Д И Н А М И Ч Н О П Р О Г Р А М И Р А Н Е

ПРИМЕРНО КОНТРОЛНО № 4 ПО "ДИЗАЙН И АНАЛИЗ НА АЛГОРИТМИ" — СУ, ФМИ (ЗА СПЕЦИАЛНОСТ "КОМПЮТЪРНИ НАУКИ", 2. ПОТОК, ФЕВРУАРИ – ЮНИ 2019 Г.)

Зад. 1. Да се намери броят на наредените n-орки $(x_1, x_2, \ldots, x_n)$, състоящи се от цели неотрицателни числа, които са решения на уравнението

$$A_1 x_1 + A_2 x_2 + \ldots + A_n x_n = B,$$

по дадени цели положителни числа A_1, A_2, ..., A_n и B.

Предложете итеративен алгоритъм. Опишете го на псевдокод като функция

`numSolEq(A[1...n]: array of int, B: int): int`

с време $O(nB)$ и динамична таблица с $O(nB)$ клетки. **(12 точки)**

Демонстрирайте алгоритъма при $A = (2; 3; 5)$ и $B = 9$. **(12 точки)**

Оптимизирайте сложността по памет до динамична таблица с $O(B)$ клетки. Опишете оптимизирания алгоритъм на псевдокод. **(16 точки)**

Решение:

```
numSolEq(A[1...n]: array of int, B: int): int
dyn[1...n][0...B]: array of int
for ñ ← 0 to n do
  dyn[ñ][0] ← 1
for B̃ ← 1 to B do
  dyn[0][B̃] ← 0
for ñ ← 1 to n do
  for B̃ ← 1 to B do
    if A[ñ] > B̃
      dyn[ñ][B̃] ← dyn[ñ-1][B̃]
    else
      dyn[ñ][B̃] ← dyn[ñ-1][B̃] + dyn[ñ][B̃-A[ñ]]
return dyn[n][B]
```

Предпоследният ред се основава на правилото за събиране: всички решения на уравнението $A_1 x_1 + A_2 x_2 + \ldots + A_n x_n = B$ в цели неотрицателни числа са два вида — такива, в които $x_n = 0$ (броят им се дава от първото събираемо), и такива, в които $x_n > 0$ (второто събираемо, т.е. броят на решенията на уравнението $A_1 x_1 + A_2 x_2 + \ldots + A_n y = B - A_n$ в цели неотрицателни числа, където е положено $y = x_n - 1$).

Демонстрация на алгоритъма при $A = (2; 3; 5)$ и $B = 9$**:**

dyn	$\widetilde{B}=0$	$\widetilde{B}=1$	$\widetilde{B}=2$	$\widetilde{B}=3$	$\widetilde{B}=4$	$\widetilde{B}=5$	$\widetilde{B}=6$	$\widetilde{B}=7$	$\widetilde{B}=8$	$\widetilde{B}=9$
$\widetilde{n}=0$	1	0	0	0	0	0	0	0	0	0
$\widetilde{n}=1$	1	0	1	0	1	0	1	0	1	0
$\widetilde{n}=2$	1	0	1	1	1	1	2	1	2	2
$\widetilde{n}=3$	1	0	1	1	1	2	2	2	3	(3)

В клетката dyn[$\widetilde{n}$][$\widetilde{B}$] се пази броят на решенията на уравнението $A_1 x_1 + A_2 x_2 + \ldots + A_{\widetilde{n}} x_{\widetilde{n}} = \widetilde{B}$ в цели неотрицателни числа**.**

От долния десен ъгъл dyn[*n*][*B*], тоест dyn[3][9], се получава отговорът на задачата. Следователно уравнението $2x_1 + 3x_2 + 5x_3 = 9$ има три решения в цели неотрицателни числа.

Оптимизация по памет може да се постигне, като се пази само един ред от динамичната таблица.

```
numSolEq(A[1...n]: array of int, B: int): int
dyn[0...B]: array of int
dyn[ 0 ] ← 1
for B̃ ← 1 to B do
  dyn[ B̃ ] ← 0
for ñ ← 1 to n do
  for B̃ ← A[ ñ ] to B do
    dyn[ B̃ ] ← dyn[ B̃ ] + dyn[ B̃ - A[ ñ ]]
return dyn[ B ]
```

Сложността по време обаче остава $\Theta(nB)$ в най-лошия случай, колкото е при първата версия. Вложеният цикъл прави малка оптимизация на времето: то намалява при големи коефициенти на уравнението, но това е най-добрият, а не най-лошият случай.

