Примерни решения на изпита по ДАА на Информатика и КН 2 поток, проведен на 25 юни 2024 г.

Зад. 1 Разглеждаме стринговете над азбуката $\Sigma = \{0, 1\}$. Нека $X \subseteq \Sigma^+$ и $\sigma \in \Sigma^+$. Ще казваме, че σ е *забележителен по отношение на* X, ако е изпълнено следното.

- За всеки $x \in X$,
- за всеки алгоритъм AlgSubstr(a, b), който връща или Да, ако a съдържа b като подстринг, или Не, ако a не съдържа b като подстринг, където $a, b \in \Sigma^+$:
- AlgSubstr(x, σ) прави достъп до всяка буква на x.

Докажете, че 0 е забележителен по отношение на Σ^+.

Докажете, че 01 не е забележителен по отношение на $\{x \in \Sigma^+ : |x| \text{ е нечетно число}\}$.

Докажете, че 01 е забележителен по отношение на $\{x \in \Sigma^+ : |x| \text{ е четно число}\}$.

Решение: Свойството, за което става дума тук, на английски се нарича *evasiveness*. По-точно, σ е *evasive pattern*, ако всеки (коректен) алгоритъм, който установява наличието или отсъствието на подстринг σ на даден стринг x, в най-лошия случай чете всяка буква на x.

Първо ще докажем, че търсенето на 0 налага четенето на всяка буква на x. Доказателството е с противник. Ето стратегията на противника: той отговаря 1 на всяко запитване на алгоритъма, като запитванията на алгоритъма са за стойности x_i, където $i \in \{1, \ldots, n\}$. Ако алгоритъмът пропусне да провери стойността на поне едно x_j, противникът ще го опровергае винаги:

- ако алгоритъмът върне ДА, то противникът ще генерира $x = 11 \cdots 1$, като това е консистентно с отговорите, които е давал на алгоритъма,
- ако алгоритъмът върне НЕ, то противникът ще генерира такъв x, че $x_j = 0$, като това е консистентно с отговорите, които е давал на алгоритъма.

Сега ще докажем, че търсенето на 01 не налага четенето на всяка буква на x, ако дължината на x е нечетна. Нека $|x| = n = 2k + 1$. Нашият алгоритъм първо чете буквите на четните позиции: x_2, x_4, $\ldots$, x_{2k}. Следните случаи са изчерпателни и взаимно изключващи се.

- Ако всички тези букви са нули, то 01 се среща като подстринг на x тстк поне една буква от x_3, x_5, $\ldots$, x_{2k+1} е единица. Ерго, няма нужда да четем x_1.
- Ако всички тези букви са единици, то 01 се среща като подстринг на x тстк поне една буква от x_1, x_3, $\ldots$, x_{2k-1} е нула. Ерго, няма нужда да четем x_{2k+1}.
- Ако редицата от тези букви започва с нули, последвани от поне една единица, то отговорът е ДА. Ерго, няма нужда да се четат други букви от x.
- Ако редицата от тези букви се състои от единици, последвани от нули, нека най-дясната единица е x_t, което означава, че най-лявата нула е x_{t+2}. Тогава 01 се среща като подстринг на x тстк 01 е подстринг на $x_1 \cdots x_t$ или $x_{t+2} \cdots x_n$, понеже стойността на x_{t+1} е без значение. Ерго, няма нужда да четем x_{t+1}.

Сега ще докажем, че търсенето на 01 налага четенето на всяка буква на x, ако дължината на x е нечетна. Нека $|x| = n = 2k$.

По време на работата на алгоритъма, противникът отговаря така, сякаш 01 не се среща. Забележете, че ако x не съдържа 01, то $x = 1^t 0^{n-t}$ за някое $t \in \{0, \ldots, n\}$. Ерго, противникът отговаря по такъв начин, сякаш x е $11 \cdots 100 \cdots 0$, но, ако пропуснем да прочетем поне един бит на x, той (противникът) винаги може да ни опровергае. За тази цел той може да промени (не напълно прочетения) x, ако пожелае, по такъв начин, че 01 да се появи като подстринг.

