Примерни решения на изпита по ДАА на КН2 & Инф, проведен на 19.06.2019 г.

Зад. 1 Дадени са n болта, нито два от които не са с еднакъв диаметър. Дадени са и n гайки, като за всеки болт има точно една гайка, която му пасва. Не е разрешено да се сравняват болт с болт или гайка с гайка. Разрешено е да се правят само сравнения от вида ***Сравни болт*** `i` ***с гайка*** `j`, като резултатът от такова сравнение е точно едно от следните три: или ***Болт*** `i` ***има по-голям диаметър от гайка*** `j`, или ***Болт*** `i` ***има по-малък диаметър от гайка*** `j`, или ***Болт*** `i` ***пасва точно на гайка*** `j`.

10 т. **а)** Докажете, че всеки алгоритъм, който намира кой болт на коя гайка съответства, използвайки само сравнения от посочения вид, работи във време $\Omega(n \lg n)$.

10 т. **б)** Предложете алгоритъм, който намира кой болт на коя гайка съответства, използвайки само сравнения от посочения вид, и има **средна** сложност по време $O(n \lg n)$. Можете да опишете алгоритъма общо: не с псевдокод, а само да посочите идеята, използвайки наготово факти, които са изучавани на лекции.

20 т. **в)** Предложете итеративен алгоритъм, който намира най-малкия болт и най-малката гайка, използвайки не повече от $2n-2$ сравнения от посочения вид. Единственият вид достъп, който Вашият алгоритъм може да прави до болтовете и гайките, е викане на функция `compare(B[i], N[j])`, където `B[i]` е `i`-ият болт, а `N[j]` е `j`-тата гайка. Тази функция връща

- 1, ако `i`-ият болт е с по-голям диаметър от `j`-тата гайка,
- -1, ако `i`-ият болт е с по-малък диаметър от `j`-тата гайка,
- 0, ако `i`-ият болт пасва точно на `j`-тата гайка.

Този алгоритъм трябва да бъде написан подробно в псевдокод.

Решение: а) Всеки болт пасва на точно една гайка по условие, а от елементарни физически съображения следва, че и всяка гайка пасва на точно един болт. Ерго, пасването болтове-гайки е биекция и всеки алгоритъм за тази задача трябва да установи биекция между множеството на болтовете и множеството на гайките. Тези биекции са точно $n!$. Следователно, всяко дърво на вземане на решения за тази задача има поне $n!$ листа. Въпросите, които е разрешено да се задават, имат по три отговора, следователно дървото има разклоненост 3. Оттук височината на дървото е поне $\log_3(n!)$. В асимптотична нотация, височината е $\Omega(n \lg n)$, използвайки добре известния резултат за асимптотиката на логаритъм от факториел.

б) Задачата лесно може да се реши от алгоритъм със средна сложност по време $O(n \lg n)$, ако използваме идеята на Quicksort. Идеята на Quicksort (ако няма еднакви елементи) е да разбием множеството спрямо някакъв елемент, наречен pivot, на множеството от елементите, по-големи от pivot, и множеството от елементите, по-малки от pivot, и да работим рекурсивно върху двете подмножества. Това обаче може да се приложи директно само ако можем да сравняваме всеки два елемента. В тази задача елементите са болтове и гайки и не е разрешено да се сравнява болт с болт или гайка с гайка. Така че леко усложняваме идеята: б.о.о., избираме произволен болт b и спрямо него разбиваме множеството от гайките на три подмножества: множеството S от гайките, които са с по-малък диаметър от b, гайката (тя е точно една) γ, която му пасва, и множеството от гайките L, които са с по-голям диаметър от b. Това правим в линейно време с пробване на всяка гайка с b.

