# Примерни решения на изпита по ДАА на Информатика и КН 2 поток, проведен на 06 юли 2023 г.

**Зад. 1** Докажете или опровергайте следното твърдение.

$$\frac{2^{\lceil \lg n \rceil + \lfloor \lg n \rfloor}}{n} = \Theta(n)$$

Направете колкото е възможно по-прецизна аргументация, тръгвайки от дефиницията на "$\Theta(n)$", без да използвате наготово леми и теореми, изучавани в курса ДАА.

**Решение:** Твърдението е вярно. Ще докажем, че

$$\frac{2^{\lceil \lg n \rceil + \lfloor \lg n \rfloor}}{n} = \Theta(n)$$

По определение, това е същото като да съществуват положителни константи $c_1$ и $c_2$ и положителна стойност $n_0$ на аргумента, такава че $\forall n \geq n_0$:

$$c_1 n \leq \frac{2^{\lceil \lg n \rceil + \lfloor \lg n \rfloor}}{n} \leq c_2 n \tag{1}$$

Тъй като $n > 0$, (1) са еквивалентни на

$$c_1 n^2 \leq 2^{\lceil \lg n \rceil + \lfloor \lg n \rfloor} \leq c_2 n^2$$

Логаритмуваме неравенствата при основа 2 и получаваме еквивалентните

$$\lg c_1 + 2 \lg n \leq \lceil \lg n \rceil + \lfloor \lg n \rfloor \leq \lg c_2 + 2 \lg n \tag{2}$$

Да разгледаме

$$\lg c_1 + 2 \lg n \leq \lceil \lg n \rceil + \lfloor \lg n \rfloor$$

Това неравенство е в сила за, примерно, $c_1 = \frac{1}{4}$ и $\forall n \geq 2$. Наистина, $\forall n \geq 2$ е в сила

$$\begin{aligned} \lg n &\leq \lceil \lg n \rceil \\ -2 + \lg n &\leq \lfloor \lg n \rfloor \end{aligned}$$

Като съберем тези неравенства, получаваме

$$-2 + 2 \lg n \leq \lceil \lg n \rceil + \lfloor \lg n \rfloor \;\leftrightarrow\; \lg \frac{1}{4} + 2 \lg n \leq \lceil \lg n \rceil + \lfloor \lg n \rfloor$$

Доказахме, че за $c_1 = \frac{1}{4}$ и $\forall n \geq 2$:

$$\lg c_1 + 2 \lg n \leq \lceil \lg n \rceil + \lfloor \lg n \rfloor$$

Да разгледаме

$$\lceil \lg n \rceil + \lfloor \lg n \rfloor \leq \lg c_2 + 2 \lg n$$

Това неравенство е в сила за, примерно, $c_2 = 4$ и $\forall n \geq 2$. Наистина, $\forall n \geq 2$ е в сила

$$\begin{aligned} \lceil \lg n \rceil &\leq 2 + \lg n \\ \lfloor \lg n \rfloor &\leq \lg n \end{aligned}$$

Като съберем тези две неравенства, получаваме

$$\lceil \lg n \rceil + \lfloor \lg n \rfloor \leq 2 + 2 \lg n \;\leftrightarrow\; \lceil \lg n \rceil + \lfloor \lg n \rfloor \leq \lg 4 + 2 \lg n$$

Доказахме, че за $c_2 = 4$ и $\forall n \geq 2$:

$$\lceil \lg n \rceil + \lfloor \lg n \rfloor \leq \lg c_2 + 2 \lg n$$

Заключаваме, че за $c_1 = \frac{1}{4}$, $c_2 = 4$ и $\forall n \geq 2$:

$$\lg c_1 + 2 \lg n \leq \lceil \lg n \rceil + \lfloor \lg n \rfloor \leq \lg c_2 + 2 \lg n$$

Но това е същото като (2). Заключаваме, че

$$\frac{2^{\lceil \lg n \rceil + \lfloor \lg n \rfloor}}{n} = \Theta(n)$$

**Зад. 2** Нека AlgX е алгоритъм, базиран на сравнения, който получава като вход наредена двойка $(A, k)$, където $A$ е множество от числа и $k \in \{1, \ldots, |A|\}$, и който връща като изход $k$-тото по големина число в $A$. Докажете, че AlgX може да бъде модифициран така, че да освен това да връща всички $k-1$ на брой най-малки елементи и всички $|A|-k$ на брой най-големи елементи на $A$, без да извършва други сравнения на числа от $A$ освен тези, които е извършил, за да намери $k$-ия по големина елемент.

