

ДИНАМИЧНО ПРОГРАМИРАНЕ

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

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

по дадени цели положителни числа A_1, A_2, \dots, 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.

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

Оптимизирайте сложността по памет до динамична таблица с $O(B)$ клетки.

Опишете оптимизирания алгоритъм на псевдокод.

Решение:

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

`dyn[1...n][0...B]: array of int`

for $\tilde{n} \leftarrow 0$ **to** n **do**

`dyn[\tilde{n}][0] \leftarrow 1`

for $\tilde{B} \leftarrow 1$ **to** B **do**

`dyn[0][\tilde{B}] \leftarrow 0`

for $\tilde{n} \leftarrow 1$ **to** n **do**

for $\tilde{B} \leftarrow 1$ **to** B **do**

if $A[\tilde{n}] > \tilde{B}$

`dyn[\tilde{n}][\tilde{B}] \leftarrow dyn[$\tilde{n}-1$][\tilde{B}]`

else

`dyn[\tilde{n}][\tilde{B}] \leftarrow dyn[$\tilde{n}-1$][\tilde{B}] + dyn[\tilde{n}][$\tilde{B}-A[\tilde{n}]$]`

return `dyn[n][B]`

Предпоследният ред се основава на правилото за събиране: всички решения на уравнението $A_1 x_1 + A_2 x_2 + \dots + A_n x_n = B$ 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 $A_1 x_1 + A_2 x_2 + \dots + A_n y = B - A_n$ в цели неотрицателни числа, където е положено $y = x_n - 1$).

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

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

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

От долния десен ъгъл $\text{dyn}[n][B]$, тоест $\text{dyn}[3][9]$, 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  $\tilde{B} \leftarrow 1$  to B do
    dyn[ $\tilde{B}$ ] ← 0
for  $\tilde{n} \leftarrow 1$  to n do
    for  $\tilde{B} \leftarrow A[\tilde{n}]$  to B do
        dyn[ $\tilde{B}$ ] ← dyn[ $\tilde{B}$ ] + dyn[ $\tilde{B} - A[\tilde{n}]$ ]
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.

Зад. 2. Дадени са три масива $A[1\dots n]$, $B[1\dots n]$ и $C[1\dots n]$. Намерете най-голямата стойност на сбор от n събираеми, който може да се образува, като се спазват следните правила:

- 1) k -тото събираемо е $A[k]$, $B[k]$ или $C[k]$;
- 2) не може да има две поредни събираеми от един и същи масив (ако например k -тото събираемо е $A[k]$, то $(k+1)$ -ото събираемо е или $B[k+1]$, или $C[k+1]$, но не и $A[k+1]$).

Предложете итеративен алгоритъм. Опишете го на псевдокод като функция `maxSum(A[1...n], B[1...n], C[1...n]: arrays of int): int` с време $O(n)$ и динамична таблица с $O(n)$ клетки.

Демонстрирайте алгоритъма при

$$\begin{aligned} A &= (2 ; 3 ; 5 ; 1 ; 8 ; 7 ; 6) ; \\ B &= (6 ; 9 ; 8 ; 0 ; 5 ; 9 ; 7) ; \\ C &= (4 ; 1 ; 2 ; 8 ; 4 ; 2 ; 5) . \end{aligned}$$

Оптимизирайте паметта до константен брой променливи от целочислен тип. Опишете оптимизирания алгоритъм на псевдокод.

Решение:

```
maxSum(A[1...n], B[1...n], C[1...n]: arrays of int): int
dyn['A'...'C'][1...n]: array of int
dyn['A'][1] ← A[1]
dyn['B'][1] ← B[1]
dyn['C'][1] ← C[1]
for  $\tilde{n} \leftarrow 2$  to  $n$  do
    dyn['A'][$ $\tilde{n}$ ] ←  $A[\tilde{n}] + \max(\text{dyn}['B'][\tilde{n}-1], \text{dyn}['C'][\tilde{n}-1])$ 
    dyn['B'][$ $\tilde{n}$ ] ←  $B[\tilde{n}] + \max(\text{dyn}['C'][\tilde{n}-1], \text{dyn}['A'][\tilde{n}-1])$ 
    dyn['C'][$ $\tilde{n}$ ] ←  $C[\tilde{n}] + \max(\text{dyn}['A'][\tilde{n}-1], \text{dyn}['B'][\tilde{n}-1])$ 
return  $\max(\text{dyn}['A'][n], \text{dyn}['B'][n], \text{dyn}['C'][n])$ 
```

В клетката `dyn[X][$ \tilde{n}]` стои най-големият възможен сбор от първите \tilde{n} елемента, ако последното събираемо е от масива с име X .

