v\}$ е бяло, преместваме върха v в списъка "нула" и караме v да запомни реброто $\{u\ ;\ v\}$ (всеки връх помни по едно входящо ребро, чрез което ще се присъедини към нарастващото дърво);
— ако v е в списъка "безкрайност", преместваме v в някой от другите списъци: в списъка "нула" — ако $\{u\ ;\ v\}$ е бяло ребро; в "единица" — ако е черно; и в двата случая караме v да запомни реброто $\{u\ ;\ v\}$.

След това обявяваме u за затворен връх и го добавяме към дървото заедно с реброто, запомнено от u. Началният връх не притежава такова ребро. Отначало дървото е празно. Нараства постепенно, започвайки от началния връх и присъединявайки все нови и нови върхове и ребра. Накрая се получава минимално покриващо дърво или съобщение, че такова дърво няма.

За да знае всеки връх в кой списък се намира, пазим още един масив от състоянията на всички върхове:

0 — върхът се намира в списъка "нула";

1 — върхът се намира в списъка "единица";

2 — върхът се намира в списъка "безкрайност";

състояние -1 означава, че върхът е затворен (присъединен към дървото).

Псевдокод на алгоритъма:

```
ALG(Adj[1...n]) // Върхове са целите числа 1, 2, ... , n.
 // Adj[u] е списък на ребрата, излизащи от върха u;
 // всяко ребро пази другия връх и цвят (бял или черен).
 1) π[1...n]: предшествениците на върховете в дървото.
 2) σ[1...n]: състояния на върховете.
 3) L0, L1, L2: двусвързани списъци от върхове.
 4) L0 ← ∅
 5) L1 ← ∅
 6) L2 ← ∅
 7) for v ← 2 to n do
 8)    π[v] ← 0
 9)    σ[v] ← 2
10)    добави v към L2
11) π[1] ← 0
12) σ[1] ← 0
13) добави 1 към L0
14) while L0 ≠ ∅ or L1 ≠ ∅ do
15)    if L0 ≠ ∅
16)       u ← първия връх от L0
17)       премахни u от L0
18)    else
19)       u ← първия връх от L1
20)       премахни u от L1
21)    σ[u] ← -1 // Затваряне на върха u.
22)    for v ∈ Adj[u] do
23)       if σ[v] > {u; v}.colour // 0 — бяло; 1 — черно
24)          π[v] ← u
25)          премахни v от Lσ[v]
26)          σ[v] ← {u; v}.colour
27)          добави v към Lσ[v]
28) if L2 ≠ ∅
29)    return NULL // Няма покриващо дърво.
30) return π[1...n] // Има покриващо дърво.
```

Анализ на времевата сложност на променения алгоритъм на Прим—Ярник: С всеки връх на графа се извършват следните операции: добавяне към списък в началото на алгоритъма, най-много две релаксации (от ∞ към 1 и от 1 към 0), извличане от списък като текущ връх и затваряне — между три и пет операции за всеки отделен връх на графа. За всички върхове на графа: $\Theta(n)$ операции. Освен това всяко ребро се обработва най-много два пъти — по веднъж за всеки от краищата си, когато стават текущи върхове. За всички ребра: $\Theta(m)$ операции. Цялото време на алгоритъма е от порядък $\Theta(m + n)$ при всякакви входни данни. Този алгоритъм отговаря на всички изисквания и носи пълен брой точки: 50 %.

По принцип задачата може да бъде решена и с алгоритъма на Крускал, но желаният порядък е достижим само приблизително. Сортирането отпада: просто разделяме ребрата по цвят. Това изисква време $\Theta(m)$, не $\Theta(m \log m)$. Така ускоряваме първия, по-бавния етап от алгоритъма. Не е ясно дали и как можем да ускорим втория етап. Разделянето на ребрата по цвят не ни помага при проверката дали дадено ребро затваря цикъл. Не е известен по-бърз начин от използването на структурата Union-Find, а тя изразходва време $\Theta(m\,\alpha(n))$. Теоретично погледнато, то не е линейно: функцията $\alpha(n)$ расте неограничено. Но тя расте толкова бавно, че $\alpha(n) < 5$ за всички практически възможни n. Тоест това решение е практически бързо, затова се оценява с 25 %.

ПОДРОБНА СХЕМА ЗА ТОЧКУВАНЕ

Задача 1 носи общо 50 %, разпределени по следния начин:
— за подробно описание на алгоритъма с приложени примери: 25 %;
— за подробно доказателство на коректността на алгоритъма: 20 %;
— за анализ на времевата сложност на алгоритъма: 5 %.

Не носят точки алгоритми, които са неясно описани или некоректни, или недостатъчно бързи. Липсата на обосновка не е пречка за точкуването на останалите етапи.

Задача 2 носи общо 50 %, ако предложеният алгоритъм е описан подробно, коректен е и има времева сложност $\Theta(m + n)$ при най-лоши входни данни. Алгоритъм, чиято времева сложност при най-лоши входни данни е $O(m\,\alpha(n))$, носи 25 %. Ако алгоритъмът е по-бавен или неясно описан, или некоректен, или необоснован — не се дават точки.