ВТОРО МАЛКО КОНТРОЛНО ПО "ДИЗАЙН И АНАЛИЗ НА АЛГОРИТМИ"
ЗА СТУДЕНТИТЕ ОТ СПЕЦИАЛНОСТ "ИНФОРМАТИКА"
(СУ, ФМИ, 2016 / 2017 УЧ. Г., ЛЕТЕН СЕМЕСТЪР)

Зад. 1. С т.нар. лихвоточки може да се получат компенсации при закупуване на жилище. Но ако не отговаряме на разни изисквания на закона, не можем да получим компенсации. Можем да прехвърлим лихвоточките си само на близък роднина. Ако и той не отговаря на изискванията, може на свой ред да прехвърли лихвоточките на свой близък роднина и т.н. Целта е с няколко сделки лихвоточките да отидат у лице с право на компенсация. Всяка сделка има разходи (напр. нотариални такси). Предложете най-бърз алгоритъм за намиране на редица с най-малък общ разход, ако разходите за всяка сделка са:
а) едни и същи; **(10 точки)** б) различни. **(10 точки)**

Моделирайте задачата с граф и съставете най-бърз алгоритъм. Оценете сложността му. Само най-бърз алгоритъм носи точки!

Решение: Информацията от условието може да се представи чрез граф, чиито върхове са хората (лицата, които могат да участват в сделките), а ребрата свързват близките роднини. Някои от върховете са маркирани: те съответстват на лицата с право на компенсация. Теглото на всяко ребро представлява разходите за съответната сделка. Тълкуваме теглата като дължини и търсим най-къс път от един определен връх (съответстващ на нас като лице в сделките) до кой да е от маркираните върхове. За целта е нужен алгоритъм, който да строи дърво на най-късите пътища от даден връх до всички други върхове. Можем да го прекратим предсрочно в момента, в който алгоритъмът намери най-къс път до някой маркиран връх.

а) Ако разходите за всяка сделка са едни и същи, то теглата на ребрата са равни, тоест търсим най-къс път по брой ребра. Тогава най-бърз алгоритъм е ***търсенето в ширина***. Той решава задачата за време $\Theta(m + n)$, където n е броят на върховете, m е броят на ребрата на графа.

б) Ако сделките имат различни разходи, то търсенето в ширина е некоректен алгоритъм. Разходите са неотрицателни числа, затова можем да ползваме ***алгоритъма на Дейкстра***. Той решава задачата за време $\Theta(m + n \log n)$, ако извършва релаксацията с помощта на приоритетна опашка, реализирана чрез пирамида на Фибоначи.

Зад. 2. Ако някоя програма P поиска достъп до ресурс (например файл или принтер), зает от друга програма Q, то операционната система спира P временно, докато Q освободи ресурса; след това изпълнението на P се възобновява. Предложете алгоритъм, който за линейно време открива дали има "мъртва хватка" — две или повече програми се изчакват взаимно (например P изчаква Q, Q изчаква R, R изчаква P). **(10 точки)**

Решение: Представяме информацията чрез ориентиран граф: върхове са програмите; ребрата съответстват на изчакванията, т.е. има ребро от P към Q тогава и само тогава, когато програмата P е спряна в изчакване на програмата Q. Всеки ориентиран цикъл съответства на "мъртва хватка" и обратно. Най-бърз алгоритъм за намиране на цикъл е ***търсенето в дълбочина***. Този алгоритъм изисква линейно време $\Theta(m + n)$.

Зад. 3. Колко са n-цифрените числа, съставени само от цифрите 1, 2 и 3 по следните правила?

— След цифрата 1, ако не е последна, стои задължително цифрата 2.

— След цифрата 2, ако не е последна, може да стои коя да е от цифрите 1 и 3.

— След цифрата 3, ако не е последна, може да стои коя да е от цифрите 2 и 3.

Трите правила образуват една задача, т.е. трябва да бъдат изпълнени и трите.

Предложете итеративен алгоритъм. Опишете го на псевдокод като функция
`cnt(n: positive integer): positive integer`
с време $O(n)$ и количество допълнителна памет $O(n)$. **(10 точки)**

Демонстрирайте алгоритъма при $n = 6$. **(10 точки)**

Оптимизирайте количеството допълнителна памет
до константен брой променливи от целочислен тип.
Опишете оптимизирания алгоритъм на псевдокод. **(10 точки)**

Решение: Попълваме двумерна динамична таблица: числото `dyn[i][j]` е броят на j-цифрените числа с последна цифра i, съставени по правилата от условието на задачата.

```
cnt(n: positive integer): positive integer
dyn[1...3][1...n]: array of integer
for i ← 1 to 3 do
  dyn[i][1] ← 1  // има три едноцифрени числа — 1, 2, 3
for j ← 2 to n do
  dyn[1][j] ← dyn[2][j-1]
  dyn[2][j] ← dyn[1][j-1] + dyn[3][j-1]
  dyn[3][j] ← dyn[2][j-1] + dyn[3][j-1]
return dyn[1][n] + dyn[2][n] + dyn[3][n]
```

Демонстрация на алгоритъма при $n = 6$:

dyn	$j = 1$	$j = 2$	$j = 3$	$j = 4$	$j = 5$	$j = 6$
$i = 1$	1	1	2	3	6	10
$i = 2$	1	2	3	6	10	19
$i = 3$	1	2	4	7	13	23

Има общо $10 + 19 + 23 = 52$ шестцифрени числа, съставени по тези правила.