Това обаче не ни дава веднага желаното разбиване, защото множеството от болтовете в момента не е разбито на три, а само на две: $\{b\}$ и останалите болтове. Вземаме γ и спрямо нея разбиваме множеството от останалите болтове на множеството A от тези, които имат по-малък диаметър от γ, и множеството B от тези, които имат по-голям диаметър от γ. Очевидно $|S| = |A|$ и $|L| = |B|$. Нещо повече, съществува биективно съответствие-пасване между болтовете от A и гайките от S, и също така има биективно съответствие-пасване между болтовете от B и гайките от L. Тогава $\langle S, A\rangle$ е пример на същата задача, но с по-малък размер; също така и $\langle L, B\rangle$ е пример с по-малък размер. Ако $|S|$ е достатъчно голямо, решаваме рекурсивно задачата върху $\langle S, A\rangle$, в противен случай решаваме задачата с брутална сила в константно време. Аналогично, ако $|L|$ е достатъчно голямо, решаваме рекурсивно върху $\langle L, B\rangle$, в противен случай решаваме задачата с брутална сила в константно време.

Очевидно, асимптотичния анализ е точно като QUICKSORT с функцията PARTITION. В най-лошия случай, т.е. когато на всяко ниво на рекурсията имаме лошия късмет да изберем екстремален болт, сложността е квадратична; в най-добрия случай е $\Theta(n \lg n)$; в средния случай е пак $\Theta(n \lg n)$, което може да се покаже със същото рекурентно уравнение, което използвахме за средната сложност на QUICKSORT. Това, че правим два PARTITION-а, веднъж на гайките спрямо произволен болт и после на болтовете срещу неговата гайка, няма значение за асимптотиката.

в) Идеята е такава: разглеждаме някаква произволна наредба на болтовете, да кажем, масив от болтовете $B[1, \ldots, n]$, и някаква произволна наредба на гайките, да кажем, масив от гайките $N[1, \ldots, n]$. За целите на задачата, и болтовете, и гайките се идентифицират с диаметрите си. Б.о.о., нека диаметрите са цели положителни числа. Сравняваме $B[1]$ с $N[1]$. Ако $B[1]$ е по-голям, отхвърляме го като невъзможен (той не може да е най-малкия болт) и пробваме с $B[2]$ и $N[1]$, и така нататък. Ако $N[1]$ е по-голяма, отхвърляме я и пробваме с $B[1]$ и $N[2]$ и така нататък. Ако $B[1]$ пасва на $N[1]$, не можем да отхвърлим тази двойка, но все още не сме сигурни, че те са най-малките. Запомняме тази двойка като текущ кандидат и минаваме нататък, тоест разглеждаме следващия болт и следващата гайка; ако няма следващ болт или следваща гайка, то край на алгоритъма. Със сигурност ще намерим поне една такава пасваща двойка, но може да има много такива. Следващата двойка пасващи болт и гайка, която ще намерим, ако изобщо има следваща, или ще е от по-малки болт и гайка, при което тя (следващата) ще стане текущата, или ще е от по-големи болт и гайка, при което ще я прескочим. Двойката, която е останала накрая, е минималната. Следният псевдокод реализира тази идея.

```
BOLTS-NUTS(B[1,...,n], N[1,...,n]: posint)
 1  Done ← FALSE, First-Occur ← TRUE
 2  i ← 1, j ← 1
 3  while not Done do
 4      if i < n or j < n
 5          switch compare(B[i], N[j])
 6              case 1:
 7                  if i < n
 8                      i++
 9                  else
10                      Done ← TRUE
11                  break
12              case −1:
13                  if j < n
14                      j++
15                  else
16                      Done ← TRUE
17                  break
18              case 0:
19                  if First-Occur
20                      i* ← i, j* ← j, First-Occur ← FALSE
21                  else
22                      if compare(B[i], N[j*]) = −1
23                          i* ← i, j* ← j
24                  i++, j++
25                  if i = n + 1 or j = n + 1
26                      Done ← TRUE
27                  break
28      else (* i = n  or j = n *)
29          if First-Occur
30              i* ← i, j* ← j
31          else
32              if compare(B[i], N[j*]) = −1
33                  i* ← i, j* ← j
34          Done ← TRUE
35  return ⟨i*, j*⟩
```

Зад. 2 Разгледайте задачата за намиране на най-къс път в тегловен ориентиран граф с положителни тегла на ребрата. Разгледайте задачата във варианта от даден връх s до всички останали върхове, като за най-късите пътища се искат само теглата им (а не самите пътища).

