Примерни решение на задачите от изпита по Дизайн и Анализ на Алгоритми, КН2, 25.06.2016 г.

Зад. 1 Даден е неориентиран свързан тегловен граф $G = (V, E)$ с тегловна функция $w : E \to \mathbb{R}$. Известно е, че $\forall e_1, e_2 \in E : e_1 \neq e_2 \to w(e_1) \neq w(e_2)$. Докажете или опровергайте всяко от следните две твърдения.

5 т.
- Съществува едно единствено минимално покриващо дърво на G.

5 т.
- Ако c е цикъл в G и e' е най-тежкото ребро в c, то всяко минимално покриващо дърво на G не съдържа e'.

Решение. И двете твърдения са верни.

Непълно, но приемливо доказателство на първото твърдение е следното. Да разгледаме алгоритъма на Крускал за МПД с вход G. Щом теглата на ребрата са уникални, то сортираната наредба от ребрата по тегла е уникална; тъй като алгоритъмът на Крускал "прескача" точно ребрата, които биха образували цикли с досега сложените ребра—а това е еднозначно определено от структурата на G и не зависи от теглата—то построения изход-дърво T е уникално (при тези тегла няма друг възможен изход). Знаем, че T е МПД, защото алгоритъмът на Крускал е коректен. Нещо повече. Друго МПД няма, защото, ако имаше, то трябваше да има ребро, по-тежко от най-тежкото ребро на T, но за да компенсира това, то трябваше да има и поне още едно ребро, което е леко (по-леко от второто най-тежкото ребро на T) и не се намира в T.

По-формално доказателство на първото твърдение е: да допуснем, че има две различни МПД-та T' и T''. Тъй като множествата от върховете им са еднакви, те трябва да се различават в поне едно ребро. Нека e е най-лекото ребро, което се среща в едното дърво, но не се среща в другото (при уникални тегла, такова ребро има и е единствено). БОО, нека e се намира в T'. Тогава добавянето на e към T'' образува уницикличен граф U. Нека c е цикълът в U. Но c съдържа поне едно ребро $\tilde{e}$, което не се намира в T'. Тъй като e се намира в T', то $e \neq \tilde{e}$. Забелязваме, че $w(e) < w(\tilde{e})$, а освен това премахването на $\tilde{e}$ от U образува покриващо дърво $\tilde{T}$ с тегло $w(T'') + w(e) - w(\tilde{e})$. Но $w(e) < w(\tilde{e})$, така че $w(\tilde{T}) < w(T'')$, в противоречие с допускането, че T'' е МПД.

Да разгледаме второто твърдение и да допуснем, че не е вярно. Да допуснем, че има МПД T, което съдържа e'. Нека $e' = (x, y)$. Премахването на e' от T води до това, че се образува гора от две дървета T' и T'', едното от които съдържа връх x, а другото, връх y. Очевидно $\{V(T'), V(T'')\}$ е разбиване на V. Да разгледаме това разбиване като срез (cut) в G. Реброто e' прекосява среза, но има поне още едно ребро от c, да го наречем e'', което също прекосява среза. Но $w(e'') < w(e')$ по конструкция, така че $T + e'' - e'$ е ПД на G с тегло, по-малко от теглото на T, в противоречие с допускането, че T е МПД.

15 т. **Зад. 2** Нека $A = \{a_1, a_2, \ldots, a_t\}$ е множество. Нека в някакъв начален момент е дадено $\mathcal{B} = \{\{a_1\}, \{a_2\}, \ldots, \{a_t\}\}$, което е разбиването на A на едноелементни подмножества. Нека всеки елемент на $\mathcal{B}$ е реализиран чрез кореново дърво така, както беше показано на лекции в темата за Union-Find структури. Започва произволно изпълнение на операции Union, всяка от които слива два елемента-множества на $\mathcal{B}$ в едно множество, както беше показано на лекции: указателят на корена на едното дърво се насочва към корена на другото дърво. $\mathcal{B}$ се променя, но остава разбиване на A.

Всяка от Union операциите се извършва съгласно Union-by-rank евристиката, където рангът е височината на дървото – точно както беше на лекции. Докажете, че след произволно изпълнение (в произволен ред) на операции Union, за всяко дърво T измежду дърветата, които представят полученото разбиване на A, е вярно, че височината на T е $O(\lg n)$, където n е броят на върховете на T (а НЕ мощността на A).

Решение. Това е теорема, доказана на лекции. Ето доказателство по индукция по височината на дървото. Подробно написано, твърдението е, че има константа $c > 0$, такава че $h(T) \leq c \lg n$ за

всяко дърво T, получено по този начин, където $h(T)$ е неговата височина, а n е броят на неговите върхове.

