Изпит по "Дизайн и анализ на алгоритми" (поправка — СУ, ФМИ, юли 2016 г.)

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

Задача	1а	1б	1в	2а	2б	3а	3б	4а	4б	5	Общо
получени точки											
максимум точки	5	5	10	10	10	20	10	20	20	20	130

Забележка: За отлична оценка са достатъчни 100 точки.

Задача 1. Да се решат рекурентните уравнения с точност до порядък:

а) $T(n) = T(n-1) + \sqrt[7]{n^{25}}$; б) $T(n) = 625\, T\left(\dfrac{n}{5}\right) + 6n^3$; в) $T(n) = \sum\limits_{k=1}^{n-1} k \,.\, T(k)$.

Задача 2. В небостъргач има няколко асансьора, всеки от които спира само на някои етажи. На един етаж може да спират няколко асансьора. Всеки асансьор може да вози и нагоре, и надолу. Предложете бърз алгоритъм, намиращ маршрут от един етаж до друг, ако искаме да стигнем:

а) за най-малко време (всички асансьори се движат с еднаква скорост);

б) с най-малък брой прекачвания.

Задача 3. Пътник трябва да стигне от град C_0 до град C_n, като мине през всеки от градовете C_1 , C_2 , ... , C_{n-1} непременно в този ред. За всеки участък от маршрута пътникът може да избира между две транспортни компании. Превозът от C_{k-1} до C_k $(k = 1, 2, \ldots, n)$ струва A_k лева с първата компания и X_k лева с втората. Двете компании имат по-ниски цени за продължение на пътуването: съответно B_k лева с първата компания и Y_k лева с втората; $B_k < A_k$, $Y_k < X_k$ $(k = 2, 3, \ldots, n)$. "Продължение" значи, че в участъка от C_{k-2} до C_{k-1} и в участъка от C_{k-1} до C_k пътникът ползва услугите на един и същи превозвач.

а) Съставете алгоритъм `OptimalTransport` $\left(A[1\ldots n],\ B[2\ldots n],\ X[1\ldots n],\ Y[2\ldots n]\right)$, който за време $\Theta(n)$ намира най-ниската цена за пътуване от C_0 до C_n .

б) Разширете алгоритъма така, че да казва с кой превозвач да бъде изминат всеки участък, та общата цена да бъде възможно най-ниска.

Упътване: Използвайте динамично програмиране с числова таблица `dyn`$[1\ldots n][1\ldots 2]$, където `dyn`$[k][i]$ е най-ниската възможна цена за пътуване от C_0 до C_k, ако последният участък от пътя (т.е. от C_{k-1} до C_k) бъде пропътуван с i-тата компания.

Задача 4. Дадени са три масива от цели положителни числа $L[1\ldots n]$, $W[1\ldots n]$ и $H[1\ldots n]$, които съдържат размерите на n играчки с формата на правоъгълен паралелепипед; тоест k-тата играчка има размери $L[k] \times W[k] \times H[k]$. Известно е, че играчките могат да бъдат вложени една в друга като редица от матрьошки, но не непременно в същия ред, в който са дадени техните размери.

а) Предложете алгоритъм с времева сложност $O(n \log n)$, който подрежда играчките в реда на тяхното влагане (първа е най-вътрешната играчка, последна — най-външната).

б) Докажете, че всеки алгоритъм, който подрежда играчките в реда на тяхното влагане, изисква време $\Omega(n \log n)$.

Задача 5. Да се докаже, че е NP-трудна следната алгоритмична задача:
"За даден граф G и дадено цяло положително число k да се разпознае дали G притежава покриващо дърво, всички върхове на което имат степени, ненадвишаващи k."

РЕШЕНИЯ

Задача 1. а) Развиваме уравнението: $T(n) = T(0) + 1^{25/7} + 2^{25/7} + \ldots + n^{25/7} \asymp n^{32/7}$.

б) С мастър-теоремата: $k = \log_5 625 = 4$, $n^{k-\varepsilon} \succ 6n^3$, напр. $\varepsilon = 0{,}01$. Значи, $T(n) = \Theta\left(n^4\right)$.

в) $T(n) = \sum\limits_{k=1}^{n-1} k \,.\, T(k)$. Заместваме n със $n-1$: $T(n-1) = \sum\limits_{k=1}^{n-2} k \,.\, T(k)$. Това уравнение го вадим от оригиналното: $T(n) - T(n-1) = (n-1) \,.\, T(n-1)$, тоест $T(n) = n \,.\, T(n-1)$. Развиваме полученото уравнение: $T(n) = n(n-1) \,.\, T(n-2) = \ldots = n!\, T(0) = \Theta(n!)$.