Задача 2. Дадени са дължините X [1...n] и ширините Y [1...n] на n пощенски плика.
Ще смятаме, че плик № i може да бъде поставен в плик № j тогава и само тогава, когато
X [i] < X [j] и Y [i] < Y [j] или X [i] < Y [j] и Y [i] < X [j].
Да се състави алгоритъм, който за време $O(n^2)$ избира възможно най-много пликове,
които могат да се вложат последователно, т.е. първият избран плик може да се сложи
във втория избран плик; вторият — в третия; третият — в четвъртия; и тъй нататък.

а) Опишете алгоритъма с думи. **(10 точки)**

б) Анализирайте времевата сложност на алгоритъма. **(10 точки)**

Решение: За време $\Theta(n^2)$ проверяваме всеки два плика: дали единият може да се сложи в другия. Същевременно построяваме граф с n върха и m ребра: върхове ма графа са пощенските пликове, а ребра са откритите възможности за влагане. По-точно, добавяме ребро от връх № i към връх № j тогава и само тогава, когато плик № i може да бъде сложен в плик № j. Със сигурност се получава ***ориентиран ацикличен граф,*** защото всеки път в графа съответства на строго растяща редица от лица на пощенски пликове, а никоя строго растяща редица не може да съдържа повторения. Върху този граф извършваме ***топологично сортиране*** по схемата ***обхождане в дълбочина.*** Най-дълга редица от вложени пощенски пликове съответства на най-дълъг път в получения граф. Търсим ***най-дълъг път*** в ориентирания ацикличен граф с помощта на ***динамично програмиране.*** Топологичното сортиране и динамичното програмиране изразходват време $\Theta(m+n) = O(n^2)$. Така общото време на алгоритъма е $\Theta(n^2)$.

Задача 3. Предложете алгоритъм с времева сложност $o(n^2)$, който намира по колко начина положителното цяло число n се представя като сбор от кубове на цели положителни числа (между които може да има еднакви). Сборове, които се различават единствено по реда на събираемите, трябва да се броят като различни представяния на числото n.

а) Опишете алгоритъма на псевдокод. **(15 точки)**

б) Направете подробна демонстрация на алгоритъма за $n = 31$. **(15 точки)**

в) Анализирайте времевата сложност на алгоритъма. **(10 точки)**

Решение:

```
Problem3(n: positive integer)
dyn[0...n]: array of positive integers
// dyn[k] = броя на начините, по които числото k може да се представи
// като сбор от кубове на цели положителни числа
dyn[0] ← 1 // числото 0 има единствено представяне — празния сбор
for k ← 1 to n do
   s ← 0
   p ← 1
   r ← p × p × p
   while r ≤ k do  // всички възможни случаи за последното събираемо (r)
      s ← s + dyn[k – r]
      p ← p + 1
      r ← p × p × p
   dyn[k] ← s
return dyn[n]
```

Демонстрация на алгоритъма за $n = 31$:

k	dyn[k]	k	dyn[k]	k	dyn[k]	k	dyn[k]
0	1	8	2	16	11	24	64
1	1	9	3	17	14	25	78
2	1	10	4	18	18	26	96
3	1	11	5	19	23	27	120
4	1	12	6	20	29	28	150
5	1	13	7	21	36	29	187
6	1	14	8	22	44	30	232
7	1	15	9	23	53	31	286

Отговора прочитаме от последната клетка на таблицата: числото 31 се представя по 286 начина като сбор от кубове на цели положителни числа.

Анализ на времевата сложност на алгоритъма:
Тялото на вътрешния цикъл (while) се изпълнява, докато е в сила неравенството $r = p^3 \le k$, което е равносилно на $p \le \sqrt[3]{k}$, тоест p приема следните стойности: 1, 2, 3, . . . , $\left\lfloor \sqrt[3]{k} \right\rfloor$. Следователно тялото на вътрешния цикъл се изпълнява $\left\lfloor \sqrt[3]{k} \right\rfloor$ пъти за всяко цяло k от 1 до n вкл.; границите на k определяме от външния цикъл (for). Следователно тялото на вътрешния цикъл се изпълнява общо (тоест за целия алгоритъм) $\sqrt[3]{1} + \sqrt[3]{2} + \sqrt[3]{3} + \cdots + \sqrt[3]{n} = \Theta\left(n . \sqrt[3]{n}\right)$ пъти. Това е времевата сложност на алгоритъма: $\Theta\left(n . \sqrt[3]{n}\right)$.