База: $h(T) = 0$. Тогава $n = 1$. Наистина, за всяка положителна константа, желаното неравенство е изпълнено.

Индуктивно предположение: Да допуснем, че някакви дървета T_1 и T_2 са получени по указания начин. Нека те имат съответно n_1 и n_2 върхове. Допускането е, че $h(T_1) \leq c \lg n_1$ и $h(T_2) \leq c \lg n_2$, за някаква положителна константа c.

Индуктивна стъпка: Изпълняваме операцията UNION върху T_1 и T_2 съгласно UNION-BY-RANK евристиката и получаваме дърво T с n върха. Очевидно, $n = n_1 + n_2$. Ако $h(T_1) \neq h(T_2)$, то $h(T) = \max\{h(T_1), h(T_2)\}$ и неравенството се запазва.

Нека $h(T_1) = h(T_2)$. Нека $h(T_1) = h$. Тогава, $h(T) = h + 1$ заради начина, по който работи операцията UNION. БОО, нека $n_1 \geq n_2$. От индуктивното предположение имаме

$$h \leq c \lg n_1$$
$$h \leq c \lg n_2$$

Искаме да покажем, че

$$h + 1 \leq c \lg n$$

Действително,

$$c \lg n = c \lg (n_1 + n_2) \geq c \lg (n_2 + n_2) = c \lg 2n_2 = c \lg n_2 + c \lg 2 = c \lg n_2 + c \geq \quad \text{(съгл. инд. предп.)}$$
$$h + c \geq h + 1 \quad \text{(за всяко c, по-голямо или равно на 1)}$$

25 т. **Зад. 3** Дадена е комуникационна мрежа, която се състои от станции $\{T_1, T_2, \ldots, T_n\}$. За всяка станция T_i са дадени една или повече други станции, с които T_i може да *комуникира директно*. Комуникацията е двупосочна: ако T_j може да комуникира директно с T_k, то и T_k може да комуникира директно с T_j. Казваме, че T_j може да *комуникира индиректно* с T_k, ако T_j не може да комуникира директно с T_k, но съществува поредица от междинни станции, такава че T_j може да комуникира директно с първата от тях, първата, с втората, и така нататък, а последната станция от поредицата може да комуникира директно с T_k. Важно ограничение е, че в поредицата от междинни станции, всички те са уникални. С други думи, при индиректна комуникация не може да има повтарящи се междинни станции.

Знае се, че всеки две станции може да комуникират или директно, или индиректно. Освен това се знае, че за всеки две станции, които може да комуникират индиректно, съществува една единствена поредица от междинни станции, чрез която да стане индиректната комуникация.

Организацията X иска да провали комуникационната мрежа като повреди една или повече станции от нея, така че всякаква комуникация да стане невъзможна. Организацията може да повреди произволно подмножество от станции, но това струва пари: за да бъде повредена станция T_i, организацията трябва да плати цена S_i. Цената за повреждането на някакво подмножество станции е просто сумата от цените за повреждането на всяка една от тях. Организацията иска да минимизира разхода за проваляването на мрежата, така че целта ѝ не е да изгаси всички станции, а само някакво подмножество станции, такива, че след тяхното излизане от строя да не е възможна никаква директна комуникация.

Дадени са станциите $T_1, T_2, \ldots, T_n$ и цените S_1, S_2, …, S_n за повреждането им. За всяка станция е дадено с кои други станции може да комуникира директно. Предложете алгоритъм със сложност по време $O(n \lg n)$, който изчислява минималното число m, такова че съществува подмножество от станции, които може да бъдат повредени на обща цена m и чието повреждане води до това, че измежду останалите станции нито две не могат да комуникират директно.

Решение. Очевидно става дума за неориентиран свързан граф без цикли—с други думи, дърво—чиито върхове са станциите, а ребрата са директните комуникации. Дървото е тегловно с тегла на върховете (а не на ребрата). Теглата, очевидно, са цените S_i. Задачата, която трябва да решим, е VERTEX COVER, но в претеглен вариант: да се намери множество върхове с минимална сумарна

цена, чието изтриване води до това, че не остават никакви ребра. За простота се иска само числено решение: само сумарното тегло на минимално покриващо множество, а не самото множество.

