Изпит по "Дизайн и анализ на алгоритми" (поправителна сесия), СУ, ФМИ, 24. 08. 2016 г.

Име: __________________________________ ФН: _____ Спец.: ____ Курс: ___

Задача	1	2	3	4	5	6	Общо
получени точки							
максимум точки	20	20	20	20	20	20	120

Забележка: За отлична оценка са достатъчни 100 точки.

Задача 1. Решете следните рекурентни уравнения:

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

б) $T(n) = T(n-1) + \dfrac{1+n}{n^2}$;

в) $T(n) = \displaystyle\sum_{i=0}^{n-1} T(i) + 2^{\frac{n}{2}}$;

г) $T(n) = 5\,T\left(\dfrac{n}{2}\right) + n^2 \lg n$.

Задача 2. Масивът $A\big[1\,,\,2\,,\,\ldots\,,\,n\big]$ съдържа m инверсии. *Инверсия* е всяка двойка индекси $(i\,,\,j)$, такива, че $1 \leq i < j \leq n$ и $A\left[\,i\,\right] > A\left[\,j\,\right]$. Докажете, че алгоритъмът InsertionSort сортира A за $\Theta(m+n)$ стъпки.

Задача 3. Даден е ориентиран граф $G(V,E)$. Докажете или опровергайте всяко от следните твърдения:

а) Ако G съдържа само една силно свързана компонента, то в G има хамилтонов цикъл.

б) Ако в G има хамилтонов цикъл, то G съдържа само една силно свързана компонента.

Задача 4. Нека $B\big[1\ldots n\big]$ е масив от крайни множества, сортиран по мощност на множествата. Предложете бърз алгоритъм, който намира най-дългата верига в B, т.е. отпечатва редица от индекси $k_1\,,\,k_2\,,\,\ldots\,,\,k_L$, за които $1 \leq k_1 < k_2 < \ldots < k_L \leq n$, $B\left[k_1\right] \subseteq B\left[k_2\right] \subseteq \ldots \subseteq B\left[k_L\right]$ и L е максимално.

Задача 5. Даден е неориентиран граф $G(V,\,E)$. Предложете бърз алгоритъм, който разпознава дали множеството от върхове може да бъде разбито по <u>единствен</u> начин на множества V_1 и V_2 така, че всяко ребро свързва връх от V_1 и връх от V_2.

Задача 6. Разглеждаме алгоритмичната задача за разпознаване MaxSAT:
Дадено: конюнктивна нормална форма F с n клаузи и цяло число k от 1 до n.
Търси се: съществува ли присвояване на логически стойности на променливите, което удовлетворява поне k от клаузите на F ?

Докажете, че задачата MaxSAT е NP-трудна.

РЕШЕНИЯ

Задача 1. Рекурентните уравнения се решават, както следва:

а) Чрез втория случай на мастър-теоремата: $k = \log_{\sqrt{2}} 2\sqrt{2} = 3, \quad T(n) = \Theta\left(n^3 \log n\right)$.

б) Чрез развиване: $T(n) = T(0) + \sum\limits_{k=1}^{n} \dfrac{1+k}{k^2} \asymp \sum\limits_{k=1}^{n} \dfrac{1}{k^2} + \sum\limits_{k=1}^{n} \dfrac{1}{k} \asymp 1 + \log n \asymp \log n$.

в) В уравнението $T(n) = \sum\limits_{i=0}^{n-1} T(i) + 2^{\frac{n}{2}}$ заменяме n със $n-1$

и получаваме ново уравнение: $T(n-1) = \sum\limits_{i=0}^{n-2} T(i) + 2^{\frac{n-1}{2}}$.

От първото уравнение вадим второто: $T(n) - T(n-1) = T(n-1) + 2^{\frac{n}{2}} - 2^{\frac{n-1}{2}}$, т.е.
$T(n) = 2\,T(n-1) + \left(1 - \dfrac{1}{\sqrt{2}}\right)\left(\sqrt{2}\right)^n$. Това линейно-рекурентно уравнение се решава с помощта на характеристично уравнение. Получава се
$T(n) = C_1\, 2^n + C_2 \left(\sqrt{2}\right)^n = \Theta(2^n)$.

