Примерни решения на изпита по ДАА на КН2 & Инф, проведен на 29.06.2019 г.

Задача Задачата за разпознаване Свързаност се дефинира така: общият пример е неориентиран граф $G = (V, E)$, а въпросът е дали G е свързан.

Докажете, че всеки алгоритъм за Свързаност, който прави достъпи до графа само чрез задаване на въпроси от вида "Има ли ребро между връх i и връх j?", задава поне n^2 въпроса, ако броят на върховете е $2n$.

Решение: Има аргументация чрез противник. Противникът разбива множеството от върховете на две подмножества U и W, които наричаме *дяловете*, като $|U| = |W| = n$, и след това действа така. Получавайки запитване за върхове i и j,

- ако i и j са в един и същи дял, противникът отговаря ДА,
- ако i и j са в различни дялове, противникът отговаря НЕ.

Съществуват точно n^2 двуелементни множества $\{x, y\}$, такива че x и y са от различни дялове. Ще покажем, че всеки алгоритъм AlgConn за тази задача трябва да запита за всяко от тях. Да допуснем противното: AlgConn е коректен алгоритъм за Свързаност и съществува $x \in U$ и съществува $y \in W$, такива че AlgConn не е направил запитване за x и y.

- Ако AlgConn отговори ДА, то противникът конструира несвързан граф, в който U и W са клики и няма нито едно ребро между връх от U и връх от W. По този начин противникът опровергава AlgConn, тъй като всички отговори, които е давал, са консистентни с този граф, а графът не е свързан.
- Ако AlgConn отговори НЕ, то противникът конструира свързан граф, в който U и W са клики и има ребро между x и y. По този начин противникът опровергава AlgConn, тъй като всички отговори, които е давал, са консистентни с този граф, а графът е свързан.

Задача Даден е масив от цели числа $A[1, \ldots, n]$. Допуснете, че числата са две по две различни. Предложете итеративен алгоритъм със сложност по време $O(n)$, който намира максималната дължина на подмасив $A[i, i+1, \ldots, j]$, такъв че $A[i] < A[i+1] < \cdots < A[j]$. Обосновете прецизно коректността на Вашия алгоритъм чрез инварианта. Дайте съвсем кратка обосновка на сложността по време.

Решение Едно възможно решение е следното.

```
Longest Contiguous Sequence(A[1, 2, ..., n])
 1  temp ← 1
 2  max ← 1
 3  for i ← 2 to n
 4      if A[i − 1] < A[i]
 5          temp ++
 6          if temp > max
 7              max ← temp
 8      else
 9          temp ← 1
10  return max
```

Ще казваме "растящ подмасив" с очевидния смисъл. Следното твърдение е инварианта за цикъла на редове 3–9:

> При всяко достигане на ред 3, променливата *temp* съдържа дължината на максималния растящ подмасив, завършващ на $A[i-1]$, а *max* съдържа дължината на максималния растящ подмасив в $A[1, \ldots, i-1]$.

Ето доказателство.

База: При първото достигане на ред 3, променливата *temp* съдържа 1 заради присвояването на ред 1, а променливата *max* съдържа 1 заради присвояването на ред 2. В този момент i има стойност 2, така че $i - 2 = 1$. Очевидно

- максималният растящ подмасив, завършващ на $A[i-1]$, е $A[1]$ е с дължина 1,
- максималният растящ подмасив в $A[1, \ldots, i-1]$ е $A[1]$ с дължина 1. ✓

Поддръжка: Да допуснем, че твърдението е изпълнено за някое достигане на ред 3, което не е последното.

