

ДИНАМИЧНО ПРОГРАМИРАНЕ
КОНТРОЛНО № 4 ПО “ДИЗАЙН И АНАЛИЗ НА АЛГОРИТМИ” —
ЗА СТУДЕНТИТЕ ОТ СПЕЦИАЛНОСТ “КОМПЮТЪРНИ НАУКИ”, 1. ПОТОК,
СУ, ФМИ, ЛЕТЕН СЕМЕСТЪР НА 2018 / 2019 УЧЕБНА ГОДИНА

ВАРИАНТ № 1

Задача 1. Всяка пермутация без повторение представлява композиция от независими цикли. Например пермутацията 5, 6, 3, 1, 4, 2 се състои от три цикъла: (5, 4, 1), (6, 2) и (3). Всяка неподвижна точка (да кажем, числото 3 от дадения пример) сама по себе си е цикъл. Нека $c(n; k)$ е броят на пермутациите без повторение на числата 1, 2, 3, ..., n , които съдържат точно k цикъла. Числата n и k са цели неотрицателни и $k \leq n$.

а) Опишете на псевдокод итеративен алгоритъм за пресмятането на функцията $c(n; k)$ със сложност по време и памет $O(nk)$.

Упътване: Разгледайте две възможности за числото n : да бъде неподвижна точка или да участва в цикъл с дължина поне 2. Във втория случай помислете колко възможности има за числото, след което се вмъква n в съответния цикъл.

б) Попълнете таблица със стойностите на $c(n; k)$ за всички n и k , ненадхвърлящи 5.

в) Оптимизирайте алгоритъма така, че сложността по памет да стане $O(k)$. Опишете оптимизацията на псевдокод. (Ако това е направено в точка “а”, само се позовете на нея.)

Задача 2. Съставете алгоритъм с времева сложност $O(n^2)$, който по дадена редица $A[1..n]$ от цели положителни числа търси най-дълга подредица, всеки член на която без последния дели следващия член на подредицата.

Задачата може да се реши поне по два начина: чрез ориентиран ацикличен граф или направо (тоест без построяване на граф). Ако решавате задачата по първия начин, опишете алгоритъма словесно: как се построява графът, какво представляват върховете и ребрата му, как се определя посоката на всяко ребро, защо графът е ацикличен, до коя известна задача за динамично програмиране при ориентирани ациклични графи се свежда задача 2.

Ако решавате задачата без граф, опишете алгоритъма на псевдокод. В този случай е достатъчно да намерите само дължината на най-дълга подредица без самата подредица. Получавате допълнителни точки, ако допишете псевдокода така, че да отпечата и самата най-дълга подредица.

Както и да решавате задачата, демонстрирайте работата на алгоритъма върху следните входни данни: $A = (7; 5; 6; 30; 14; 210; 150; 250)$.

СХЕМА НА ТОЧКУВАНЕ

Цялото контролно носи максимум 20 точки, разпределени по задачи, както следва.

Задача 1 съдържа 12 точки — по 4 точки за всяко подусловие.

Задача 2 съдържа 8 точки — по 4 точки за всяка стъпка:

— описание на алгоритъма;

— демонстрация на алгоритъма.

Допълнителни 4 точки (извън предвидения максимум от 20 т.) носи алгоритъм на псевдокод (неизползващ графи), който в задача 2 намира самата подредица (а не само дължината ѝ).

РЕШЕНИЯ

Задача 1. Пермутациите без повторение на числата $1, 2, \dots, n$, съдържащи точно k цикъла, са два вида. В първия вид числото n е неподвижна точка, тоест само образува цикъл. Затова след премахването му остава пермутация без повторение на числата $1, 2, \dots, n-1$, съдържаща точно $k-1$ цикъла. Броят на тези пермутации е $c(n-1; k-1)$.

Пермутациите от другия вид съдържат числото n в цикъл с дължина поне 2. След изтриването на n остава пермутация без повторение на числата $1, 2, \dots, n-1$, съдържаща точно k цикъла. Тези пермутации са $c(n-1; k)$ на брой, а числото n може да се вмъкне в коя да е от тях, общо на $n-1$ места — зад което и да е от числата $1, 2, \dots, n-1$.

Ето защо функцията $c(n, k)$ удовлетворява рекурентното уравнение

$$c(n; k) = c(n-1; k-1) + (n-1) \cdot c(n-1; k) \text{ при } n > k > 0,$$

както и следните начални условия:

$c(n; 0) = 0$ за всяко цяло $n \geq 1$ (всяка непразна пермутация съдържа цикъл);

$c(n; n) = 1$ за всяко цяло $n \geq 0$ (идентитетът съдържа най-много цикли).

