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

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

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

Забележка: За отлична оценка са достатъчни 100 точки.

Задача 1. Даден е насочен ацикличен граф с n върха и m ребра. Предложете алгоритъм с линейна времева сложност $\Theta(m+n)$ за намиране на хамилтонов път в дадения граф.

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` $\big(A[1\ldots n],\ B[2\ldots n],\ X[1\ldots n],\ Y[2\ldots n]\big)$, 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 $\texttt{dyn}[1\ldots n][1\ldots 2]$, където $\texttt{dyn}[k][i]$ 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. Както е известно, задачата за разпознаване, дали сред n цели числа има равни, изисква време $\Omega(n \log n)$ в общия случай. Каква е времевата сложност на задачата в следните частни случаи:

а) когато всичките n числа са четни?

б) когато всичките n числа са в интервала от $2n$ до $5n$?

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. Извършваме топологично сортиране на графа (чрез обхождане в дълбочина) за време $\Theta(m+n)$. Резултатът е линейна наредба на върховете: v_1 , v_2 , … , v_n . Още веднъж обхождаме върховете (в този ред) и проверяваме има ли ребро от v_1 към v_2 , от v_2 към v_3 и тъй нататък. Това обхождане също изисква време $\Theta(m+n)$. Ако всички проверки завършат успешно, то $v_1 \longrightarrow v_2 \longrightarrow \ldots \longrightarrow v_n$ е хамилтонов път. Ако някоя от проверките не успее (т.е. липсва ребро от v_k към v_{k+1} за някое k), то следва, че няма хамилтонов път.

Общото време на алгоритъма е линейно: $\Theta(m+n)$. Алгоритъмът е коректен, защото, ако насочен ацикличен граф съдържа хамилтонов път, то редът на върховете в хамилтоновия път е единствената възможна топологична сортировка.

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 $\big($т.е. от C_{k-1} до $C_k\big)$ 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) Когато всички числа са четни, задачата за разпознаване на повторения все още изисква време $\Omega(n \log n)$. Това се доказва чрез следната редукция: умножаваме дадените числа по 2, така общият случай (произволни цели числа) се свежда до частния случай (четни числа).

```
UniqueGeneral (A[1...n])
    for k ← 1 to n
        A[k] ← 2 × A[k]
    return UniqueEven(A)
```

Коректността на редукцията следва от факта, че две числа са равни тогава и само тогава, когато са равни удвоените им стойности.

Цикълът изразходва време от порядък $n \prec n \log n$, от което следва, че редукцията е достатъчно бърза за целите на доказателството.

б) Когато числата са в интервала от $2n$ до $5n$, задачата за разпознаване на повторения има по-малка времева сложност: $\mathrm{O}(n)$. Това може да се докаже чрез построяване на алгоритъм с линейна сложност. Понеже дължината $3n$ на интервала е от порядък n, можем да използваме идеята на сортирането чрез броене (CountingSort).

```
Unique2n5n (A[1...n])
    C[0...3n]: array of boolean
    for i ← 0 to 3n
        C[i] ← false   // числото i + 2n не е било срещнато все още
    for k ← 1 to n
        i ← A[k] − 2n
        if C[i] = true   // числото i + 2n се среща за втори път
            return false   // има повторения
        C[i] ← true   // числото i + 2n се среща за първи път
    return true   // няма повторения
```

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.