- Да допуснем, че $A[i-1] < A[i]$. От една страна, това влече, че дължината на максималния растящ подмасив, завършващ на $A[i]$, е с единица по-голяма от дължината на максималния растящ подмасив, завършваща на $A[i-1]$. От друга страна, на ред 4 променливата *temp* се инкрементира. Съгласно допускането, сега е вярно, че *temp* съдържа дължината на максималния растящ подмасив, завършващ на $A[i]$.
  - Да допуснем, че дължината на максималния растящ подмасив, завършваща на $A[i]$, е по-голяма от дължината на максималния растящ подмасив в $A[1, \ldots, i-1]$. Съгласно допускането и току-що доказаното, вярно е, че $temp > max$. Условието на ред 6 е истина и се изпълнява присвояването на ред 7. След него е вярно, че *max* съдържа дължината на максимални растящ подмасив в $A[1, \ldots, i]$. Изпълнението отива на ред 3.
  - Да допуснем, че дължината на максималния растящ подмасив, завършваща на $A[i]$, не е по-голяма от дължината на максималния растящ подмасив в $A[1, \ldots, i-1]$. Съгласно допускането и току-що доказаното, вярно е, че *max* съдържа дължината на максимални растящ подмасив в $A[1, \ldots, i]$ и $temp \leq max$. Условието на ред 6 е лъжа и изпълнението отива на ред 3.
- Да допуснем, че $A[i-1] > A[i]$. От една страна, това влече, че дължината на максималния растящ подмасив, завършващ на $A[i]$, е 1. От друга страна, на ред 9 променливата *temp* получава стойност 1. Сега *temp* съдържа дължината на максималния растящ подмасив, завършваща на $A[i]$. Съгласно допускането и факта, че $A[i-1] > A[i]$, променливата *max* съдържа дължината на максималния растящ подмасив в $A[1, \ldots, i]$.

При следващото достигане на ред 3, i се инкрементира имплицитно. Спрямо новата стойност на i е вярно, че променливата *temp* съдържа дължината на максималния растящ подмасив, завършваща на $A[i-1]$, а *max* съдържа дължината на максималния растящ подмасив в $A[1, \ldots, i-1]$.

Терминация: При последното достигане на ред 3 променливата i съдържа $n+1$. Заместваме i с $n+1$ в инвариантата и получаваме

max съдържа дължината на максимална непрекъсната растяща поредица в $A[1, \ldots, n]$.

Следователно, на ред 10 алторитъмът връща дължината на максимална непрекъсната растяща поредица в $A[1, \ldots, n]$.

Сложността по време е линейна, понеже тялото на цикъла се изпълнява $\Theta(n)$ пъти, а всяко изпълнение става за време $\Theta(1)$.

Задача Фирмата "Грандиозо Ай Ти" има n служители. Във фирмата има строга йерархия: всеки служител освен Големия Началник има точно един шеф. Големия Началник няма шеф, естествено. Предстои да се направи фирмено тържество, на което трябва да бъдат поканени колкото е възможно повече служители на фирмата. Но има проблем: всеки служител е във взаимна непоносимост със своя шеф, а на тържеството не трябва да има хора, които не се понасят.

Предложете ефикасен алгоритъм,

- чийто вход е описание на фирмената йерархия, а именно за всеки служител (без Големия Началник) името на неговия или нейния шеф, и
- чийто изход е максималният брой служители, които нямат взаимни непоносимости.

Не е необходимо да пишете псевдокод. Достатъчно е да обясните ясно и недвусмислено идеята на Вашия алгоритъм и по коя алгоритмична схема, измежду тези, които сте изучавали на лекции, е изграден той. Можете да ползвате алгоритми, изучавани на лекции, но трябва да обясните как работят – не е достатъчно да кажете името на задачата и да кажете, че алгоритъм за нея е изучаван. И трябва накратко да обосновете както коректността, така и сложността по време.

Решение: Това е задачата Максимална Антиклика, като се иска само размерът на оптимално решение. Графът е (кореново) дърво, защото става дума за йерархия с точно един корен. На лекции са изучавани два алгоритъма за Максимална Антиклика върху дървета: единият базиран на алчната схема, а другият базиран на схемата Динамично програмиране. Всеки от тях може да бъде ползван за решението.

Ето решение по Динамично програмиране. Входът представя кореново дърво чрез масив от родителите. От това, в линейно време може да се генерират списъците на съседство. Пуска се обхождане в postorder. Всеки връх u има етикет (u^+, u^-), като

- u^+ е размерът на максимална антиклика в поддървото, вкоренено в u, която съдържа u,
- а u^- е размерът на максимална антиклика в поддървото, вкоренено в u, която не съдържа u.