1 т. **а)** Напишете алгоритъма на Dijkstra за тази задача.

19 т. **б)** Докажете коректността на този алгоритъм.

Решение Това се преподава на лекции.

Зад. 3 Представете си лабиринт, който се състои от стаи и коридори между стаите. Коридорите са еднопосочни: във всеки коридор може да се върви само в едната посока и никога в обратната посока. Знае се, че в този лабиринт е невъзможно да се въртим в кръг; тоест, ако напуснем дадена стая през някакъв коридор, излизащ от нея, няма как да се върнем в нея повече. Даден е лабиринтът като множеството от стаите $\{r_1, r_2, \ldots, r_n\}$ и множеството от коридорите, като всеки коридор е наредена двойка (r_i, r_j) от началната си и крайната си стая.

Предложете ефикасен алгоритъм, който по дадени стаи r_p и r_q връща броя на различните начини да тръгнем от стая r_p и да пристигнем в стая r_q, движейки се в лабиринта. Обосновете накратко коректността и сложността по време.

Решение: Задачата може да се моделира с ориентиран граф, чиито върхове са стаите, а коридорите са ребрата. Невъзможността да се въртим в кръг казва, че графът е даг. Трябва да направим ефикасен алгоритъм, който намира броя на ориентираните пътища от даден връх s до друг даден връх t в дага.

```
NUM-PATHS-DAG(G = (V, E) : dag, s, t ∈ V)
1  for x ∈ V
2      num[x] ← 0
3  num[s] ← 1
4  TOPOSORT(G)
5  for x ∈ V in the topo order
6      for y ∈ Adj[x]
7          num[y] ← num[y] + num[x]
8  return num[t]
```

Коректността може да се обоснове чрез следната рекурсия. Нека $n(x)$ означава броя на пътищата от s до x.

$$n(x) = \begin{cases} 1, & \text{ако } x = s \\ \sum\limits_{(y,x)\in E} n(y), & \text{в противен случай} \end{cases}$$

Щом няма цикли, очевидно има точно един път от s до s. За всеки друг връх x, броят на пътищата от s до x е сумата по всички предшественици y на x на броевете на пътищата от s до y.

Сложността по време е $\Theta(n + m)$. От лекции знаем, че сложността на топологичното сортиране е $\Theta(n+m)$, а цикълът на редове 5–7 също има сложност $\Theta(n+m)$, защото той минава през всеки списък на съседства по веднъж.

Зад. 4 Контекстно-свободна граматика (КСГ) е наредена четворка $\Gamma = \langle \Sigma, N, S, R \rangle$, където Σ е крайно множество от терминали (Σ е азбуката), N е крайно множество от нетерминали, такова че $N \cap \Sigma = \emptyset$, $S \in N$ е фиксиран нетерминал, наречен начален нетерминал, и R е множеството от правила (още се наричат продукции) от вида $A \to X$, където A е нетерминал, а $X \in (\Sigma \cup N)^*$. Формално, R е бинарна релация $R \subseteq N \times (\Sigma \cup N)^*$. Дефинираме бинарната релация $\Rightarrow \subseteq (\Sigma \cup N)^* \times (\Sigma \cup N)^*$ така:

$$x \Rightarrow z \overset{\text{def}}{\leftrightarrow} \exists A \in N \; \exists x_1, x_2 \in (\Sigma \cup N)^* \; (x = x_1 A x_2 \text{ и има правило } A \to \alpha \text{ и } z = x_1 \alpha x_2)$$

Релацията $\overset{*}{\Rightarrow}$ е рефлексивното и транзитивно затваряне на $\Rightarrow$. Ако $x \overset{*}{\Rightarrow} y$, казваме, че има *деривация от x до y в Γ*. За всеки $y \in \Sigma^*$ казваме, че y *има деривация в Γ*, ако $S \overset{*}{\Rightarrow} y$.

