Примерни решения на изпита по ДАА на КН2, проведен на 23.06.2018 г.

Зад. 1 Даден е неориентиран свързан тегловен граф $G = (V, E)$ с тегловна функция $w : E \to \mathbb{N}$, такава че $\forall e \in E \, \forall e' \in E : e \neq e' \to w(e) \neq w(e')$. Конструирайте ефикасен алгоритъм, който получава като вход графа G, тегловната функция w и някакво ребро $e \in E$, и връща:

- ДА, ако съществува минимално покриващо дърво на G, което съдържа e;
- НЕ, в противен случай.

Обосновете кратко, но съдържателно коректността и сложността по време на Вашия алгоритъм.

Решение: Построяваме граф $G' = (V, E')$, където E' са всички ребра от E с тегла, по-малки от $w(e)$, където e е реброто от входа. Това очевидно може да стане във време $O(|V| + |E|)$. След това намираме свързаните компоненти на G' във време $O(|V| + |E'|)$. Нека краищата на реброто e са върхове u и v. Ако u и v са в различни свързани компоненти на G', връщаме ДА, в противен случай връщаме НЕ. Коректността следва (почти) директно от МПД-теоремата. Сложността по време е $O(|V| + |E|)$.

Зад. 2 Разгледайте следните две задачи за разпознаване.

Изчислителна задача ЗадачаX
Общ пример: Масив $A[1 \dots n]$ от цели числа
Въпрос: Дали има индекси i, j, k, такива че $1 \leq i \leq j \leq k \leq n$ и $A[i] + A[j] + A[k] = 0$?

Изчислителна задача ЗадачаY
Общ пример: Масиви $B[1 \dots n]$ и $C[1 \dots n]$ от цели числа
Въпрос: Дали има индекси i, j, k, такива че $1 \leq i \leq j \leq n$, $1 \leq k \leq n$ и $B[i] + B[j] = C[k]$?

Нека е даден алгоритъм AlgY за ЗадачаY. Конструирайте алгоритъм AlgX за ЗадачаX, който:

- първо извършва някаква работа във време $O(n)$,
- след това прави точно едно извикване на AlgY с вход, който той (AlgX) вече е конструирал,
- ако AlgY върне НЕ, връща НЕ; ако AlgY върне ДА, връща ДА.

Обосновете съвсем кратко коректността и сложността по време на предложения от Вас алгоритъм AlgX. Подчертава се, че алгоритъмът AlgY е **даден**, така че не се иска нито да го конструирате, нито да го анализирате; можете да мислите за него като за "черна кутия", която работи коректно и само за единица време. Това, което трябва да конструирате, е подходящ вход за AlgY.

Решение

```
AlgX(A[1...n]: int)
1  създаваме масиви B[1...n] и C[1...n]
2  for i ← 1 to n
3      B[i] ← A[i]
4      C[i] ← −A[i]
5  if AlgY(B, C) = ДА
6      return ДА
7  else
8      return НЕ
```

Аргументацията за коректност е тривиална: просто забелязваме, че въпросът "Дали има индекси i, j, k, такива че $1 \leq i \leq j \leq k \leq n$ и $A[i]+A[j]+A[k] = 0$?" от дефиницията на ЗадачаX, която е известна под името 3SUM, може да се препише като "Дали има индекси i, j, k, не непременно различни, такива че $A[i] + A[j] = -A[k]$?". Работата преди викането на AlgY очевидно е линейна и също така очевидно връщаме точно това, което връща AlgY, така че сме конструрали редукция с желаните свойства.

Зад. 3 Нека $a_1\ a_2\ \cdots\ a_n$ е пермутация на множеството $\{1, 2, \ldots, n\}$. *Аномалия* в тази пермутация се нарича всяка наредена двойка индекси (i, j), такава че $1 \le i < j \le n$ и $a_i > a_j$.