Задачата се решава лесно с алгоритъм, изграден по схемата динамично програмиране. Нека дървото е D. Правим дървото кореново с корен произволен връх r. За всеки връх i, нека $D\langle i\rangle$ означава поддървото, вкоренено във връх i. Очевидно, $D\langle r\rangle$ е самото D. За всеки връх i поддържаме две стойности, да ги наречем $A[i]$ и $B[i]$. $A[i]$ има смисъл на сумата на теглата на минимално върхово покриване на $D\langle i\rangle$, в което връх i **не участва**. $B[i]$ има смисъл на сумата на теглата на минимално върхово покриване на $D\langle i\rangle$, в което връх i **участва**. Нека $A[1,\ldots,n]$ и $B[1,\ldots,n]$ да са масиви. Алгоритъмът работи върху кореновото дърво отдолу нагоре (bottom-up), като за всяко не-листо i изчислява $A[i]$ и $B[i]$, използвайки A-стойностите и B-стойностите на децата на i, както и цената S_i на i. Ако i е листо, $A[i]$ и $B[i]$ се пресмятат тривиално чрез S_i, както е показано по-долу.

Търсеният отговор очевидно е $\min\{A[r], B[r]\}$, тъй като той участва или не участва във върхово покритие с минимално сумарно тегло.

Работата отдолу нагоре в кореновото дърво може да се реализира лесно чрез едно единствено пускане на DFS от връх r. За всяко не-листо i, нека $\text{children}(i)$ е множеството от децата на i. Математически, следната система от взаимно рекурентни уравнения описва всичко, което ни трябва, за да решим задачата:

$$A[i] = 0, \qquad \text{ако } i \text{ е листо.} \tag{1}$$

$$B[i] = S_i, \qquad \text{ако } i \text{ е листо.} \tag{2}$$

$$A[i] = \sum_{u\in\text{children}(i)} B[u], \qquad \text{ако } i \text{ не е листо.} \tag{3}$$

$$B[i] = S_i + \sum_{u\in\text{children}(i)} \min\{A[u], B[u]\}, \qquad \text{ако } i \text{ не е листо.} \tag{4}$$

Обосновката е елементарна. Началните условия 1 и 2 са очевидни. Ако връх i не принадлежи на минималното върхово покритие на $D\langle i\rangle$, то задължително всички негови деца трябва да са в покритието – в противен случай, поне едно ребро между i и някое негово дете би останало непокрито. Това съображение обосновава случай 3. От друга страна, ако i участва в покритието, то ребрата между него и децата му са "обгрижени" и за всяко дете може да вземем по-доброто (тоест, с по-малкото сумарно тегло) върхово покритие на съответното вкоренено поддърво. Това съображение обосновава случай 4.

Разполагайки с рекурентните уравнения, тривиално е да се построи модифициран DFS, който попълва A- и B-стойностите на всеки връх в момента, в който финализира този връх. Сложността на DFS е $\Theta(n+m)$, но понеже става дума за дърво, то $m = n-1$, така че сложността е $\Theta(n)$.

25 т. **Зад. 4** *Палиндром* се нарича всеки стринг, който "се чете" по същия начин отдясно наляво. Формално, нека е дадена крайна азбука Σ и някакъв $x \in \Sigma^*$. Нека $x = x_1x_2\cdots x_n$ за някое $n \geq 0$, където $x_i \in \Sigma$ за $1 \leq i \leq n$. x е палиндром тогава и само тогава, когато $x = x_nx_{n-1}\cdots x_1$.

Предложете алгоритъм, който по зададен стринг $x \in \{a,b\}^*$ намира най-дългия палиндром, който е подстринг на x. Вашият алгоритъм трябва да работи във време $O(n^2)$.

Решение. Нека $x = x_1x_2\cdots x_n$. Решението с брутална сила се основава на следното елементарно наблюдение: всеки палиндром има начало и край; формално, всеки палиндром е $x_ix_{i+1}\cdots x_j$ за някакви стойности на i и j, такива че $1 \leq i \leq j \leq n$. Всеки символ сам по себе си е палиндром, така че има смисъл да се проверява за $1 \leq i < j \leq n$ Достатъчно е за всички $\Theta(n^2)$ възможности за индексите i и j да проверим дали $x_ix_{i+1}\cdots x_j$ е палиндром. Една такава проверка обаче би отнела време $\Theta(j-i)$. Ако направим точна сметка, излиза, че подходът с брутална сила води до кубичен алгоритъм. А се иска не по-лош от квадратичен, в асимптотичния смисъл.