КСГ е в Нормална Форма на Чомски (пишем кратко КСГНФЧ), ако всички правила са от вида $A \to BC$ или $A \to a$ или $S \to \epsilon$, където A, B и C са нетерминали, a е терминал, а ϵ е празният стринг.

Предложете ефикасен алгоритъм, който по дадена КСГНФЧ $\Gamma = \langle \Sigma, N, S, R \rangle$ и $y \in \Sigma^*$ връща 1, ако y има деривация в Γ, или 0 в противен случай. Дайте съвсем кратка обосновка на коректността и сложността по време на предложения от Вас алгоритъм. Приемете, че размерът на Γ е $O(1)$.

Решение: Ще построим алгоритъм по схемата динамично програмиране. Този алгоритъм е добре известен и се нарича CYK на имената на създателите си Cocke, Younger и Kasami.

Нека $y = y_1 y_2 \cdots y_n$. Б.о.о., нека $n \geq 1$. Ако за всеки $y_i \cdots y_j$, където $1 \leq i \leq j \leq n$, намерим множеството $X_{i,j} = \{A \in N \mid A \overset{*}{\Rightarrow} y_i \cdots y_j\}$, то ние сме готови! Тогава връщаме 1, ако $S \in X_{1,n}$, и връщаме 0 в противен случай. Това, което позволява да направим ефикасен алгоритъм е, че при $i < j$ можем да пресмятаме ефикасно всяко $X_{i,j}$ от множествата $X_{i',j'}$, където $i \leq i' \leq j' \leq j$ и $j' - i' < j - i$.

Базата на рекурсията са множествата $X_{i,i}$ за $1 \leq i \leq n$:

$$\forall i \in \{1, 2, \ldots, n\} : X_{i,i} = \{A \in N \mid \text{има правило } A \to y_i\}$$

защото в КСГНФЧ, ако по време на деривацията в сентенциалната форма се появи нетерминал, той никога не изчезва в смисъл, че в крайния стринг ще има поне един съответстващ на него терминал; ерго, правилата от вида $A \to BC$ никога не могат да стартират деривация на еднобуквен стринг.

При $i < j$, $X_{i,j}$ е обединението, по всички нетривиални факторизации $\langle y_i \cdots y_k, y_{k+1} \cdots y_j \rangle$ на стринга $y_i \cdots y_j$ (тоест, по всички $k \in \{i, \ldots, j-1\}$) на тези нетерминали, които са A-то в продукции от вида $A \to BC$, такива че $B \overset{*}{\Rightarrow} y_i \cdots y_k$ и $C \overset{*}{\Rightarrow} y_{k+1} \cdots y_j$. Но тези нетерминали B и C са елементите на множествата $X_{i,k}$ и $X_{k+1,j}$. При изчисляването на $X_{i,j}$ множествата $X_{i,k}$ и $X_{k+1,j}$ са вече известни.

Самият алгоритъм може да се реализира по начин, аналогичен на реализацията на алгоритъма за Matrix-Chain Multiplication, изучаван на лекции.

```
Derivation(Γ = ⟨Σ, N, S, R⟩, y ∈ Σ*)
 1  n ← |y|
 2  for i ← 1 to n
 3      M[i, i] ← {A | (A → y_i) ∈ R}
 4  for diag ← 2 to n
 5      for i ← 1 to n − diag + 1
 6          j ← i + diag − 1
 7          M[i, j] ← ∅
 8          for k ← i to j − 1
 9              for B ∈ M[i, k]
10                  for C ∈ M[k + 1, j]
11                      for P ∈ R
12                          if P = (A → BC)
13                              M[i, j] ← M[i, j] ∪ {A}
14  if S ∈ M[1, n]
15      return 1
16  else
17      return 0
```

Обосновката на коректността следва от обосновката на рекурсията. Сложността по време е $\Theta(n^3)$, ако граматиката има константна сложност – тогава множествата $M[a, b]$ имат големини, ограничени от константа, така че и трите най-вътрешни цикъла "въртят" само константен брой пъти.