Демонстрация на алгоритъма при

$$A = (2 ; 3 ; 5 ; 1 ; 8 ; 7 ; 6) ;$$

$$B = (6 ; 9 ; 8 ; 0 ; 5 ; 9 ; 7) ;$$

$$C = (4 ; 1 ; 2 ; 8 ; 4 ; 2 ; 5) :$$

dyn	$\tilde{n} = 1$	$\tilde{n} = 2$	$\tilde{n} = 3$	$\tilde{n} = 4$	$\tilde{n} = 5$	$\tilde{n} = 6$	$\tilde{n} = 7$
'A'	2	9	18	18	34	38	49
'B'	6	13	17	18	31	43	45
'C'	4	7	15	26	22	36	48

Отговорът на задачата е най-голямото число в последния стълб, тоест 49. Това е най-големият сбор, който можем да получим при спазване на правилата от условието. За да намерим кои събираеми образуват този сбор, трябва още при построяване на динамичната таблица да пазим във всяка клетка указател към по-голямата от двете сравнявани клетки от предишния стълб. Тръгвайки от най-голямата клетка в последния стълб и движейки се по тези указатели, получаваме път (оцветените в жълто клетки), който ни дава информация за събираемите на максималния сбор:

$$\begin{aligned} C[1] + B[2] + A[3] + C[4] + A[5] + B[6] + A[7] &= \\ = 4 + 9 + 5 + 8 + 8 + 9 + 6 &= 49. \end{aligned}$$

Оптимизация по памет (но не и по време) можем да постигнем, като пазим само един стълб от динамичната таблица (но така губим пътя и събираемите).

```

maxSum (A[1...n], B[1...n], C[1...n]: arrays of int): int
oldMaxA ← A[1]
oldMaxB ← B[1]
oldMaxC ← C[1]
for  $\tilde{n} \leftarrow 2$  to  $n$  do
    newMaxA ← A[ $\tilde{n}$ ] + max(oldMaxB, oldMaxC)
    newMaxB ← B[ $\tilde{n}$ ] + max(oldMaxC, oldMaxA)
    newMaxC ← C[ $\tilde{n}$ ] + max(oldMaxA, oldMaxB)
    oldMaxA ← newMaxA
    oldMaxB ← newMaxB
    oldMaxC ← newMaxC
return max(newMaxA, newMaxB, newMaxC)

```

Зад. 3. В множеството $\{a, b, c\}$ е въведено неасоциативно и некомутативно умножение:

		втори множител		
		a	b	c
първи множител	a	b	c	b
	b	b	a	a
	c	c	c	a

Даден е низ $S[1 \dots n]$, съставен от буквите a, b и c . Ако S се тълкува като произведение, можем ли да сложим скоби тъй, че S да получи стойност a ? Например при $S = abcab$ това е възможно: $(ab)((ca)b) = c(cb) = cc = a$. Но при $S = abb$ е невъзможно: $(ab)b = cb = c$; $a(bb) = aa = b$.

Предложете итеративен алгоритъм. Опишете го на псевдокод като функция $isA(S[1 \dots n] : \text{array of char}) : \text{bool}$

с време $O(n^3)$, без да извиквате наготово алгоритъма СҮК.

Демонстрирайте своя алгоритъм при $S = bcaab$.

Решете задачата и като ползвате наготово алгоритъма СҮК. За целта съставете безконтекстна граматика в нормална форма на Чомски.

Решение: Безконтекстна граматика (в нормална форма на Чомски), пораждаща изразите, които могат да имат стойност a при подходящо поставяне на скоби:

$$\begin{aligned}
 A &\rightarrow a \\
 B &\rightarrow b \\
 C &\rightarrow c \\
 A &\rightarrow BB \\
 A &\rightarrow BC \\
 A &\rightarrow CC \\
 B &\rightarrow AA \\
 B &\rightarrow AC \\
 B &\rightarrow BA \\
 C &\rightarrow AB \\
 C &\rightarrow CA \\
 C &\rightarrow CB
 \end{aligned}$$

Стартовият символ е A . Първите три правила дефинират значението на нетерминалите. Останалите девет правила кодират таблицата за умножение.