Ако u е листо, то $(u^+, u^-) = (1, 0)$ по очевидни причини. В противен случай, ако децата на u са $v_1, \ldots, v_k$:

$$u^+ = 1 + \sum_{i=1}^{k} v_i^-$$

$$u^- = \sum_{i=1}^{k} \max\{v_i^-, v_i^+\}$$

Обосновката накратко: ако u се съдържа в антиклика, то нито едно от децата му не може да се съдържа, така че за сумираме вторите компоненти на етикетите; ако обаче u не се съдържа, за всяко дете вземаме по-голямата от двете компоненти на етикета и така максимизираме стойността на u^-.

Алгоритъмът връща $\max\{r^+, r^-\}$, където r е коренът (Големия Шеф) и това е коректно, понеже за всяка максимална антиклика е вярно, че или съдържа, или не съдържа корена.

Сложността по време е линейна, което следва от факта, че сложността на обхождането е линейна—примерно, можем да ползваме DFS—и за всеки вътрешен връх на дървото добавяме работа, пропорционална на броя на децата му.

Задача Разгледайте следния алгоритъм

ALG1($A[1, \ldots, n]$: масив от цели положителни числа, като $n \geq 2$)

1 Нека $B[1 \ldots n]$ е масив от цели положителни числа
2 $B[1] \leftarrow A[1]$
3 $t \leftarrow -\infty$
4 $i \leftarrow 2$
5 **do**
6 $\quad B[i] \leftarrow \max(B[i-1], A[i] - (i-1))$
7 $\quad t \leftarrow \max(t, B[i-1] + A[i] + (i-1))$
8 $\quad i{+}{+}$
9 **while** $i \leq n$
10 **return** t

Докажете формално и прецизно, че алгоритъмът връща

$$\max\{A[p] + A[q] + (q-p) \mid 1 \leq p < q \leq n\}$$

Казано на прост български: алгоритъмът връща максималната сума на два различни елемента и разстоянието между тях в масива, за всички двойки различни елементи.

Решение: Следното твърдение е инварианта за цикъла.

При всяко достигане на ред 9:

1. $B[i-1]$ съдържа $\max\{A[k] - (k-1) \mid 1 \leq k \leq i-1\}$,
2. t съдържа $\max\{A[p] + A[q] + (q-p) \mid 1 \leq p < q \leq i-1\}$

Ето доказателство.

База: При първото достигане на ред 9, i съдържа 3 заради присвояването на ред 3 и инкрементацията на ред 8, така че $B[i-1]$ е всъщност $B[2]$.

От една страна, ако четем псевдокода, $B[2]$ съдържа $\max\{A[1], A[2] - 1\}$ заради присвояванията на ред 2 и ред 6; забележете, че при първото достигане на ред 6, i съдържа 2. От друга страна, по отношение на текущото i, което е 3, множеството $\{A[k] - (k-1) \mid 1 \leq k \leq i-1\}$, за което говори инвариантата, е $\{A[1] - 0, A[2] - 1\}$. Първата част на инвариантата е истина.

От една страна, ако четем псевдокода, t съдържа $A[1] + A[2] + 1$ заради присвояването на ред 7; в този момент $i = 2$. От друга страна, по отношение на текущото (при достигането на ред 9) $i = 3$, множеството $\{A[p] + A[q] + (q-p) \mid 1 \leq p < q \leq i-1\}$, за което говори инвариантата, всъщност е $\{A[1] + A[2] + (2-1)\}$. И втората част на инвариантата е истина.

Поддръжка: Нека инвариантата е истина за някое достигане на ред 9, което не е последното. Ще се убедим, че тя се запазва при изпълнението на тялото на цикъла.