Конструирайте ефикасен алгоритъм, който изчислява броя на аномалиите в дадена пермутация $a_1\ a_2\ \cdots\ a_n$ на множеството $\{1, 2, \ldots, n\}$. Обосновете кратко, но съдържателно коректността и сложността по време на Вашия алгоритъм.

Решение: Задачата бе решена в час чрез малка модификация на MergeSort, запазваща сложността по време $\Theta(n \lg n)$. Единствената разлика е, че се говореше за "инверсии", а не за "аномалии", но смисълът е абсолютно същият.

Зад. 4 Дадена е азбука $\{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$. Дадени са два непразни стринга $\mathtt{X} = x_1 \cdots x_n$ и $\mathtt{Y} = y_1 \cdots y_m$ над тази азбука. Даден и още един символ, който ще наричаме *шпация* и ще записваме така: $\sqcup$. Шпацията може да бъде поставяна във всеки от $\mathtt{X}$ или $\mathtt{Y}$, колкото пъти искаме и където искаме. Резултатът от нула или повече поставяния на $\sqcup$ между символите на $\mathtt{X}$ се нарича *попълване на* $\mathtt{X}$, и съответно дефинираме *попълване на* $\mathtt{Y}$. Например, ако $\mathtt{X} = \mathtt{ACCT}$, попълвания на $\mathtt{X}$ са стринговете $\mathtt{A}\sqcup\mathtt{C}\sqcup\mathtt{CT}$, $\sqcup\sqcup\mathtt{ACCT}\sqcup$ и $\mathtt{ACCT}$. Подчертаваме, че оригиналните символи на $\mathtt{X}$ се запазват – попълването с $\sqcup$ не трие нищо.

Подравняване на $\mathtt{X}$ *и* $\mathtt{Y}$ е всяко попълване на всеки от $\mathtt{X}$ и $\mathtt{Y}$ с нула или повече шпации по такъв начин, че дължините на двата попълнени стринга да са равни. Разглеждаме само такива подравнявания, в които на всяка позиция $i \in \{1, \ldots, N\}$ във всеки от двата попълнени стринга, поне единият символ не е $\sqcup$, където N е дължината на всеки от попълнените стрингове. Нека $\mathtt{W}$ е стрингът, получен от $\mathtt{X}$, а $\mathtt{Z}$ е стрингът, получен от $\mathtt{Y}$, и нека N е дължината на всеки от $\mathtt{W}$ и $\mathtt{Z}$. Тогава, за всяко $i \in \{1, \ldots, N\}$, правим следната *оценка на позиция* i:

- ако на i-та позиция в $\mathtt{W}$ и $\mathtt{Z}$ се намира един и същи символ (който не може да е шпация съгласно правилата за подравняване), оценката на тази позиция е $+2$.
- ако на i-та позиция в $\mathtt{W}$ и $\mathtt{Z}$ се намират различни символи, и двата различни от шпация, оценката на тази позиция е -1.
- ако на i-та позиция в $\mathtt{W}$ и $\mathtt{Z}$ се намират различни символи, точно единият от които е шпация, оценката на тази позиция е -2.

Общата оценка за подравняването е сумата от оценките по позиции. Например, ако $\mathtt{X} = \mathtt{GACTCAGG}$ и $\mathtt{Y} = \mathtt{AGTCGTC}$, то следното подравняване има обща оценка -6 (под всяка позиция в попълнените стрингове е написана нейната оценка):

$\sqcup$	G	A	C	T	C	A	G	$\sqcup$	G
A	G	$\sqcup$	T	$\sqcup$	C	$\sqcup$	G	T	C
-2	$+2$	-2	-1	-2	$+2$	-2	$+2$	-2	-1

Но ето това подравняване на $\mathtt{X}$ и $\mathtt{Y}$ има обща оценка -3:

$\sqcup$	G	A	C	T	C	A	G	$\sqcup$	G
A	G	$\sqcup$	$\sqcup$	T	C	$\sqcup$	G	T	C
-2	$+2$	-2	-2	$+2$	$+2$	-2	$+2$	-2	-1

Предложете ефикасен алгоритъм, който по дадени два непразни стринга $\mathtt{X} = x_1 \cdots x_n$ и $\mathtt{Y} = y_1 \cdots y_m$ над $\{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$ изчислява подравняване с максимална обща оценка. Достатъчно е алгоритъмът да изчислява само стойността, а не самото подравняване.