Ще използваме алгоритмичната схема динамично програмиране. Нека $P(i,j)$ е двуместен предикат, такъв че

$$P(i,j) = \begin{cases} 1, & \text{ако } 1 \leq i \leq j \leq n \text{ и } x_ix_{i+1}\cdots x_j \text{ е палиндром,} \\ 0, & \text{ако } 1 \leq i \leq j \leq n \text{ и } x_ix_{i+1}\cdots x_j \text{ не е палиндром,} \\ \textit{недефиниран}, & \text{ако } \neg(1 \leq i \leq j \leq n) \end{cases}$$

Ще ползваме таблица $P[1\ldots n, 1\ldots n]$, чрез която ще изчислим всички стойности на този предикат. Под главния диагонал стойностите са недефинирани. За останалата част на таблицата ползваме следните математически съображения:

$$P(i,i) = 1, \qquad \text{за } 1 \le i \le n \tag{5}$$

$$P(i,i+1) = 1 \leftrightarrow x_i = x_{i+1}, \qquad \text{за } 1 \le i \le n-1 \tag{6}$$

$$P(i,j) = 1 \leftrightarrow x_i = x_j \wedge P(i+1,j-1), \text{ за } 1 \le i < j-1 \le n \tag{7}$$

Ако съобразим, че управляващият параметър на рекурсията е разликата $j-i$, тези съображения се транслират директно в код:

```
ALG PAL(x_1 x_2 ... x_n ∈ Σ*)
 1  for i ← 1 to n
 2      P[i, i] ← 1
 3  for i ← 1 to n − 1
 4      if x_i = x_{i+1}
 5          P[i, i + 1] ← 1
 6      else
 7          P[i, i + 1] ← 0
 8  for diag ← 3 to n
 9      for i ← 1 to n − diag + 1
10          if x_i = x_{i+diag−1} ∧ P[i + 1, i + diag − 2] = 1
11              P[i, i + diag − 1] ← 1
12          else
13              P[i, i + diag − 1] ← 0
```

След като напълним таблицата по този начин, търсим максимално отдалечен диагонал (от главния диагонал), на който има единица – такава единица отговаря на максимален палиндром.

```
PRINT PAL(P[1...n, 1...n])
1  for diag ← n to 1
2      for i ← 0 to n − diag
3          if P[i, i + diag − 1]
4              return x_i x_{i+1} ... x_{i+diag−1}
```

Както при повечето алгоритми по схемата динамично програмиране, коректността на алгоритъма следва (почти) веднага от коректността на рекурсията, в случая (5), (6), (7).

15 т. **Зад. 5** Изчислителната задача XYZ се дефинира така:

Изчислителна Задача XYZ
Общ пример: Масиви $A[1,\ldots,n]$ и $B[1,\ldots,n]$ от уникални естествени числа.
Решение: Пермутация на елементите на $B[]$, такава че:

- най-малкият елемент на $B[]$ се намира на същата позиция, на която е най-малкият елемент на $A[]$,
- вторият най-малък елемент на $B[]$ се намира на същата позиция, на която е вторият най-малък елемент на $A[]$,
- и така нататък ...,
- най-големият елемент на $B[]$ се намира на същата позиция, на която е най-големият елемент на $A[]$.

Подчертава се, че елементите на $A[]$ не се местят – местят се само елементите на $B[]$. Докажете, че $\Omega(n \lg n)$ е асимптотична долна граница за задачата XYZ.

Решение. Задачата Сортиране се свежда до XYZ в смисъл, че е неин частен случай: ако $A[]$ е $[1, 2, \ldots, n]$, то да се пренареди $B[]$ по указания начин е същото като $B[]$ да се сортира. А ние знаем, че Сортиране има асимптотична долна граница $\Omega(n \lg n)$.

20 т. **Зад. 6** Изчислителната задача Зад1 се дефинира така:

Изчислителна Задача Зад1
Общ пример: Множество A, множество $\mathcal{B} \subseteq 2^A$, число $k \in \mathbb{N}$
Въпрос: Съществува ли $C \subseteq A$, такова че $|C| \leq k$ и $\forall X \in \mathcal{B} : X \cap C \neq \emptyset$? □

Докажете, че Зад1 е **NP**-пълна.

Решение. В теорията тази задача е известна като Hitting Set.

Това, че Hitting Set $\in$ **NP** е очевидно: недетерминираната машина на Тюринг отгатва подмножество C и проверява, че $|C| \leq k$ и че C има поне един общ елемент с всяко множество от $\mathcal{B}$. Също така очевидно е, че това може да стане в полиномиално време.

Ще покажем, че Vertex Cover $\propto$ Hitting Set. Нека е даден пример на Vertex Cover: неориентиран граф $G = (V, E)$ и естествено число k'. Построяваме пример на Hitting Set, в който $A \leftarrow V$, $\mathcal{B} \leftarrow E$, и $k \leftarrow k'$. Очевидно е, че графът да има върхово покритие с мощност не по-голяма от k' е същото като A да има подмножество с не повече от k елемента, такова че всяко (двуелементно) множество в $\mathcal{B}$ да има сечение с него. Конструкцията очевидно може да се извърши в полиномиално време.