На ред 6, $B[i]$ получава стойност $\max\{B[i-1], A[i] - (i-1)\}$. По допускане,

$$B[i-1] = \max\{A[1], A[2] - 1, A[3] - 2, \ldots, A[i-1] - (i-2)\}$$

Тогава

$$B[i] = \max\{\max\{A[1], A[2] - 1, A[3] - 2, \ldots, A[i-1] - (i-2)\}, A[i] - (i-1)\}$$

което може да запишем накратко като

$$B[i] = \max\{A[1], A[2] - 1, A[3] - 2, \ldots, A[i] - (i-1)\}$$

След инкрементирането на i на ред 8, изразено чрез новото i, това става

$$B[i-1] = \max\{A[1], A[2] - 1, A[3] - 2, \ldots, A[i-1] - (i-2)\}$$

Виждаме, че първата част на инвариантата отново е истина.

На ред 7, t получава стойност $\max(t, B[i-1] + A[i] + (i-1))$. По допускане, старото t е

$$t_{\text{old}} = \max\{A[p] + A[q] + (q-p) \mid 1 \leq p < q \leq i-1\}$$

а $B[i-1]$ е

$$B[i-1] = \max\{A[1], A[2]-1, A[3]-2, \ldots, A[i-1]-(i-2)\}$$

Тогава $B[i-1] + A[i] + (i-1)$ е

$$\begin{aligned}
&\max\{A[1], A[2]-1, A[3]-2, \ldots, A[i-1]-(i-2)\} + A[i] + (i-1) = \\
&\max\{A[1] + A[i] + (i-1), A[2] - 1 + A[i] + (i-1), \ldots, A[i-1] - (i-2) + A[i] + (i-1)\} = \\
&\max\{A[k] + A[i] + (i-k) \,|\, 1 \leq k \leq i-1\}
\end{aligned}$$

И така, старото t съдържа стойността на максимална сума на два различни елемента и разстоянието между тях, по всички елементи на $A[1, \ldots, i-1]$, а $B[i-1] + A[i] + (i-1)$ съдържа стойността на максимална сума на два различни елемента и разстоянието между тях, по всички елементи на $A[1, \ldots, i]$, единият от които е $A[i]$. Тогава техният максимум, което новото t, съдържа стойността на максимална сума на два различни елемента и разстоянието между тях, по всички елементи на $A[1, \ldots, i]$. След инкрементирането на ред 8, изразено в новото i, това става

$$t = \max\{\{A[p] + A[q] + (q-p) \,|\, 1 \leq p < q \leq i-1\}$$

Виждаме, че втората част на инвариантата отново е истина.

Терминация: Разглеждаме момента, в който изпълнението е на ред 9 за последен път. Очевидно i съдържа $n+1$ в този момент. От втората част на инвариантата получаваме, заествайки i с $n+1$, че t съдържа стойността на максимална сума на два различни елемента и разстоянието между тях, по всички елементи на $A[1, \ldots, n]$. И на ред 10 алгоритъмът връща точно това.

Зад. 5 Дадени са n монети, разположени една до друга в редица. Монетите са от най-различни деноминации. За целите на задачата, деноминациите са произволни цели положителни числа. Има двама играчи, да ги наречем Албена и Борис, които се редуват. Всеки от тях, когато е неговият или нейният ред, взема точно една монета или от левия, или от десния край на редицата. Играта продължава до изчерпването на монетите в редицата (което означава, че монетите са разпределени между играчите). След играта всеки задържа монетите, които е събрал(а). Албена играе първа.

Иска са да конструирате ефикасен алгоритъм, който намира максималната възможна сума за Албена.

- Първо покажете, че алчната стратегия "вземам по-голямата крайна монета, налична в момента" не води непременно до оптимално решение.
- Предложете ефикасен алгоритъм, с който Албена намира оптимално решение. Допустимо е алгоритъмът да връща само стойността на някое оптимално решение, а не самото решение. Ако Вашият алгоритъм е по схемата Динамично Програмиране, допустимо е да покажете само рекурсивната декомпозиция, без да пишете псевдокод.

Можете да допуснете, че n е четно.

Решение: Да опровергаем алчната идея. Контрапример е $1, 2, 100, 2$. Ако Албена алчно грабне двойката вдясно, понеже е по-голяма от единицата вляво, оставя редица $1, 2, 100$ и Борис взема стотицата. Ако обаче Албена започне, вземайки единицата вляво, то Борис има избор между двете двойки в $2, 100, 2$, при което Албена взема стотицата.