**Решение** Нека $|A| = n$. Нека $k$-ият по големина елемент бъде наречен $x$. Нека

$$S = \{a \in A \mid a < x\}$$
$$L = \{a \in A \mid a > x\}$$

Очевидно $|S| = k - 1$ и $|L| = n - k$. На прост български, иска се да се докаже, че информацията, получена в процеса на идентифицирането на $x$, е достатъчна, за да бъдат идентифицирани освен това и множествата $S$ и $L$.

Може да добавим код към AlgX, който строи ориентиран граф $G$, чиито върхове са елементите на $A$, а ребро $(A[i], A[j])$ се слага тстк е било извършено сравнение на $A[i]$ с $A[j]$, резултатът от което е $A[i] < A[j]$. Очевидно $G$ е даг[†] – това следва веднага от антисиметричността на релацията $<$. Забелязваме, че построяването на $G$ не иска допълнителни сравнения освен тези, които се ползват за намирането на $x$.

Твърдим, че в момента, в който $x$ вече е намерен и AlgX терминира, връщайки $x$, е вярно, че

- за всеки $y \in S$, в $G$ съществува път от $y$ до $x$,
- за всеки $y \in L$, в $G$ съществува път от $x$ до $y$.

БОО, ще докажем само първото от тези твърдения. Да допуснем противното: съществува $y \in S$, такъв че в $G$ няма път от $y$ до $x$. Спрямо $y$ конструираме множеството $Y$[‡]

$$Y = \{a \in A \mid \text{ в } G \text{ съществува път от } y \text{ до } a\}$$

Правим аргументация с противник. Установявайки наличието на $y$, който принадлежи на $S$, но от него до $x$ няма път, противникът увеличава стойностите на всички върхове от $Y$ по такъв начин, че

- стойността на $y$ да стане по-голяма от стойността на $x$,
- резултатите от извършените сравнения да останат същите и
- елементите на $A$ да останат уникални.

При това $x$ престава да е $k$-ият по големина елемент; по този начин противникът опровергава коректността на AlgX.

И така, по отношение на $G$, $x$ е достижим от всеки връх на $S$ и всеки връх на $L$ е достижим от $x$ след приключването на сравненията. Модифицирането на AlgX включва както построяването на $G$, така и топо-сортиране на $G$. Очевидно е, че в топо-сортирането, върховете преди $x$ са точно върховете от $S$, а върховете след $x$ са точно върховете от $L$. Тривиално е да се добави код към AlgX, който сканира елементите на $A$ в реда на топо-сортирането и генерира $S$, $x$ и $L$, и връща $(S, x, L)$, както се иска в задачата.

[†] $G$ е дагът на сравненията, разгледан на лекции.

[‡] $Y$ се състои точно от върховете, достижими от $y$. Забележете, че $Y$ е непразно, защото съдържа $y$.

**Зад. 3** Формулирайте и докажете теоремата на Cook.

**Решение:** Това е правено на лекции.

**Зад. 4** Ориентиран граф се нарича *полусвързан*, ако за всеки два негови върха $u$ и $v$ е вярно, че съществува път от $u$ до $v$ или съществува път от $v$ до $u$. Предложете колкото можете по-ефикасен алгоритъм, който изчислява дали ориентиран граф е полусвързан. Докажете коректността му и изследвайте сложността му по време. Не е необходимо да пишете псевдокод. Достатъчно е да опишете алгоритъма като идея, но тази идея трябва да е описана кристално ясно и недвусмислено – по такъв начин, че от нея да може да се напише псевдокод директно. Доказателството за коректност трябва да е формално прецизно.

**Решение:** Нека е даден произволен ориентиран граф $G = (V, E)$. Ето как може да намерим ефикасно дали е $G$ полусвързан. Първо, използвайки алгоритъма на Kosaraju, намираме силно свързаните компоненти на $G$ и оттам конструираме фактор-графа $G/SC(G)$, където $SC(G)$ е релацията на силна свързаност върху $V$. Второ, топо-сортираме $G/SC(G)$. Трето, проверяваме дали за всеки връх в това топо-сортиране има ребро към съседа вдясно. Ако това е изпълнено, връщаме ДА, в противен случай връщаме НЕ.

**Факт 1:** $G$ е полусвързан тогава и само тогава, когато фактор-графът му е полусвързан. Нека силно свързаните компоненти на $G$ са $G_1, \ldots, G_t$.

