Поправителен изпит по ДАА, 08. 02. 2016 г., СУ, ФМИ

Име: ______________________________ ФН: ______ Спец.: ___ Курс: ___

Задача	1	2	3	4	5	6	Общо
получени точки							
максимум точки	20	20	30	20	20	20	130

Забележка: За отлична оценка са достатъчни 100 точки!

Задача 1. Решете следните рекурентни уравнения:

а) $T(n) = \sqrt{3}\, T\left(\dfrac{n}{2}\right) + n$;

б) $T(n) = 2\, T\left(\dfrac{n}{\sqrt{3}}\right) + n \lg n$;

в) $T(n) = T(n-1) + \lg n$;

г) $T(n) = n^2 + 2\sum\limits_{i=0}^{n-1} T(i)$.

Задача 2. Някои от компютрите в една мрежа са свързани с комуникационни канали. Всеки от каналите се характеризира с определена надеждност — число от интервала $(0\,;\,1)$, което показва вероятността за безпроблемно предаване на съобщение по канала. Предложете бърз алгоритъм, който намира най-надеждния път между два дадени компютъра от мрежата. (Надеждността на пътя е <u>произведението</u> от надеждностите на съставящите го канали.)

Задача 3. Съставете алгоритъм, който по зададено цяло положително число n намира най-малкия брой точни квадрати със сбор n. (20 точки)
Разширете алгоритъма така, че да отпечатва представянето на n като сбор на най-малък брой точни квадрати. (10 точки)
Демонстрирайте алгоритъма при $n = 10$. Анализирайте сложността по време и по памет за $\forall n$.

Задача 4. Масивите $A\left[1 \ldots n^2\right]$ и $B\left[1 \ldots n^3\right]$ съдържат цели числа и са сортирани. Предложете бърз алгоритъм, който изчислява колко различни числа са общи за двата масива. Опишете алгоритъма с думи, не е нужен псевдокод.
Забележка: Обърнете внимание, че масивът B съдържа много повече елементи от масива A.

Задача 5. В магазин за сувенири има n артикула с цени $a_1, a_2, \ldots, a_n$ лева. Искаме да купим коледни подаръци на близките си, но имаме само M лева. Целта ни е да зарадваме с подаръци възможно най-много хора. Предложете бърз алгоритъм за избор на подаръци.

Задача 6. Задачата Клика остава ли NP-пълна за двуделни графи?
Ако да — предложете подходяща редукция. Ако не — съставете бърз алгоритъм.

РЕШЕНИЯ

Задача 1.

а) Прилагаме мастър-теоремата: $k = \log_b a = \log_2 \sqrt{3} < 1$, защото $\sqrt{3} < 2 = 2^1$. Сравняваме n^k с $f(n) = n$. От $k < 1$ следва, че $\exists \varepsilon > 0$, за което $n^{k+\varepsilon} \prec n$, например $\varepsilon = \dfrac{1-k}{2} \cdot$ Попадаме в третия случай на мастър-теоремата, търсим $c \in (0\,;\,1)$, за което $cf(n) \geq \sqrt{3}\, f\left(\dfrac{n}{2}\right)$, т.е. $c\,n \geq \sqrt{3}\,\dfrac{n}{2}$, за достатъчно големи n. Неравенството е вярно за $c \in \left[\dfrac{\sqrt{3}}{2}\,;\,1\right)$, напр. $c = 0,9$. Следователно $T(n) = \Theta(n)$.

б) Прилагаме мастър-теоремата: $k = \log_{\sqrt{3}} 2 > 1$ и сравняваме n^k с $f(n) = n \lg n$. От $k > 1$ следва, че $\exists \varepsilon > 0$, за което $n^{k-\varepsilon} \succ n \log n$, например $\varepsilon = \dfrac{k-1}{2} \cdot$ Попадаме в първия случай на мастър-теоремата, следователно $T(n) = \Theta\left(n^{\log_{\sqrt{3}} 2}\right)$.

в) След развиване на уравнението се получава $T(n) = T(0) + \lg 1 + \lg 2 + \ldots + \lg n$. Сборът на логаритмите се оценява асимптотично чрез интеграл от $\lg x$ или се свежда до известната асимптотична оценка $\log(n!) = \Theta(n \log n)$. Окончателно, $T(n) = \Theta(n \log n)$.