Противникът поддържа два индекса ℓ и r, като подстрингът $X_L = x_1 \cdots x_\ell$ се състои само от единици, подстрингът $X_R = x_r \cdots x_n$ се състои само от нули, а подстрингът $X_M = x_{\ell+1} \cdots x_{r-1}$ е неясен до момента. В самото начало $\ell = 0$, $r = n + 1$, така че x съвпада с X_M. След всяко запитване за стойността на x_i от страна на алгоритъма, за какво да е $i \in \{1, \ldots, n\}$, противникът отговаря ето така.

- Ако $i \in \{1, \ldots, \ell\}$, то противникът дава стойност 1 на x_i, понеже x_i попада в X_L.
- Ако $i \in \{r, \ldots, n\}$, то противникът дава стойност 0 на x_i, понеже x_i попада в X_R.
- В противен случай е в сила $i \in \{\ell + 1, \ldots, r - 1\}$, тоест x_i попада в X_M. Точно един от $x_{\ell+1} \cdots x_i$ и $x_i \cdots x_{r-1}$ има четна дължина.
  - Ако $x_{\ell+1} \cdots x_i$ е с четна дължина, което е същото като $i - \ell$ да е четно, противникът прави $\ell \leftarrow i$, което означава, че X_L нараства с четен брой букви. Съответно X_M намалява със същия брой букви, а X_R не се променя.
  - Ако $x_i \cdots x_{r-1}$ е с четна дължина, което е същото като $r - i$ да е четно, противникът прави $r \leftarrow i$, което означава, че X_R нараства с четен брой букви. Съответно X_M намалява със същия брой букви, а X_L не се променя.

Твърдението "точно един от $x_{\ell+1} \cdots x_i$ и $x_i \cdots x_{r-1}$ има четна дължина" се доказва тривиално с инвариант, който гласи, че при всяко четене на бит на x от страна на алгоритъма, подстрингът X_M е с четна дължина. Това е вярно в самото начало, понеже тогава X_M съвпада с x, а след това X_M намалява с четен брой букви след всяко четене на бит от страна на алгоритъма. Щом $X_M = x_{\ell+1} \cdots x_{r-1}$ има четна дължина и $i \in \{\ell + 1, \ldots, r - 1\}$, очевидно точно единият от $x_{\ell+1} \cdots x_i$ и $x_i \cdots x_{r-1}$ има четна дължина.

Да допуснем, че нашият алгоритъм, който проверява дали 01 се среща в x, пропусне да прочете поне един бит от x. Ако подстрингът X_M е непразен, противникът ни опровергава веднага: ако сме отговори ДА, той прави $x = 11 \cdots 100 \cdots 0$, примерно записвайки само нули в X_M, а ако сме отговорили НЕ, той слага 01 някъде в X_M; забележете, че щом X_M е с четна дължина, по-голяма от нула, място за 01 има. А ако подстрингът X_M е празен, което е същото като $x = X_L X_R$, то,

- ако сме отговорили ДА, то противникът прави $x = 11 \cdots 100 \cdots 0$, попълвайки непрочетените битове по подходящ начин,
- а ако сме отговорили НЕ и непрочетеният бит е от X_L, то този непрочетен бит не е най-десният бит на X_L и противникът го прави 0 и така се появява подстринг 01,
- а ако сме отговорили ДА и непрочетеният бит е от X_R, то този непрочетен бит не е най-левият бит на X_L и противникът го прави 1 и така се появява подстринг 01.

5 т. **Зад. 2** Напишете псевдокод на алгоритъма, който построява Ойлеров цикъл в мултиграф G. Този алгоритъм е изучаван по Дискретни Структури. Какви свойства трябва да притежава G, за да има Ойлеров цикъл? Не е необходимо да давате подробен псевдокод. Достатъчно е да алгоритъмът да е описан ясно и недвусмислено и да се виждат циклите в него.

Може да допуснете, че G няма примки.

15 т. Докажете коректността на алгоритъма чрез инварианти на цикли. Не е необходимо доказателството да е подробно! Напишете го само в общи линии. Важното е инвариантите да са смислени.

Решение В общи линии, това е правено на лекции по Дискретни Структури.

Зад. 3 Трябва да бъдат проведени изпити $t_1, t_2, \ldots, t_n$. Всеки изпит t_i се характеризира със своето стартово време s_i и своето време на приключване f_i, като $s_i < f_i$. За простота допуснете, че числата $s_1, \ldots, s_n, f_1, \ldots, f_n$ са уникални.