Упътване и уточнение: използвайте динамично програмиране. *За максимален брой точки е достатъчно да напишете коректно рекурентно уравнение, да опишете таблицата (каква е, едномерна, двумерна, или?), как се запълва и коя клетка от нея е търсеният отговор.*

Решение: Задачата е добре известната задача от изчислителната биология Sequence Alignment. Четирите букви в азбуката са първите букви на четирите бази на ДНК.

Задачата се решава с динамично програмиране. Разглеждаме всички префикси на $\mathtt{X} = x_1 \cdots x_n$ и $\mathtt{Y} = y_1 \cdots y_m$, включително и празните. Нека $M(i, j)$, за $0 \le i \le n$ и $0 \le j \le m$ означава максималната обща оценка за префиксите $x_1 \cdots x_i$ и $y_1 \cdots y_j$. Емилиян Рогачев: Ако и двата префикса са празни, оценката е нула, което дава начално условие $M(0, 0) = 0$. Но $M(i, 0)$ за $1 \le i \le n$ не са нули и, аналогичнно,

и $M(0,j)$ за $1 \leq j \leq m$ не са нули! Да разсъждаваме за $M(i,0)$ за $i > 0$. Тъй като задължително съпоставяме (под)стрингове с **еднаква** дължина, $M(i,0)$ отговаря на съпоставяне, позиция по позиция, на два стринга с държина i, а **не** на стринг с дължина i и на празен стринг. Но $M(i,0)$ по определение е оценката за префикса $x_1 \cdots x_i$ и празния префикс (на Y). Това означава, че попълваме празния префикс с шпации, i на брой, така че да стане обект, който може да бъде съпоставян с $x_1 \cdots x_i$ позиция по позиция. Очевидно оценката е $-2i$. Тогава $M(i,0) = -2i$ за $1 \leq i \leq n$ и $M(0,j) = -2j$ за $1 \leq j \leq m$

Ако нито един от префиксите не е празен, $M(i,j)$ се определя от три числа:

- $M(i-1,j-1)$, което означава, че съпоставяме x_i с y_j, което дава оценка на тази позиция (при изравняването на стринговете, това е ЕДНА И СЪЩА позиция и в двата, защото префиксите може да попълнени с празни символи), която е $+2$ в случай на съвпадение или -1 в противен случай. Тъй като общата оценка е адитивна, $M(i-1,j-1)$ се сумира с $+2$ или -1.
- $M(i-1,j)$, което означава, че съпоставяме x_{i-1} с празен символ на позиция j във втория стринг (попълване във втория стринг). Локалната оценка е -2, която се сумира с $M(i-1,j)$.
- $M(i,j-1)$, което означава, че съпоставяме y_{j-1} с празен символ на позиция i във първия стринг (попълване във първия стринг). Локалната оценка е -2, която се сумира с $M(i,j-1)$.

И така:

$$M(i,j) = \begin{cases} 0, & \text{ако } i = 0 \text{ и } j = 0, \\ -2i, & \text{ако } j = 0, \\ -2j, & \text{ако } i = 0, \\ \max\{M(i-1,j-1) + \delta, M(i-1,j) - 2, M(i,j-1) - 2\}, & \text{ако } i > 0 \text{ и } j > 0, \end{cases}$$

където δ е $+2$, ако $x_i = y_j$, или -1, в противен случай.