- В едната посока, допускаме, че $G$ е полусвързан. Ще докажем, че фактор-графът е полусвързан. Върховете на фактор-графа са силно свързаните компоненти на $G$. Разглеждаме произволни силно свързани компоненти $G_i$ и $G_j$. Ще докажем, че във фактор-графа има път от $G_i$ до $G_j$ или от $G_i$ до $G_j$.

  Нека $u$ е прозволен връх от $G_i$. Нека $v$ е произволен връх от $G_j$. Щом $G$ е полусвързан, в него има път от $u$ до $v$ или от $v$ до $u$. БОО, нека има път $p$ от $u$ до $v$.

  Забелязваме, че $p$ се явява конкатенация, в този ред, на подпътища $p_{s_1}, \ldots, p_{s_k}$, които се намират съответно в силно свързани компоненти $G_{b_1}, \ldots, G_{b_k}$, две по две различни, като $G_i = G_{b_1}$ и $G_j = G_{b_k}$. Причината е, че ако $p$ веднъж "излезе" от някоя силно свързана компонента, той не може повече да се "върне" в нея. Очевидно е, че подпътищата $p_{s_1}, \ldots, p_{s_k}$, в този ред, задават път във фактор-графа от $G_i$ до $G_j$.

- В другата посока, нека фактор-графът е полусвързан. Ще докажем, че $G$ е полусвързан. Нека $u$ и $v$ са произволни върхове в $G$. Ще докажем, че в $G$ има път от $u$ до $v$ или от $v$ до $u$.

  Ако $u$ и $v$ са от една и съща силно свързана компонента на $G$, твърдението е очевидно вярно. Нека $u$ и $v$ са съответно от силно свързани компоненти $G_i$ и $G_j$, където $i \neq j$. Щом фактор-графът е полусвързан, в него има път от $G_i$ до $G_j$ или от $G_j$ до $G_i$. БОО, нека има път от $G_i$ до $G_j$. Тогава очевидно в $G$ има път от $u$ до $v$.

**Факт 2:** Нека $D = (\{a_1, a_2, \ldots, a_k\}, E_D)$ е произволен даг. $D$ е полусвързан тогава и само тогава, когато $D$ има Хамилтонов път.

- В едната посока, ако $D$ има Хамилтонов път $p = a_{i_1}, a_{i_2}, \ldots, a_{i_k}$, очевидно за всеки два върха $a_{i_j}$ и $a_{i_\ell}$, където $i_j < i_\ell$, е вярно, че има път от $a_{i_j}$ и $a_{i_\ell}$ или от $a_{i_\ell}$ до $a_{i_j}$; а именно, подпътят на $p$ от $a_{i_j}$ до $a_{i_\ell}$.

- В другата посока, да допуснем, че $D$ е полусвързан. Щом $D$ е даг, $D$ има поне едно топо-сортиране $t = [a_{i_1}, a_{i_2}, \ldots, a_{i_k}]$. Твърдим, че $t$ представлява Хамилтонов път в $D$ в смисъл, че $(a_{i_j}, a_{i_{j+1}}) \in E_D$ за $j \in \{1, \ldots, k-1\}$. Наистина, ако $(a_{i_j}, a_{i_{j+1}}) \notin E_D$ за поне едно $j \in \{1, \ldots, k-1\}$, очевидно няма път нито от $a_{i_j}$ до $a_{i_{j+1}}$, нито от $a_{i_{j+1}}$ до $a_{i_j}$, в противоречие с това, че $D$ е полусвързан.

**Факт 3:** Даг има Хамилтонов цикъл тстк има точно едно топо-сортиране тстк в произволно негово топо-сортиране има ребро от всеки връх, без последния, към съседа вдясно. Това е доказвано на лекции.

**Факт 1**, **Факт 2** и **Факт 3**, плюс факта, че фактор-графът на $G$ е даг (това е доказвано на лекции!) са достатъчна обосновка на коректността на алгоритъма.

Сега да разгледаме сложността по време. Твърдим, че гореспоменатата идея може да се реализира чрез алгоритъм, линеен в размера на $G$. Допускаме, че $G$ е реализиран чрез списъци на съседство; ако $G$ е реализиран с матрица на съседство, няма как да постигнем сложност $\Theta(m + n)$ винаги.