Зад. 5 Докажете, че следната задача е **NP**-пълна. Дадено е множество A от софтуерни приложения. Дадено и е множество C от "способности" (capabilities), като всяко приложение притежава една или повече от тези способности. Дадено е и число k. Пита се дали има подмножество $X \subseteq A$, такова че обединението от способностите на всички приложения в X е точно C и $|X| \leq k$.

Решение: Задачата може да се моделира математически по два начина. Даденото условие позволява и двете интерпретации. Да разгледаме следните изчислителни задачи. "Фамилия над S" означава множество от подмножества на S.

Изчислителна задача Set Cover
Общ пример: Опорно множество S, фамилия $\mathcal{F}$ над S и естествено число k.
Въпрос: Дали съществува $\mathcal{F}' \subseteq \mathcal{F}$, такова че $\forall x \in S\, \exists Y \in \mathcal{F}' : x \in Y$ и $|\mathcal{F}'| \leq k$?

Изчислителна задача Hitting Set
Общ пример: Опорно множество S, фамилия $\mathcal{F}$ над S и естествено число k.
Въпрос: Дали съществува $S' \subseteq S$, такова че $\forall Y \in \mathcal{F}\, \exists x \in S' : x \in Y$ и $|S'| \leq k$?

Тук задачите нарочно са формулирани по начин, който подчертава, че те са дуални една на друга. Типично, изискването за $\mathcal{F}'$ в Set Cover се записва така "$\bigcup \mathcal{F}' = S$", а за S' в Hitting Set, така "$\forall Y \in \mathcal{F} : Y \cap S' \neq \emptyset$".

Да помислим как тези задачи може да моделират **Зад. 5**. Една възможна интерпретация е, че множеството от приложенията A е опорното множество, а всяка способност е множеството от приложенията, които я имат, така че множеството от способностите C е фамилия над A. При тази интерпретация **Зад. 5** е Hitting Set: иска се минимално множество от приложения, такова че всяка способност съдържа поне едно приложение от него[†].

Възможна е и друга интерпретация обаче. При нея, опорното множество е множеството от способностите C, а всяко приложение е множеството от своите способности, така че множеството от приложенията A е фамилия над C. При тази интерпретация **Зад. 5** е Set Cover: иска се минимално множество от приложения, които в своята съвкупност имат всички способности[‡].

Set Cover и Hitting Set са дуални в следния смисъл. Ако мислим и за приложенията, и за способностите като за протоелементи, тогава отношението между тях се описва от двуделен граф, чиито върхове са приложенията и способностите (единият дял са приложенията, а другият, способностите), а ребро между приложение и способност има точно тогава, когато това приложение има тази способност[§]. Разговорно казано, ако гледаме този двуделен граф откъм приложенията (те да са първичното нещо), виждаме задачата Hitting Set, а ако го гледаме откъм способностите, виждаме задачата Set Cover. Независимо откъде гледаме двуделния граф, пита се едно и също нещо: дали има подмножество на единия дял с мощност $\leq k$, такова всеки връх от другия дял да е съсед на някой връх от това него. Това е задачата на фундаментално ниво. Тя става

- Hitting Set, ако дялът, в който търсим такова подмножество, е опорно множество, а другият дял е фамилия над него, или
- Set Cover, ако дялът, в който търсим такова подмножество, е фамилия над опорно множество, което е другият дял.

Ще покажем поотделно, че Hitting Set и Set Cover са **NP**-пълни.