г) Чрез първия случай на мастър-теоремата: $k = \log_2 5, \quad T(n) = \Theta\left(n^{\log_2 5}\right)$.

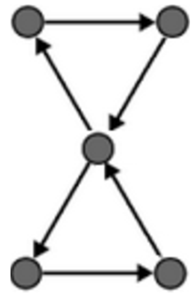
Задача 2. Една възможна реализация на алгоритъма InsertionSort:

```
InsertionSort(A[1...n])
1  for i ← 2 to n
2      j ← i − 1
3      while j > 0 and A[j] > A[j+1] do
4          swap (A[j], A[j+1])
5          j ← j − 1
```

Всяко изпълнение на ред № 4 намалява броя на инверсиите с единица, следователно тялото на цикъла **while** се изпълнява точно m пъти общо за цялото изпълнение на InsertionSort. За всяка стойност на i ред № 3 се изпълнява, докато има инверсии в подмасива $A[1 \dots i]$, и още веднъж, когато инверсиите свършат, при излизане от цикъла **while**. Общо за цялата сортировка ред № 3 се изпълнява $m + n - 1$ пъти. Редове № 1 и № 2 се изпълняват $n - 1$ пъти.

Сумираме горните оценки и получаваме време за изпълнение $\Theta(m+n)$.

Задача 3.



а) Твърдението не е вярно. Графът, показан на чертежа, съдържа само една компонента на силна свързаност, но въпреки това не е хамилтонов.

б) Твърдението е вярно. Хамилтоновият цикъл минава през всички върхове, затова за всеки два върха A и B можем, движейки се по цикъла, да стигнем от A до B и от B до A. Тоест всеки два върха са силно свързани, следователно целият граф е една силно свързана компонента.

Задача 4 може да се реши с помощта на динамично програмиране. За тази цел използваме два целочислени масива $\operatorname{dyn}[1 \ldots n]$ и $\operatorname{prev}[1 \ldots n]$. В тях пазим следната информация:

$\operatorname{dyn}[k]$ = дължината на най-дългата растяща по включване редица сред първите k множества от фамилията B, при условие че последното множество от редицата е $B[k]$;

$\operatorname{prev}[k]$ = индекса (от 1 до n) на предпоследното множество от споменатата най-дълга редица (нула, ако редицата съдържа само едно множество).

Тези масиви се попълват така:

$\operatorname{dyn}[k] = 1 + \max\limits_{i} \left\{ \operatorname{dyn}[i] : \ 1 \leq i \leq k-1, \ B[i] \subseteq B[k] \right\}$, като $\max \varnothing$ се смята за 0;

$\operatorname{prev}[k]$ = онова i, за което се достига максимумът (нула, ако множеството е празно).

Най-големият елемент на масива dyn е дължината на търсената най-дълга редица, а неговият индекс е номерът на последното множество от редицата.

```
LongestChain(B[1...n])
dyn[1...n]: array of integers
prev[1...n]: array of integers
maxLength ← 0
bestLastSet ← 0
// търсене на най-дългата растяща редица:
for k ← 1 to n
    dyn[k] ← 0
    for i ← 1 to k − 1
        if B[i] ⊆ B[k] and dyn[i] > dyn[k]
            dyn[k] ← dyn[i]
            prev[k] ← i
    dyn[k] ← dyn[k] + 1
    if dyn[k] > maxLength
        maxLength ← dyn[k]
        bestLastSet ← k
// отпечатване на най-дългата растяща редица:
Indices ← empty list
while bestLastSet > 0 do
    add bestLastSet to the beginning of Indices
    bestLastSet ← prev[bestLastSet]
print Indices
return maxLength
```

Сложността на този алгоритъм по време е $\Theta\left(n^2\right)$, а по памет е $\Theta(n)$.

Задача 5. Използваме търсене в ширина, като проверяваме дали графът е двуделен и свързан едновременно. Сложността по време и по памет е $\Theta(m+n)$.

Задача 6. NP-трудната алгоритмична задача SAT е частен случай на MaxSAT при $k = n$. Редукцията е полиномиална, защото присвояването $k = n$ се изпълнява за константно време. Щом задачата SAT е NP-трудна, то и нейното обобщение MaxSAT също е NP-трудна задача.