Ако не викаме наготово алгоритъма СУК, можем да използваме идеята му.

```

isA(S[1...n]: array of char) : bool
dyn[1...n][1...n]['a','b','c']: array of bool
for i ← 1 to n do
    for j ← i to n do
        dyn[i][j]['a'] ← false
        dyn[i][j]['b'] ← false
        dyn[i][j]['c'] ← false
for i ← 1 to n do
    dyn[i][i][S[i]] ← true
for L ← 2 to n do
    for i ← 1 to n + 1 - L do
        j ← i + L - 1
        // разцепваме S[i...j] на всички възможни места:
        // S[i...k] и S[k+1...j]
        for k ← i to j - 1 do
            // таблицата за умножение
            for p ← 'a' to 'c' do
                for q ← 'a' to 'c' do
                    if dyn[i][k][p] and dyn[k+1][j][q]
                        dyn[i][j][p × q] ← true
return dyn[1][n]['a']

```

Идеята е, че $\text{dyn}[i][j][p] = \text{true} \Leftrightarrow$ произведението $S[i \dots j]$ може да приеме стойност p , където p е някой от елементите a , b или c .

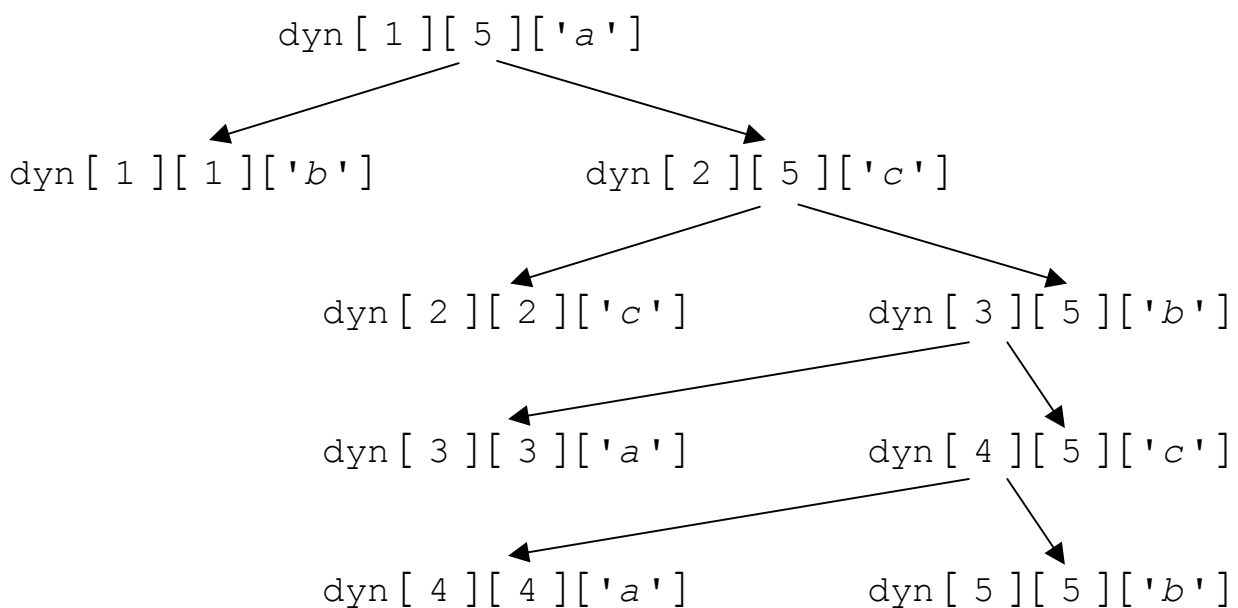
Демонстрация на алгоритъма при $S = bcaab$:

dyn	$j = 1$	$j = 2$	$j = 3$	$j = 4$	$j = 5$
$i = 1$	b	a	b, a	a, c, b	a, b, c
$i = 2$		c	c	c	c, a
$i = 3$			a	b	b, a
$i = 4$				a	c
$i = 5$					b

За удобство третото измерение е проектирано върху равнината на първите две. Тъй като в горния десен ъгъл присъства стойността a , то възможно е да поставим скоби, така че стойността на израза S да бъде равна на a .

Ако искаме да намерим самото разположение на скобите, трябва във всяка клетка със стойност `true` да пазим указател към онези клетки, от които е получена тази стойност. С други думи, на предпоследния ред от псевдокода трябва в клетката $\text{dyn}[i][j][p \times q]$ да запазим два указателя към клетките $\text{dyn}[i][k][p]$ и $\text{dyn}[k+1][j][q]$.

За да не претрупваме демонстрацията, ще покажем само дървото на израза, което се получава от клетката $\text{dyn}[1][n]['a']$.



Това дърво съответства на следното разположение на скоби: $S = b(c(a(ab)))$.

Действително, $S = b(c(a(ab))) = b(c(ac)) = b(cb) = bc = a$.

Зад. 4. 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.

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

Демонстрирайте алгоритъма при $n = 6$.

Оптимизирайте количеството допълнителна памет до константен брой променливи от целочислен тип. Опишете оптимизирания алгоритъм на псевдокод.

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
```