Има и по-изтънчена алчна идея, ако n е четно. Да разбием монетите на тези, които са четни позиции, и тези които са на нечетни. Условно да ги наречем "червените" и "сините". Примерно,

$$1, 3, 6, 3, 1, 3$$

Играейки първа, Албена лесно може да събере монетите от даден цвят (и само тях). Така че тя може първо да сметне кое множество има не по-малка сума от другото и след това да събере монетите му, принуждавайки Борис да събере останалите. Тази идея гарантира на Албена, че ще вземе не по-малко от Борис, но не ѝ гарантира, че ще вземе максимална сума. В примера: и червеното множество има сума 8, а синьото, 9, така че Албена ще вземе монети на стойност 9, следвайки тази идея. Обаче максималната ѝ печалба е 10: нека тя започне с тройката вдясно, оставяйки на Борис избор от две единици:

$$1, 3, 6, 3, 1$$

Която и единица да вземе Борис, Албена взема другата единица, след което Борис трябва да избира между две тройки:

$$3, 6, 3$$

Която и тройка да вземе Борис, Албена взема шестицата. Тя е взела общо $3 + 1 + 6 = 10$.

Има ефикасен алгоритъм по схемата Динамично Програмиране, който гарантира максимална печалба за Албена. Да кажем, че $\mathrm{opt}(i, j)$ е оптималната сума, която Албена може да събере, ако има пред себе си подредицата от i-тата монета включително до j-тата монета включително, за $1 \le i < j \le n$.[1] Това означава, че както и да играе Борис, тя ще вземе поне толкова. Ако Албена може да изчислява тази функция ефикасно, тя разполага и с ефикасно решение, защото ѝ трябва $\mathrm{opt}(1, n)$. Ще измислим подходяща рекурсия за изчислението на $\mathrm{opt}(i, j)$ чрез някакви $\mathrm{opt}(i', j')$, където $j' - i' < j - i$. С други думи, управляващият параметър е разликата между втория и първия аргумент. Тази функция ще се изчислява чрез запълване на (част от) двумерен масив $\mathrm{opt}[1, \ldots, n][1, \ldots, n]$. Решението е $\mathrm{opt}[1][n]$.

Нека масив $M[1, \ldots, n]$ съдържа редицата от монети; тоест, $M[i]$ е стойността на i-тата монета отляво, за $1 \le i \le n$.

Дъното на рекурсията се реализира чрез

$$\mathrm{opt}[i, i + 1] = \max\{M[i], M[i + 1]\}$$

за $1 \le i < n$. Идеята е проста: ако Албена играе върху двуелементната редица от монети $M[i], M[i+1]$, тя взема тази, която е не по-малка от другата, после Борис взема другата и играта приключва.

Същината е изчислението на $\mathrm{opt}[i, j]$ за $j > i + 1$. Албена избира дали да вземе $M[i]$, или да вземе $M[j]$. Други възможности няма.

[1] Забележете, че няма смисъл да допускаме $i = j$, защото при четно n, Албена в своя последен ход играе върху редица от две монети. Последното вземане задължително се прави от Борис. А алгоритъмът, който строим, е за Албена.

- Ако Албена вземе $M[i]$, на ход е Борис, който избира между $M[i+1]$ и $M[j]$. Тъй като допускаме, че Борис играе оптимално, той ще направи изборът, който му дава максимална печалба, и за Албена ще остане по-лошото от $\mathrm{opt}[i+1, j-1]$ и $\mathrm{opt}[i+2, j]$. Така че Албена ще вземе най-много, при оптимална игра на Борис,

  $$M[i] + \min\{\mathrm{opt}[i+1, j-1], \mathrm{opt}[i+2, j]\}$$

- Ако Албена вземе $M[j]$, на ход е Борис, който избира между $M[i]$ и $M[j-1]$. Тъй като допускаме, че Борис играе оптимално, той ще направи изборът, който му дава максимална печалба, и за Албена ще остане по-лошото от $\mathrm{opt}[i, j-2]$ и $\mathrm{opt}[i+1, j-1]$. Така че Албена ще вземе най-много, при оптимална игра на Борис,

  $$M[j] + \min\{\mathrm{opt}[i, j-2], \mathrm{opt}[i+1, j-1]\}$$

Албена трябва да избере максимума от тези. И така:

$$\mathrm{opt}[i, j] = \max\left\{M[i] + \min\{\mathrm{opt}[i+1, j-1], \mathrm{opt}[i+2, j]\}, M[j] + \min\{\mathrm{opt}[i, j-2], \mathrm{opt}[i+1, j-1]\}\right\}$$