Задача 2. Разглеждаме ненасочен мултиграф, чиито върхове са номерата на етажите. Между връх № i и връх № j има ребро тогава и само тогава, когато съществува асансьор, който вози от етаж № i до етаж № j. Асансьорите возят в двете посоки, затова мултиграфът е ненасочен. За всяко ребро дефинираме тегло — разстоянието между етажите. По-конкретно, ако реброто е между връх № i и връх № j, то теглото на реброто е $|\, i - j \,|$.

а) Щом всички асансьори се движат с еднаква скорост, то времето за изминаване на път е правопропорционално на дължината му, която е равна на сбора от теглата на ребрата. Търси се най-къс път в мултиграф с неотрицателни тегла на ребрата. Подходящ за този случай е алгоритъмът на Дейкстра.

б) Броят на прекачванията е равен на броя на ребрата минус едно. Пак търсим най-къс път, но сега дължината на пътя е равна на броя на неговите ребра, т.е. теглата не играят роля. Подходящо за този случай е търсенето в ширина.

Задача 3. За краткост на кода предполагаме, че функцията `min` връща наредена двойка, чийто първи елемент е по-малката от двете стойности, а втори елемент е поредният ѝ номер. С други думи, `min(r,s)` връща `(r,1)`, ако `r < s`, и `(s,2)` — в противен случай.

```
OptimalTransport ( A[1...n] , B[2...n] , X[1...n] , Y[2...n] )
dyn[1...n][1...2]: array of numbers   // цени на най-евтин превоз
previous[2...n][1...2]: array of numbers   // предишен превозвач (№ 1 или № 2)
dyn[1][1] ← A[1]
dyn[1][2] ← X[1]
for k ← 2 to n
    ( dyn[k][1] , previous[k][1] ) ← min ( dyn[k−1][1] + B[k] , dyn[k−1][2] + A[k] )
    ( dyn[k][2] , previous[k][2] ) ← min ( dyn[k−1][1] + X[k] , dyn[k−1][2] + Y[k] )
// p = най-ниската възможна цена на пътуването
// i = номер на превозвач в текущия участък от пътя
( p , i ) ← min ( dyn[n][1] , dyn[n][2] )
// отпечатваме избраните превозвачи в обратен ред
for k ← n downto 2
    print "В участък № " , k , " ползваме превозвач № " , i , "."
    i ← previous[k][i]
print "В участък № " , 1 , " ползваме превозвач № " , i , "."
return p
```

В таблицата `previous` пазим номера на предишния превозвач. По-точно, `previous`$[k][i]$ е превозвачът, с който трябва да пътуваме в $(k-1)$-ия участък от пътя, ако k-тият участък (т.е. от C_{k-1} до C_k) бъде изминат с i-тия превозвач. Тази таблица е излишна в подусловие "а", където не ни интересува списъкът на превозвачите. В този случай можем да премахнем редовете № 2, № 9, № 11, № 12, № 13, № 14 и № 15, а функцията `min` може, както обикновено, да връща само едно число — по-малката от стойностите на аргументите.

Достатъчно е в паметта да се намират само k-тият и $(k-1)$-ият ред от таблицата `dyn`. Това може да се използва за оптимизация на количеството допълнителна памет, но не влияе на времето за изпълнение на алгоритъма: $\Theta(n)$.

Задача 4.

а) За линейно време $\Theta(n)$ пресмятаме обемите на играчките: $V[k] = L[k] \times W[k] \times H[k]$, $k = 1, 2, \ldots, n$. После за време $\Theta(n \log n)$ сортираме играчките по обем; получава се тъкмо желаната наредба: първата играчка има най-малък обем и е най-вътрешна, а последната има най-голям обем и е най-външна. Общото време на алгоритъма е $\Theta(n) + \Theta(n \log n) = \Theta(n \log n) = \mathrm{O}(n \log n)$.

б) Долната граница $\Omega(n \log n)$ се доказва чрез редукция от задачата Sort за сортиране на числов масив $A[1 \ldots n]$, за която същата долна граница е доказана по-рано. А именно: на всяко число $A[k]$ съпоставяме играчка с формата на куб $A[k] \times A[k] \times A[k]$.

```
Sort(A[1...n])
    L[1...n], W[1...n], H[1...n]: arrays of numbers
    for k ← 1 to n
        L[k] ← A[k]
        W[k] ← A[k]
        H[k] ← A[k]
    return SortToys(L, W, H)  // връща пермутацията, която сортира масива
```

Коректността на редукцията следва от това, че кубчетата се влагат едно в друго според наредбата на своите размери: по-малкото кубче се влага в по-голямото.

Цикълът се изпълнява за време от порядък $n \prec n \log n$, т.е. описаната редукция е достатъчно бърза за целите на доказателството.

Задача 5. В частния случай $k = 2$ покриващото дърво представлява хамилтонов път. Редукцията е полиномиална, защото присвояването $k = 2$ се извършва за константно време. Разглежданата алгоритмична задача е обобщение на NP-трудната задача ХамилтоновПът и значи също е NP-трудна.