г) Изваждаме равенствата, дефиниращи $T(n)$ и $T(n-1)$:

$$T(n) = n^2 + 2\sum_{i=0}^{n-1} T(i),$$

$$T(n-1) = (n-1)^2 + 2\sum_{i=0}^{n-2} T(i).$$

Получаваме $T(n) - T(n-1) = 2\,T(n-1) + 2n - 1$, т.е. $T(n) = 3\,T(n-1) + 2n - 1$, което решаваме чрез характеристично уравнение. Хомогенната част поражда уравнението $x = 3$ с мултимножество от корени $\{3\}_{\text{М}}$. Нехомогенната част поражда мултимножеството $\{1\,,\,1\}_{\text{М}}$. Като обединим двете мултимножества, получаваме пълния списък от корени: $\{1\,,\,1\,,\,3\}_{\text{М}}$. Следователно $T(n) = C_1 + C_2\, n + C_3\,.\,3^n = \Theta\left(3^n\right)$, тъй като последното събираемо расте най-бързо.

Задача 2. Моделираме компютърната мрежа чрез тегловен граф:
— върховете на графа съответстват на компютрите;
— ребрата съответстват на комуникационните канали;
— теглото на всяко ребро го полагаме да бъде равно на $w = -\ln p$, където p е надеждността на съответния канал, а логаритъмът е натурален (всъщност основата може да бъде всяко число, по-голямо от 1).

От $p < 1$ следва, че $\ln p < 0$, откъдето $w = -\ln p > 0$, т.е. теглата на ребрата са положителни, следователно могат да се интерпретират като дължини. Логаритмуването свежда умножението на надеждности до събиране на техните логаритми. Така теглото на път е равно на сбора от теглата на неговите ребра. Следователно теглото на път също може да се интерпретира като дължина на пътя (поради цитираното адитивно свойство). Заради знака минус теглото w е намаляваща функция на надеждността p: колкото по-къс е пътят, толкова по-надежден е. Дадената задача се свежда до задачата за намиране на най-къс път в граф с положителни тегла на ребрата. Подходящ за този случай е алгоритъмът на Дейкстра.

Задача 3 може да се реши чрез динамично програмиране. Таблицата представлява едномерен масив $\operatorname{dyn}[0 \ldots n]$ от цели числа, като $\operatorname{dyn}[k]$ е най-малкият брой точни квадрати със сбор k. Масивът се попълва с помощта на началното условие $\operatorname{dyn}[0] = 0$ и рекурентната формула $\operatorname{dyn}[k] = 1 + \min\limits_{i} \left\{ \operatorname{dyn}\left[k - i^2\right] \right\}$, $k > 0$, където минимумът е по всички $i = 1, 2, 3, \ldots, \lfloor \sqrt{k} \rfloor$.

```
SQUARES(n: positive integer)
 1  dyn[0...n]: array of positive integers;
 2  // dyn[k] = най-малкият брой точни квадрати със сбор k.
 3  addend[1...n]: array of positive integers;
 4  // addend[k] = най-малкото събираемо в представянето на k
 5  //             като сбор на минимален брой точни квадрати.
 6  dyn[0] ← 0
 7  for k ← 1 to n
 8      dyn[k] ← k + 1
 9      i ← 1
10      j ← 1
11      while j ≤ k do
12          if dyn[k − j] < dyn[k]
13              dyn[k] ← dyn[k − j]
14              addend[k] ← j
15          i ← i + 1
16          j ← i × i
17      dyn[k] ← dyn[k] + 1
18  k ← n
19  while k > 0 do
20      j ← addend[k]
21      print j
22      k ← k − j
23  return dyn[n]
```

Демонстрация на работата на алгоритъма при $n = 10$:

k	0	1	2	3	4	5	6	7	8	9	10
dyn	0	1	2	3	1	2	3	4	2	1	2
addend		1	1	1	4	1	1	1	4	9	1

Таблицата се попълва отляво надясно. Например последната колонка е получена тъй:
$$\operatorname{dyn}[10] = 1 + \min \left\{ \operatorname{dyn}\left[10 - 1^2\right] ;\ \operatorname{dyn}\left[10 - 2^2\right] ;\ \operatorname{dyn}\left[10 - 3^2\right] \right\} =$$
$$= 1 + \min \left\{ \operatorname{dyn}[9] ;\ \operatorname{dyn}[6] ;\ \operatorname{dyn}[1] \right\} = 1 + \min \left\{ 1 ;\ 3 ;\ 1 \right\} = 1 + 1 = 2,$$
което означава, че за числото 10 са нужни два квадрата.