Таблицата е правоъгълна с размери $(n+1) \times (m+1)$. От началните условия имаме колона 0 запълнена, отгоре надолу, с $0, -2, -4, \ldots, -2n$ и ред 0, запълнен, отляво надясно, с $0, -2, -4, \ldots, -2m$. После попълваме отгоре надолу и отляво надясно, като $M[i,j]$ се изчислява чрез $M[i-1,j-1]$, $M[i-1,j]$ и $M[i,j-1]$. Търсеният числен отговор е $M[n,m]$.

Ако искаме и самото решение (подравняването), трябва да държим сметка за това как точно се получава максимум за всяка позиция—от коя от трите преди нея сме получили максимум, при повече от една вземаме произволна от тях—и след получаване на $M[n,m]$ да проследим обратно чак до $M[1,1]$. При това се оформя една "начупена линия" от $M[1,1]$ до $M[n,m]$, при която ход по диагонал надолу-и-надясно означава подравняване на символ от `X` със символ от `Y`, ход само надолу означава попълване на `X` с празен символ, а ход само надясно означава попълване на `Y` с празен символ.

Зад. 5 Двама играчи, наречени А и Б, играят следната игра. В началото има купчина от n камъни за някое $n \geq 2$. Играчите се редуват, вземайки камъни (поне един) от купчината. Първи играе А. Камък, който е махнат от купчината, вече не се връща в нея, така че купчината само намалява след всеки ход на играч. Печели този играч, след чийто (последен) ход остане точно един камък. Съществено ограничение е, че ако в купчината има точно k камъка, играчът, който е на ход, може да махне само такъв брой k' камъни, че k' е делител на k и $k' \neq k$. Предложете ефикасен алгоритъм, който връща 'А' в случай, че А има печеливша стратегия, или 'Б' в противен случай (тоест, когато Б има печеливша стратегия за всеки първи ход на А). Обосновете съвсем накратко коректността на Вашия алгоритъм. Оценете сложността по време на Вашия алгоритъм отгоре в асимптотичния смисъл. С други думи, достатъчно е да дадете добра $O(f(n))$ оценка, за подходяща функция $f(n)$. Не е необходимо $f(n)$ да е точна горна граница, достатъчно е да намерите разумно бързо растяща полиномиална функция.

Допуснете, че всеки от двамата играчи играе оптимално. Ще поясним какво означава това. Ако $n = 2$, очевидно А печели задължително, защото единственият делител на 2, различен от 2, е 1, така че А маха един камък, остава купчина с един камък и А печели. Ако $n = 3$, печели Б, защото единственият делител на 1, различен от 3, е 1, така че А няма друга възможност, освен да махне точно един камък, и остава купчина с два камъка; а вече видяхме, че ако в купчината има точно два камъка, печели този, който е на ход в момента, и това е играч Б. Сега да разгледаме $n = 4$. Делителите, които са допустими за А, са 1 и 2:

- Ако А махне 1 камък, ще остави на Б купчина с 3 камъка, което, както видяхме, е губеща конфигурация за този, който е на ход, а това е играч Б. С други думи, печеливша за А.

- Ако А махне два камъка, ще остави ще остави на Б купчина с 2 камъка, което, както видяхме, е печеливша конфигурация за този, който е на ход, а това е играч Б. С други думи, губеща за А.

И тъй като А играе оптимално, той или тя ще вземе **задължително** 1 камък на първия си ход и ще спечели.

Решение: Следният алгоритъм решава задачата чрез динамично програмиране и едномерна таблица status$[2, \ldots, n]$. За всяко $k \in \{2, \ldots, n\}$, status[k] има две възможни стойности: 'Печели' или 'Губи', с които означава съответно дали играчът, който е на ход, печели или губи при оптимална игра и на двамата играчи, ако е изправен(а) пред купчина с точно k камъка. Коректността следва от това, че:

- на status[2] се присвоява стойност 'Печели', съгласно наблюдението, което вече направихме – че при два камъка печели този, който е на ход;
- на status[k] за $k \geq 3$ се присвоява 'Печели' тстк за поне един същински делител m на k, status$[k-m]$ е 'Губи'. Ако правехме подробно доказателство по индукция, това, status$[k-m]$ е 'Губи' щеше да означава, че другият играч е в губеща позиция, ако е изправен(а) пред пред купчина с точно $k - m$ камъка, и това щеше да идва от индуктивната хипотеза. Ключовото наблюдение е, че за $k \geq 3$, играчът, изправен пред купчина от k камъка, не може да спечели с един ход, така че печели тстк може с един ход да вкара опонента си в ситуация, която е задължително губеща за него или нея (за опонента).