Алгоритъмът на Kosaraju има сложност $\Theta(n+m)$, както знаем от лекции. Имплементацията, която видяхме на лекции, връща разбиване на $V$, което отговаря на силно свързаните компоненти в смисъл, че дяловете на разбиването индуцират точно силно свързаните компоненти. Сега искаме да получим фактор-графа като отделен граф със свои върхове и ребра. За краткост ще наричаме фактор-графа $H$. Върховете на $H$ може да бъдат получени много лесно: във второто викане на SCC, а именно $\mathrm{SCC}(G^T, L)$ (ред 3 в алгоритъма на Kosaraju), във втория **for**-цикъл на основната функция SCC, на всяка итерация създаваме нов връх с идентификатор $i$ (нов връх на $H$) и след това маркираме с $i$ всички върхове на $G$, които викането на SCC-REC посещава, като по този начин запомняме, че това са точно върховете, принадлежащи са силно свързаната компонента с име $i$. По отношение на конструирането на $H$, създаваме нов списък, отговарящ на връх $i$.

По-триково е ефикасното конструиране на ребрата на $H$. Ако след приключване на алгоритъма на Kosaraju сканираме списъка на всеки връх на $G$ (това означава едно пълно минаване през списъците на съседство на $G$, което е същото като разглеждането на всички ребра на $G$), ще разпознаем важните ребра по това, че началото и краят са от различни силно свързани компоненти (вече маркирахме всеки връх на $G$ с името на силно свързаната компонента, на която принадлежи). А важните ребра на $G$ ни дават ребрата на $H$. И така, за всеки връх $u \in V$ сканираме списъка на съседство на $u$ и за всеки връх $v$ в този списък, ако $i$ е името на силно свързаната компонента на $G$, от която е $u$, и $j$ е името на силно свързаната компонента на $G$, от която е $v$:

- ако $i = j$, прескачаме $v$,
- ако $i \neq j$, в списъка на $i$ (в представянето на $H$) вкарваме връх $j$.

По този начин $H$ в общия случай е мултиграф, защото е възможно за две силно свързани компоненти $G_i$ и $G_j$ на $G$ да има много ребра от връх в $G_i$ към връх в $G_j$. Току-що описаната процедура ще сложи по едно ребро от $i$ към $j$ в $H$ за всяко от тях. Забелязваме, че размерът на това представяне на $H$ е $O(n+m)$. По този начин получаваме $H$, реализиран със списъци, във време $\Theta(n+m)$.

Това, че $H$ е мултиграф, не е проблем. Топологическото сортиране на $H$ се състои в едно пускане на DFS и записване на върховете по времената на финализиране в намаляващ ред. DFS работи без проблеми върху мултиграфи: за всеки сноп паралелни ребра, ако DFS използва едно от тях за откриване на нов (бял) връх, останалите няма да се използват за тази цел, защото при опитите за тяхното "прекосяване", въпросният връх вече няма да е бял. Топо-сортирането на $H$ става във време $O(n+m)$, защото, както вече видяхме, размерът на представянето на $H$ е $O(n+m)$.

Остава да видим дали по отношение на топо-сортирането на $H$ има ребро от всеки връх, без последния, към съседа вдясно. Това става като сканираме топо-сортирането отляво надясно и за всеки връх $u$ в него без последния, ако $v$ е върхът непосредствено вдясно от $u$, сканираме списъка на $u$ и, ако открием $v$, продължаваме със сканирането на топо-сортирането, а ако не открием $v$, връщаме отговор НЕ. Ако стигнем до края на топо-сортирането без да сме върнали НЕ, връщаме ДА. В най-лошия случай ще извършим работа, пропорционална на сумарната големина на списъците на $H$. Ерго, тази фаза на алгоритъма се изпълнява във време $O(n+m)$.

Като цяло, сложността на алгоритъма е $\Theta(n+m)$.

**Зад. 5** Дадена е булева матрица $M$ с $m$ реда и $n$ колони. За всяка нейна подматрица ще казваме, че е *единична*, ако съдържа само единици. Предложете алгоритъм със сложност $O(mn)$, който намира **квадратна** единична подматрица $M'_{k\times k}$, като това $k$ е максимално. Вашият алгоритъм трябва да върне наредена тройка $(i, j, k)$, където $M[i, j]$ е клетката на $M'$, намираща се долу вдясно. Обосновете коректността и сложността по време на Вашия алгоритъм.

**Решение:** Ето решение по схемата **Динамично Програмиране**. Даден е булев масив $M[1\,..\,m, 1\,..\,n]$. Конструираме масив $R[1\,..\,m, 1\,..\,n]$, чиито елементи са естествени числа със следния смисъл: за $i \in \{1, \ldots, m\}$ и $j \in \{1, \ldots, n\}$, $R[i, j]$ е дължината на страната на максималната единична квадратна подматрица, чиято долна дясна клетка е $M[i, j]$. $R[i, j] = 0$ означава, че такава квадратна подматрица няма.