Можем да намалим количеството допълнителна памет, като пазим само два последователни стълба на таблицата**:**

```
cnt(n: positive integer): positive integer
old1, old2, old3, new1, new2, new3: integer
old1 ← 1
old2 ← 1
old3 ← 1
for j ← 2 to n do
  new1 ← old2
  new2 ← old1 + old3
  new3 ← old2 + old3
  old1 ← new1
  old2 ← new2
  old3 ← new3
return new1 + new2 + new3
```

Задачата може да се реши и по друг начин: с помощта на подходящ граф. Върховете на графа са различните цифри, от които е образувана редицата, тоест цифрите 1, 2 и 3. Ако цифрата j може да стои след цифрата i, то графът съдържа ребро от върха i към върха j.

$$1 \rightleftarrows 2 \rightleftarrows 3 \circlearrowleft$$

Всяко n-цифрено число, образувано по правилата от условието, съответства на маршрут с n върха, т.е. $n-1$ ребра (маршрутът може да повтаря върхове). Значи, търсим броя на маршрутите с дължина $n-1$. Техният брой е равен на сбора от елементите на матрицата A^{n-1}, където $A = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}$ е матрицата на съседствата на графа. Всяко умножение на две матрици с три реда и стълба изисква константно време, защото размерите на матрицата не зависят от n. (Предполагаме, че всеки елемент на матрицата се побира в една променлива от целочислен тип, което обаче е вярно само за малки степенни показатели.) Извод: времето за степенуване е равно по порядък на броя на умноженията на матрици. Ако степенуваме по наивния алгоритъм, тоест ако изчисляваме всички степени на матрицата: $A^k = A \,.\, A^{k-1}$ за $k = 2, 3, \ldots, n-1$, тогава времето на целия алгоритъм е $\Theta(n)$, като алгоритъмът използва 27 променливи от целочислен тип — елементите на трите матрици. Това решение е вярно и носи 30 точки (от тях 10 точки са за демонстрацията: вж. по-долу).

Ако ползваме бързо степенуване, т.е. последователно повдигане на квадрат, то времето е още по-малко: $\Theta(\log\ n)$. Това решение се оценява с 40 точки (от тях 10 точки са за демонстрацията, други 10 точки са бонус за постигнатото бързодействие).

Демонстрация на алгоритъма (с бързо степенуване) при $n = 6$**:**

$$A = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}, \quad A^2 = A\,.\,A = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 2 & 1 \\ 1 & 1 & 2 \end{pmatrix}, \quad A^4 = A^2.\,A^2 = \begin{pmatrix} 2 & 1 & 3 \\ 1 & 5 & 4 \\ 3 & 4 & 6 \end{pmatrix},$$

$A^5 = A^4.\,A = \begin{pmatrix} 1 & 5 & 4 \\ 5 & 5 & 9 \\ 4 & 9 & 10 \end{pmatrix}$**.** Сборът на елементите на матрицата A^5 е равен на

$1 + 5 + 4 + 5 + 5 + 9 + 4 + 9 + 10 = 52$, което е отговорът на задачата, т.е. броят на всички шестцифрени числа, съставени по правилата от условието. Всяко отделно събираемо е броят на шестцифрените числа от един или друг вид. Например събираемото 4 в първия ред и третия стълб означава, че има четири шестцифрени числа, започващи с цифрата 1 и завършващи с цифрата 3. Това са числата 123333, 123323, 123233 и 121233.

В тази задача графът и матрицата са симетрични, което позволява да се намали използваното количество памет приблизително наполовина (за целта пазим половината матрица — под или над главния диагонал). Тази оптимизация не носи точки, тъй като не променя порядъка на сложността по памет.

В общия случай графът и матрицата са несиметрични, което не е пречка: алгоритъмът остава в сила.

По тези два начина — чрез динамично програмиране и чрез степенуване на матрицата на съседствата на подходящ граф — могат да се решават и други подобни задачи, в които се търси броят на редиците от определен вид. Важно е изискванията към редиците да са локални, т.е. да засягат само съчетаването на съседни елементи. Не е съществен видът на елементите — цифри, букви, цветове, действия и др.