Най-малкият елемент на мултимножеството $\left\{ 1 ;\ 3 ;\ 1 \right\}$ е числото 1, което се среща два пъти и съответства на събираемите 1^2 и 3^2. Без значение е кое от събираемите ще вземем. Така, както алгоритъмът е оформен по-горе (със строго неравенство на ред № 12), той взима по-малкото събираемо, затова $\operatorname{addend}[10] = 1^2 = 1$. (Ако на ред № 12 има нестрого неравенство, тогава в addend ще се пази по-голямото събираемо. В такъв случай алгоритъмът ще работи вярно, но по-бавно: редове № 13 и № 14 ще се изпълняват излишно.)

Самото представяне на 10 като сбор от точни квадрати се получава с помощта на масива addend. Тръгваме от колонката $k = 10$ и намираме addend $= 1$, което е най-малкото събираемо в сбора. Изваждаме това събираемо от сумата и остава $10 - 1 = 9$. От колонката $k = 9$ намираме следващото събираемо: addend $= 9$. Изваждаме това събираемо от оставащата сума: $9 - 9 = 0$. Не остава нищо, така че алгоритъмът приключва работа. Намерено е представянето $10 = 1 + 9$.

Анализ на сложността на алгоритъма:

Най-голямо количество допълнителна памет изразходват масивите dyn и addend, затова те определят сложността по памет: $\Theta(n)$.

Сложността по време зависи от броя итерации на циклите. Цикълът на редове № 11 – № 16 се изпълнява, докато $j = i^2 \leq k$, т.е. за $i = 1, 2, 3, \ldots, \lfloor \sqrt{k} \rfloor$, което прави $\lfloor \sqrt{k} \rfloor$ итерации, а това по порядък е равно на $\sqrt{k}$. Следователно сложността на цикъла на редове № 7 – № 17 е равна по порядък на $\sum\limits_{k=1}^{n} \sqrt{k} = \sum\limits_{k=1}^{n} k^{0,5} = \Theta\left(n^{1,5}\right) = \Theta\left(n\sqrt{n}\right)$.

Тялото на цикъла на редове № 19 – № 22 се изпълнява $\mathrm{O}(n)$ пъти, защото на всяка итерация стойността на k намалява поне с една единица. Останалите команди (ред № 6 и ред № 23) изразходват константно време, затова не влияят на порядъка на сложността.

Времето на алгоритъма е сбор от времената на частите му: $\Theta\left(n\sqrt{n}\right) + \mathrm{O}(n) = \Theta\left(n\sqrt{n}\right)$. Окончателно, времевата сложност на алгоритъма е $\Theta\left(n\sqrt{n}\right)$.

Задача 4. Идеята е да обходим единия масив последователно и всяка стойност, която е различна от вече срещнатите, да я търсим двоично в другия масив.

Проверката, дали текущата стойност е различна от вече срещнатите, е лесна, тъй като масивите са сортирани: достатъчно е да сравним текущия елемент с предходния.

Ако обхождаме последователно масива A, а извършваме двоично търсене в масива B, то сложността на получения алгоритъм ще бъде $\Theta\left(n^2 \log\left(n^3\right)\right) = \Theta\left(n^2 \,.\, 3 \log n\right) = \Theta\left(n^2 \log n\right)$.

Ако обхождаме последователно масива B, а извършваме двоично търсене в масива A, то сложността на получения алгоритъм ще бъде $\Theta\left(n^3 \log\left(n^2\right)\right) = \Theta\left(n^3 \,.\, 2 \log n\right) = \Theta\left(n^3 \log n\right)$.

Тъй като $n^2 \log n \prec n^3 \log n$, то първият вариант е по-добър. Тоест по-добре е да обходим последователно масива A и всяка срещната различна стойност да я търсим двоично в масива B.