Очевидно най-горният ред на $R$ просто повтаря най-горния ред на $M$: ако $M[1, j] = 0$, няма единична квадратна подматрица на $M$, чиято долна дясна клетка е $M[1, j]$; ако $M[1, j] = 1$, максималната единична квадратна подматрица, чиято долна дясна клетка е $M[1, j]$, е самата $M[1, j]$ със страна единица. Напълно аналогично, най-лявата колона на $R$ просто повтаря най-лявата колона на $M$.

Нека $i \in \{2, \ldots, m\}$ и $j \in \{2, \ldots, n\}$.

- Ако $M[i, j] = 0$, то страната на максималната единична квадратна подматрица, чиято долна дясна клетка е $M[i, j]$, е нула – такава няма. Поради това, ако $M[i, j] = 0$, то $R[i, j] = 0$.
- Нека $M[i, j] = 1$.
  - Ако $M[i, j-1] = 0$ или $M[i-1, j] = 0$ или $M[i-1, j-1] = 0$, то клетка $M[i, j]$ не "разширява" единична квадратна подматрица в посока надолу-надясно, така че $R[i, j] = 1$.
  - Нека $M[i, j-1] = M[i-1, j] = M[i-1, j-1] = 1$. Нека $M'$ е максималната единична квадратна подматрица, чиято долна дясна клетка е $M[i, j-1]$. Нека $M''$ е максималната единична квадратна подматрица, чиято долна дясна клетка е $M[i-1, j]$. Нека $M'$ е $k' \times k'$, а $M''$ е $k'' \times k''$.
    * Ако $k' \neq k''$, нека $k = \min\{k', k''\}$. Тогава максималната единична квадратна подматрица, чиято долна дясна клетка е $M[i, j]$, има страна $1 + k$. В този подслучай клетка $M[i-k, j-k]$ съдържа единица.
    * Ако $k' = k'' = k$, то клетка $M[i-k, j-k]$ съдържа нула. Тогава максималната единична квадратна подматрица, чиято долна дясна клетка е $M[i, j]$, има страна $k$; можем да мислим, че тя се явява $M'$, транслирана с една позиция надясно, или, алтернативно, $M''$, транслирана с една позиция надолу.
      Щом клетка $M[i-k, j-k]$ съдържа нула, максималната единична квадратна подматрица, чиято долна дясна клетка е $M[i-1, j-1]$, е с размери $(k-1) \times (k-1)$. Имаме право да кажем, че страната на максималната единична квадратна подматрица, чиято долна дясна клетка е $M[i, j]$ е равна на страната на максималната единична квадратна подматрица, чиято долна дясна клетка е $M[i-1, j-1]$, плюс едно.

  Всичко това може да се обобщи така: ако $M[i, j] = M[i, j-1] = M[i-1, j] = M[i-1, j-1] = 1$, то $R[i, j] = 1 + \min(R[i, j-1], R[i-1, j], R[i-1, j-1])$.

Следният алгоритъм реализира тази рекурсивна декомпозиция.

```
MaxSquareUnitSubmatrix(M[1 .. m, 1 .. n]: boolean)
 1  създай R[1 .. m, 1 .. n] от тип естествени числа
 2  for j ← 1 to n
 3      R[1, j] ← M[1, j]
 4  for i ← 1 to m
 5      R[i, 1] ← M[i, 1]
 6  for j ← 2 to n
 7      for i ← 2 to m
 8          if M[i, j] = 0
 9              R[i, j] ← 0
10          else if M[i, j − 1] = 0 or M[i, j − 1] = 0 or M[i − 1, j − 1] = 0
11              R[i, j] ← 1
12          else
13              R[i, j] ← 1 + min(R[i, j − 1], R[i − 1, j], R[i − 1, j − 1])
14  return enhanced-max(R[1 .. m, 1 .. n])
```

Функцията enhanced-max връща наредена тройка $(i, j, k)$, където $k$ е стойността на максимален елемент на $R$, а $i$ и $j$ са неговите координати в $R$.

Коректността на алгоритъма е очевидна предвид коректността на рекурсивната декомпозиция, а сложността е очевидно $O(mn)$, понеже enhanced-max може да се реализира така, че да работи във време $O(mn)$.

**Зад. 6** Напишете алгоритъма за намиране на най-къси пътища в дагове, изучаван на лекции, и докажете коректността му. Дискутирайте предимствата на този алгоритъм пред алгоритъма на Dijkstra.

**Решение:** Това е правено на лекции.