Поначало са били дадени n зали за провеждането на изпитите. След това обаче е излязла заповед изпитите да се проведат в колкото е възможно по-малко зали, но при очевидното ограничение да няма момент, в който два или повече изпита са в една и съща зала. Допуснете, че всяка зала е достатъчно голяма за кой да е от изпитите.

Задачата е тази: дадени са изпитите като наредени двойки $t_1 = (s_1, f_1)$, $t_2 = (s_2, f_2)$, $\ldots$, $t_n = (s_n, f_n)$ и Вие трябва да конструирате ефикасен алгоритъм, който намира минималния брой зали, в които може да бъдат проведени тези изпити при ограничението във всяка зала, във всеки момент да има не повече от един изпит. Докажете подробно коректността на Вашия алгоритъм и намерете сложността му по време. Точки ще се дават само на решения с добре обоснована коректност и с оптимална, в асимптотичния смисъл, сложност по време.

Решение: Да кажем, че изпити t_i и t_j са *безконфликтни*, ако може да бъдат проведени в една и съща зала. Това може да стане тстк $f_i < s_j$ или $f_j < s_i$. Нека обратното е t_i и t_j да са *в конфликт*.

Нека T е множеството от изпитите. Задачата е да бъде намерено минималното число k, такова че съществува разбиване на T на k подмножества, такова че във всяко подмножество, всеки два различни изпита са безконфликтни.

Да кажем, че *дебелината* на T, която ще означаваме с "$th(T)$", е максималният брой изпити, всеки два от които са в конфликт. Ако мислим за изпитите като за затворени интервали, дебелината е максималният брой интервали, които имат обща точка. Очевидно $1 \leq th(T) \leq n$. Дебелина 1 е възможна съгласно тези дефиниции, ако допуснем, че всеки изпит е в конфликт със себе си, което, в някакъв смисъл, е точно така. Ако дебелината е 1, няма никакви конфликти между различни изпити и всички изпити може да се проведат в само една зала. Другата крайност е дебелината да е n, което означава, че всеки два изпита са в конфликт, така че n зали са необходими. И изобщо, $th(T)$ е **очевидна** долна граница за броя на необходимите зали.

Оказва се, че $th(T)$ е точна долна граница за броя на залите. Това ще докажем конктруктивно чрез следния алчен алгоритъм.

НАМЕРИ ЗАЛИ, ОБЩ($T = \{t_1 = (s_1, f_1), t_2 = (s_2, f_2), \ldots, t_n = (s_n, f_n)\}$)
1 сортирай T по времена на стартиране
2 (* За по-просто именуване, допускаме, че $s_1 < s_2 < \cdots < s_n$ *)
3 нека $r_1, \ldots, r_n$ са зали, в които може да се провеждат изпити
4 $Q \leftarrow \emptyset$, count $\leftarrow 0$
5 **for** $i \leftarrow 1$ **to** n
6 ако в Q има зала, в която може да се проведе t_i, разпределяме t_i в нея
7 инак, ангажираме нова зала r_j, разпределяме t_i в нея, $Q \leftarrow Q \cup \{r_j\}$ и count++
8 **return** count

Този псевдокод е прекалено общ (откъдето и името), за да е решението на задачата, но по него можем да направим доказателството за коректност, а после ще детайлизираме имплементацията, за да анализираме сложността.

Да кажем, че T^i е редицата от изпити $(t_1, \ldots, t_i)$ след сортирането на ред 1. Следното твърдение е инвариант за цикъла.

При всяко достигане на ред 5:

❶ *за всяка зала в Q, изпитите, разпределени в нея, са два по два безконфликтни,*

❷ $th(T^{i-1}) = count = |Q|$.

Ето доказателство. Базата е първото достигане на ред 5. В този момент $Q = \emptyset$, count $= 0$ и $i = 1$. Част ❶ от инварианта е вярна в празния смисъл. Част ❷ също е вярна, защото T^0 е празната редица, а нейната дебелина е 0, а от друга страна, count съдържа 0 и Q е празното множество, така че наистина $|Q| =$ count в този момент.

Да допуснем, че инвариантът е в сила при някое достигане на ред 5, което не е последното. Разглеждаме следните две възможности, изчерпателни и взаимно изключващи се.