Алгоритъмът работи във време $O(n^2)$. Тази горна граница се обосновава със съображението, че за всяка стойност k на външния цикъл, вътрешният работи в $O(k)$.

```
Кой печели(n: int)
 1  status[2] ← 'Печели'
 2  for k ← 3 to n
 3      tmp ← 'Губи'
 4      for m ∈ proper_divisors(k)
 5          if status[k − m] = 'Губи'
 6              tmp ← 'Печели'
 7              break
 8      status[k] ← tmp
 9  if status[n] = 'Печели'
10      return 'А'
11  else
12      return 'Б'
```

Зад. 6 Докажете, че следната задача е **NP**-пълна.

Изчислителна задача ЗадачаZ
Общ пример: Множество от булеви променливи $X = \{x_1, \ldots, x_n\}$ и множество дизюнктивни клаузи $C = \{c_1, \ldots, c_m\}$ над X.
Въпрос: Дали съществуват поне две (различни) удовлетворяващи валюации за C?

Решение: Тази задача е известна като Double SAT. Първо, принадлежността на Double SAT към **NP** е очевидна – при даден ДА-пример на Double SAT, недетерминираната машина на Тюринг отгатва две удовлетворяващи валюации, проверява, че са различни, и проверява, че всяка от тях наистна е удовлетворяваща за дадения пример на Double SAT. Това очевидно може да се извърши в полиномиално време.

За да покажем, че Double SAT е **NP**-пълна, ще направим редукция от 3SAT. Да си припомним 3SAT:

Изчислителна задача 3SAT
Общ пример: Множество от булеви променливи $X = \{x_1, \ldots, x_n\}$ и множество дизюнктивни клаузи

$C = \{c_1, \ldots, c_m\}$ над X, такива че всяка клауза има точно три литерала.
Въпрос: Дали съществува удовлетворяваща валюация за C?

Нека е даден пример $\langle X, C\rangle$ на 3SAT. Ще конструираме пример $\langle X', C'\rangle$ на Double SAT, такъв че C е удовлетворим тогава и само тогава, когато C' има поне две удовлетворяващи валюации. За целта добавяме една нова булева променлива (която не е в X), да я наречем z, и правим следната клауза $k = \{z, \overline{z}\}$. После правим $C' \leftarrow C \cup k$.

Да допуснем, че C е удовлетворима. Тогава тя има поне една удовлетворяваща валюация $f : X \to \{0,1\}^n$. Щом f е удовлетворяваща, във всяка клауза от C има поне един литерал α, такъв че $f(\alpha) = 1$. Конструираме две валюации g и g' за C' така: рестрикцията на всяка от тях до домейна X е точно като f, а $g(z) = 0$ и $g'(z) = 1$. Очевидно g и g' са различни, имайки различни стойности върху z. Ще покажем, че във всяка клауза на $c \in C'$ има поне един литерал, който има стойност 1 както под g, така и под g':

- ако $c \in C$, това следва от фактите, че f е удовлетворима за C и че g и g' върху променливите от X съвпадат със съответните стойности на f.
- ако $c = \{z, \overline{z}\}$, то $g(\overline{z}) = 1$ и $g'(z) = 1$.

Сега да допуснем, че C' е удовлетворима от поне две различни удовлетворяващи валюации g и g'. Тогава и под g, и под g' е вярно, че за всяка клауза на C' има литерал, чиято стойност е 1, което веднага влече, че рестрикцията на коя да е от тях върху домейн X е удовлетворяваща валюация за C.