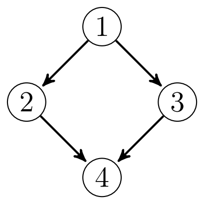
Зад. 6 В тази задача става дума за топологически сортирания на дагове.

- Професор Дълбоков казва, че даг G има едно единствено топологическо сортиране тогава и само тогава, когато G има точно един източник и точно един сифон. Прав ли е професор Дълбоков? Обосновете отговора си.

- Предложете ефикасен алгоритъм, който връща "ДА", ако G има едно единствено топологическо сортиране, или връща две различни топологически сортирания на G, ако има такива. Обосновете накратко коректостта и сложността му.

- Професор Факториелски казва, че разполага с полиномиален алгоритъм, който получава на входа даг G и връща всички топологически сортирания на G. Какво бихте казали за това?

Решение: Професор Дълбоков греши. Ето малък пример за даг с точно един източник и точно един сифон, който има повече от едно топологическо сортиране:



Това са двете му топологически сортирания:

$1, 2, 3, 4$
$1, 3, 2, 4$

Даг има точно едно топологическо сортиране тогава и само тогава, когато има ориентиран Хамилтонов път. С други думи, тстк неговото рефлексивно и транзитивно затваряне е релация на линейна наредба. В едната посока, ако има Хамилтонов път, очевидно единственото възможно топологическо сортиране е това, в което върховете са подредени по начина, по който са подредени в Хамилтоновия път. В другата посока, наличието на единствено топологическото сортиране влече, че за всеки два поредни върха u и v (в този ред) в него има ребро (u, v) в дага; ако за поне два поредни u и V няма ребро (u, v) в дага, те може да бъдат разменени в сортирането и при това да се получи друго валидно топо-сортиране. Щом за всеки два поредни u и v, в този ред, има ребро (u, v), то в дага има Хамилтонов път.

Това дава идея конструиране на желания алгоритъм. Пускаме алгоритъм за топо-сортиране, да кажем алгоритъма на Tarjan, след което за всеки два поредни върха u и v, в този ред, проверяваме дали има ребро (u, v). Ако се окаже, че за всеки два поредни върха има такова ребро, връщаме ДА. В противен случай, за първата двойка поредни върхове u и v, за която няма ребро (u, v), генерираме второ топо-сортиране, което се получава от даденото чрез транспозиция на u и v, и връщаме двете сортирания.

Коректността следва директно по-горния параграф. Да разгледаме сложността по време. Правим стандартното допускане, че графът е представен чрез списъци на съседство. Първата фаза, а именно топо-сортирането, работи във време $\Theta(n + m)$, както бе показано на лекции. Но втората фаза също работи във време $\Theta(n + m)$, защото отново става дума за "преброждане" на всички списъци: за всеки връх u в топо-наредбата, който не е последният, в най-лошия случай прочитаме целия му списък, за да видим дали следващия в наредбата връх е в този списък. Това е същото като "преброждането" на всички списъци, което, както знаем, става във време $\Theta(n + m)$.

Дори да се наложи да се генерира второ топо-сортиране, сложността по време остава $\Theta(n + m)$.

Професор Факториелски лъже и/или не знае за какво говори. Всички топо-сортирания може да са супер-полиномиално много в размера на графа. Най-лошият случай е празен даг (няма ребра). Тогава всяка пермутация на върховете е едно топо-сортиране, така че само броят на топо-сортиранията е $n!$. Генерирането на $n!$ обекта няма как да стане в полиномиално време.