- Има зала r_k в Q, такава че t_i е безконфликтен с всички изпити, които досега са разпределени в нея. Това означава, че има зала в Q, а именно r_k, такава че t_i може да се проведе в нея заедно с всички други изпити, които досега са разпределени в нея. Тогава условието на ред 6 е истина и t_i се разпределя в зала r_k. Да се убедим, че инвариантът остава в сила.

  1. Съгласно предположението, разпределението на изпитите преди слагането на t_i в r_k е безконфликтно. Но t_i се слага в зала r_k, с чиито досега сложени изпити той (t_i) няма конфликти. Следователно, след слагането на t_i в r_k продължава да е вярно, че за всяка зала в Q, изпитите, разпределени в нея, са два по два безконфликтни.
  2. Тъй като нова зала не се назначава, $|Q|$ се запазва и count се запазва (ред 6). Остава да покажем, че $\mathtt{th}(T^{i-1}) = \mathtt{th}(T^i)$. Но това е (почти) очевидно предвид факта, че t_i е безконфликтен с всеки изпит, назначен в зала r_k, и допускането, че във всяка друга зала в Q, всички изпити, сложени в нея до момента, са безконфликтни. Оттук и ключовото наблюдение, че всяка друга зала може да допринесе най-много единица към $\mathtt{th}(T^i)$ с добавянето на изпита t_i, а тези други зали са $\mathrm{th}(T^{i-1}) - 1 = |Q| - 1$ на брой. Следователно, дебелината не нараства с добавянето на t_i, така че $\mathtt{th}(T^i) = \mathrm{count} = |Q|$. След инкрементирането на i на ред 5, отново е вярно, че $\mathtt{th}(T^{i-1}) = \mathrm{count} = |Q|$.

- Във всяка зала в Q има изпит, който е в конфликт с t_i. Това означава, че няма зала в Q, в която да се проведе t_i заедно с изпитите, разпределение досега. Изпълнението отива на ред 7. Да се убедим, че инвариантът остава в сила.

  1. Това е очевидно вярно, понеже t_i се разполага като единствен изпит в новодобавената зала r_j, а в останалите зали изпитите са безконфликтни по допускане.
  2. $|Q|$ нараства с единица при ангажирането на новата зала r_j, но и count нараства с единица на ред 7. Забелязваме, че $\mathtt{th}(T^i) = \mathtt{th}(T^{i-1}) + 1$, защото t_i се оказа в конфликт с поне един изпит във всяка от досега ангажираните зали, а техният брой по допускане е $\mathtt{th}(T^{i-1})$. Заключаваме, че $\mathtt{th}(T^i) = \mathrm{count} = |Q|$. След инкрементирането на i на ред 5 отново е вярно, че $\mathtt{th}(T^{i-1}) = \mathrm{count} = |Q|$.

С което инвариантът е доказан. Оттук следва, че при последното достигане на ред 5 е вярно, че за всяка зала в Q, изпитите, разпределени в нея, са два по два безконфликтни, и $\mathtt{th}(T) = \mathrm{count}$. Ерго, на ред 7, алгоритъмът връща именно $\mathtt{th}(T)$. Както вече забелязахме, $\mathtt{th}(T)$ е долна граница за броя на залите, така че алгоритъмът връща оптималната стойност за брой на зали, в които да се проведат изпитите без конфликти.

Сега да разгледаме ефикасна имплементация на този общо описан алгоритъм. Пре-сортирането става във време $\Theta(n \lg n)$ и това е оптимално, както знаем от лекции. Да помислим как да проверяваме ефикасно за всеки новоразгледан изпит дали може да бъде сложен във вече ангажирана зала или се налага да му ангажираме нова зала (ред 5 в общия код). Лесно се вижда, че t_i е безконфликтен с изпитите, сложени в дадена зала тстк $f' < s_i$, където f' е максималното време на приключване на изпит в тази зала. Ерго, за всяка ангажирана зала е достатъчно да помним максималното време на приключване на изпит в нея. И ни интересува минималното такова време, по всички досега ангажирани зали – него сравняваме със s_i и, ако то се окаже по-малко, слагаме t_i в съответната зала, като обновяваме максималното време за приключване на изпит в нея с f_i, а в противен случай ангажираме нова зала и придвояваме f_i на максималното време за приключване на изпит в нея. За да не разглеждаме на всяка итерация числата на всички зали в Q, ще използваме приоритетна опашка от тип min. Ключовете на елементите в нея са времената на приключване на изпитите. Ето код.

```
НАМЕРИ ЗАЛИ(T = {t_1 = (s_1, f_1), t_2 = (s_2, f_2), ..., t_n = (s_n, f_n)})
 1  сортирай T по времена на стартиране
 2  направи празна приоритетна опашка Q от тип min, реализирана с двоична пирамида
 3  сложи f_1 в Q
 4  count ← 1
 5  for i ← 2 to n
 6      x ← MIN(Q)
 7      if x < s_i
 8          INCREASEKEY(Q, x, f_i)
 9      else
10          INSERT(Q, f_i)
11          count++
12  return count
```