INTERSECTION($A[1 \ldots n^2]$, $B[1 \ldots n^3]$: sorted arrays of integers)

```
if BinarySearch(B, A[1])
    print A[1]
for k ← 2 to n^2
    if A[k] ≠ A[k − 1]
        if BinarySearch(B, A[k])
            print A[k]
```

Както беше установено по-горе, времевата сложност на този алгоритъм е $\Theta\left(n^2 \log n\right)$ в най-лошия случай. Алгоритъмът може да се оптимизира малко: след всяко двоично търсене да се запомня позицията от B, на която е бил намерен текущият елемент $A[k]$, и следващото двоично търсене да се извършва в частта от B, която е надясно от запомнената позиция. Тази оптимизация наистина ускорява алгоритъма, но не по порядък. Може да се докаже, че всеки алгоритъм, основан на сравнения, който решава тази задача, изисква време $\Omega\left(n^2 \log n\right)$. Тоест предложеният алгоритъм е най-бърз (по порядък).

Задача 5. За да купим възможно най-много подаръци, трябва да изберем най-евтините. Сортираме подаръците по цени, след което, като започнем от най-евтиния, последователно купуваме всички подаръци, за които ни стигнат парите.

Сортирането изисква време $\Theta(n \log n)$, а самото купуване — време $\Theta(n)$. Следователно времевата сложност на целия алгоритъм е $\Theta(n \log n) + \Theta(n) = \Theta(n \log n)$ в най-лошия случай.

Задачата може да се реши и по-бързо: за линейно време $\Theta(n)$. Това става с помощта на алгоритъма PICK за търсене на медиана, който има линейна времева сложност в най-лошия случай (https://people.csail.mit.edu/rivest/pubs/BFPRT73.pdf). Като знаем медианата, разделяме сувенирите на две равни по големина групи — скъпи и евтини сувенири. (Ако n е нечетно, едната група ще съдържа един сувенир повече от другата.) Пресмятаме общата цена S на евтините сувенири.

Ако $S = M$, купуваме евтините сувенири и приключваме с пазаруването.

Ако $S < M$, купуваме евтините сувенири и разполагаме с още $M - S$ лева, с които може би ще успеем да купим някои от скъпите сувенири. Изпълняваме рекурсивно същия алгоритъм върху множеството на скъпите сувенири с разполагаем паричен ресурс $M - S$.

Ако $S > M$, не можем да купим всички евтини сувенири и можем да забравим за скъпите. Изпълняваме рекурсивно алгоритъма върху множеството на евтините сувенири с първоначалния паричен ресурс M.

Анализ на алгоритъма: Нека $T(n)$ е времевата сложност в най-лошия случай. Тогава $T(n) = T\left(\frac{n}{2}\right) + \Theta(n)$, защото рекурсивното извикване е върху половината масив, а всички други операции — търсенето на медианата на цените, разделянето на сувенирите на скъпи и евтини и пресмятането на S — изразходват линейно време. Решаваме полученото рекурентно уравнение чрез мастър-теоремата и намираме $T(n) = \Theta(n)$.

Откриването на второто решение не е задължително. Първият алгоритъм е достатъчно бърз и носи пълен брой точки.

Задача 6. Алгоритмичната задача Клика приема като входни данни един граф $G(V, E)$ и едно цяло положително число k и отговаря на въпроса дали графът G съдържа клика от ред k. От теорията е известно, че когато графът G е произволен, тази алгоритмична задача е NP-пълна.

Но ако G е двуделен граф, задачата Клика вече не е NP-пълна (при условие че $\mathrm{P} \neq \mathrm{NP}$), тъй като за нея в този случай съществува алгоритъм с полиномиална времева сложност:

1) Ако $k \geq 3$, то няма клика от ред k, защото всяка такава клика съдържа триъгълник, а в двуделен граф не може да има цикъл с нечетна дължина.
2) Ако $k = 1$, то има клика от ред k — кой да е връх на графа G (смятаме, че $V \neq \varnothing$).
3) Остава случаят $k = 2$; в този случай има клика от ред k точно тогава, когато графът G съдържа поне едно ребро, т.е. когато $E \neq \varnothing$.

Сравненията на k с 1 и с 3 работят в константно време. Проверката, дали $E \neq \varnothing$, изисква обхождане на графа, т.е. работи в линейно време. Затова времевата сложност на алгоритъма е линейна, следователно полиномиална, което трябваше да се докаже.