[†]Тук се допуска, че няма две способности, които имат едни и същи приложения. Ако това не е изпълнено, способностите не са множество, а мултимножество. В задачата никъде не е казано, че не може да има две способности с едни и същи приложения, но нашата цел е да покажем алгоритмична неподатливост. Ако я покажем за ограничения вариант, в който способностите са множество, очевидно по-общия вариант, в който са мултимножество, също е алгоритмично неподатлив.

[‡]Тук се допуска, че няма две приложения, които имат едни и същи способности. Ако това не е изпълнено, приложенията не са множество, а мултимножество. В задачата никъде не е казано, че не може да има две приложения с едни и същи приложения, но нашата цел е да покажем алгоритмична неподатливост. Ако я покажем за ограничения вариант, в който приложенията са множество, очевидно по-общия вариант, в който са мултимножество, също е алгоритмично неподатлив.

[§]В контекста на този двуделен граф не ни интересува дали има две приложения с едни и същи способности или две способности се притежават от едни и същи приложения.

Първо ще покажем, че HS $\in$ **NP**-с. Очевидно HS $\in$ **NP**: има недетерминирана машина на Тюринг, която отгатва множество S' и после в полиномиално време проверява, че $S' \subseteq S$, $|S'| \leq k$ и всеки елемент на $\mathcal{F}$ съдържа някой елемент на S'.

Ще покажем, че Vertex Cover $\propto$ Hitting Set. При даден пример $\langle G = (V, E), k\rangle$ на VC, правим в полиномиално време пример $\langle S, \mathcal{F}, k'\rangle$ на HS, в който $k' = k$, $S = V$ и $\mathcal{F} = E$. Очевидно е, че конструкцията може да се извърши в полиномиално време.

Коректността на конструкцията е също очевидна. Vertex Cover е частен случай на Hitting Set! Ако мислим за ребрата на графа като за двуелементни множества, то Vertex Cover е Hitting Set, при ограничението, че всеки елемент на $\mathcal{F}$ има точно два елемента.

Сега ще покажем, че SC $\in$ **NP**-с. Очевидно SC $\in$ **NP**: има недетерминирана машина на Тюринг, която отгатва подмножество $\mathcal{F}'$ и после в полиномиално време проверява, че $\mathcal{F}' \subseteq \mathcal{F}$, $|\mathcal{F}'| \leq k$ и $\mathcal{F}'$ покрива S.

Ще покажем, че Vertex Cover $\propto$ Set Cover. При даден пример $\langle G = (V, E), k\rangle$ на VC, правим в полиномиално време пример $\langle S, \mathcal{F}, q\rangle$ на SC, в който $q = k$, $S = E$ и $\mathcal{F} = \{\mathcal{I}(v) \,|\, v \in V\}$, където "$\mathcal{I}(v)$" означава множеството от ребрата на G, инцидентни с връх v.

Ако $\langle G = (V, E), k\rangle$ е ДА-пример на VC, то по дефиниция има $U \subseteq V$, такова че на всяко ребро от E поне единият край е в U и $|U| \leq k$. Всеки връх от U задава точно едно множество-елемент на $\mathcal{F}$. За съвкупността от тези множества, да я наречем $\mathcal{F}'$, е вярно, че всеки елемент на S е в някое от тях; с други думи, $\mathcal{F}'$ покрива S. Освен това, $|\mathcal{F}'| = k = q$.

В обратната посока, ако $\langle S, \mathcal{F}, q\rangle$ е ДА-пример на SC, то има подмножество $\mathcal{F}' \subseteq \mathcal{F}$, такова че $\mathcal{F}'$ покрива S и $|\mathcal{F}'| = q$. В примера на VC, на $\mathcal{F}'$ съответстват $q = k$ различни множества от ребра, като във всяко от тях ребрата имат общ край. С други думи, $\mathcal{F}'$ задава върхово покриване с размер не повече от k на графа.

Очевидно е, че конструкцията може да се извърши в полиномиално време.