Има смисъл да започнем от втория изпит, за да не се опитваме да търсим минимума на празна опашка. Няма смисъл да викаме ExtractMin, защото зали не се деангажират в този контекст.

Сложността по време на цялото изпълнение на цикъле е $O(n \lg n)$, ако опашката е реализирана с пирамида. Практически същото нещо е доказвано на лекции. Сумирано с $\Theta(n \lg n)$ от предварителното сортиране, сложността по време става $\Theta(n \lg n)$.

Що се отнася до бонуса, графовата задача, която решихме, е намирането на хроматичното число на интервален граф. Тази сложност по време е достижима само ако не се опитваме да представяме графа "истински" с матрица или списъце на съседство, а пуснем алгоритъма директно върху интервалите, които представят графа.

Зад. 4 Дадени са n човека h_1, h_2, ..., h_n. За всеки двама различни човека h_i и h_j е вярно точно едно от трите:

- h_i следи h_j, но h_j не следи h_i,
- нито единият от двамата не следи другия, но е възможно единият да започне да следи другия, като обаче другият не следи първия: ако h_i започне да следи h_j, не може h_j да следи h_i, а ако h_j започне да следи h_i, не може h_i да следи h_j,
- не е разрешено нито h_i да следи h_j, нито h_j да следи h_i.

И, разбира се, никой не следи себе си.
Известно е, че двойките, които се следят, са такива, че за всяко $k \geq 3$ **не** съществува множество индекси $\{i_1, i_2, \ldots, i_k\} \subseteq \{1, 2, \ldots, n\}$, такова че h_{i_1} следи h_{i_2}, h_{i_2} следи h_{i_3}, и така нататък, $h_{i_{k-1}}$ следи h_{i_k}, h_{i_k} следи h_{i_1}.
Задачата е тази: предложете ефикасен алгоритъм, който при вход

- дадено множество от хора,
- дадени двойки, първият от които следи втория,
- и дадени двойки, за които е възможно единият да следи другия,

изчислява за всяка двойка, за която е възможно единият да следи другия, точно кой кого да следи. При това гореспоменатото ограничение да остане в сила, а именно за всяко $k \geq 3$ да **не** съществува множество индекси $\{i_1, i_2, \ldots, i_k\} \subseteq \{1, 2, \ldots, n\}$, такова че h_{i_1} следи h_{i_2}, h_{i_2} следи h_{i_3}, и така нататък, $h_{i_{k-1}}$ следи h_{i_k}, h_{i_k} следи h_{i_1}. Анализирайте коректността и сложността по време.

Решение: Това е задача върху графи. Даден е граф, някои ребра на който са ориентирани (двойките, единият от които следи другия), а останалите ребра са неориентирани (двойките, единият от които може да започне да следи другия). Ориентираните ребра индуцират даг: няма цикли с дължина 1, защото никой не следи себе си, няма цикли с дължина 2, защото, ако някой следи друг, другия не следи първия, и няма цикли с дължина ≥ 3 заради даденото ограничение.

Иска се алгоритъм, който ефикасно намира такава ориентация на всяко от неориентираните ребра, че графът да остане ацикличен. Конструирането на такъв алгоритъм е обсъждано на лекции: топологически се сортира дагът, индуциран от ориентираните ребра, и след това на всяко неориентирано ребро (u, v) се дава ориентация от u към v тстк u предшества v в топо-сортировката.

Коректността е очевидна: знаем, че всеки даг има топо-сортировка, и даването на ориентация на неориентираните ребра по гореописания начин не може да доведе до цикли. Сложността по време очевидно е $\Theta(n + m)$, където n е броят на хората, а m е броят на двойките от следящи се хора и хора, за които е възможно да се следят.