Таблицата по-долу съдържа първите няколко стойности на функцията $c(n; k)$.

$n \backslash k$	0	1	2	3	4	5	6	7	8	9
0	1									
1	0	1								
2	0	1	1							
3	0	2	3	1						
4	0	6	11	6	1					
5	0	24	50	35	10	1				
6	0	120	274	225	85	15	1			
7	0	720	1764	1624	735	175	21	1		
8	0	5040	13068	13132	6769	1960	322	28	1	
9	0	40320	109584	118124	67284	22449	4536	546	36	1

В комбинаториката тези числа са известни като **числа на Стирлинг от първи род**.

Псевдокод на алгоритъма:

$c(n, k) \quad // \quad 0 \leq k \leq n$

`dyn[0...n][0...k]: array of integers`

for $m \leftarrow 1$ **to** n **do**

`dyn[m][0] \leftarrow 0`

for $j \leftarrow 0$ **to** k **do**

`dyn[j][j] \leftarrow 1`

for $m \leftarrow 1$ **to** n **do**

for $j \leftarrow 1$ **to** $\min(m-1, k)$ **do**

`dyn[m][j] \leftarrow dyn[m-1][j-1] + (m-1) \times dyn[m-1][j]`

return `dyn[n][k]`

Изложеното решение има сложност $\Theta(nk)$ — колкото е размерът на динамичната таблица. Може да се постигне сложност по памет $\Theta(k)$ с помощта на следното наблюдение: числата във всеки ред от таблицата зависят само от числата в предходния ред. Затова е достатъчно да пазим само един ред от таблицата (и да го пресмятаме отдясно наляво).

Псевдокод на оптимизирания алгоритъм:

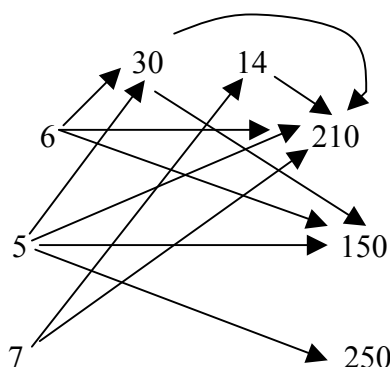
```

c(n, k) // 0 ≤ k ≤ n
if n = k
    return 1
if k = 0
    return 0
dyn[0...k]: array of integers
dyn[0] ← 0
for m ← 1 to n do
    if m ≤ k
        dyn[m] ← 1
    for j ← min(m-1, k) downto 1 do
        dyn[j] ← dyn[j-1] + (m-1) × dyn[j]
return dyn[k]

```

Задача 2. По дадения масив $A[1..n]$ построяваме ориентиран граф с върхове $1, 2, \dots, n$ (индексите на елементите на масива). От върха i към върха j има ребро тогава и само тогава, когато $i < j$ и $A[i]$ дели $A[j]$. Така построеният ориентиран граф е ацикличен, защото ребрата сочат само от връх с по-малък към връх с по-голям номер. Търсената най-дълга подредица съответства на най-дълъг път в графа; за тази задача разполагаме с готов алгоритъм, изучен на лекции — динамично програмиране при ориентирани ациклични графи. Можем да си спестим топологичното сортиране на графа: върховете са сортирани поначало, защото ребрата сочат от връх с по-малък към връх с по-голям номер.

При $A = (7; 5; 6; 30; 14; 210; 150; 250)$ графът изглежда така:



Най-дългите пътища в графа имат дължина 2 (тоест състоят се от две ребра и три върха). Има няколко такива пътя и всеки от тях съответства на най-дълга подредица от (три) числа, всяко от които е делител на следващото. Една такава подредица е $(7; 14; 210)$. Има и други, например $(5; 30; 150)$ и $(6; 30; 150)$.

Разновидност на горното решение е да добавим два фиктивни върха s и t : от s излизат ребра към всички други върхове (вкл. t), а в t влизат ребра от всички други върхове (вкл. s). Сега търсим най-дълъг път от s до t вместо най-дълъг път между всеки два върха.

Построяването на графа изисква време $\Theta(n^2)$: трябва да проверим за всяка двойка числа дали има ребро между тях. Търсенето на най-дълъг път в получения граф изразходва време, линейно спрямо размера на графа, което пак е $\Theta(n^2)$: толкова са ребрата в най-лошия случай (когато всяко число дели всички следващи числа). Времето на целия алгоритъм е сборът от тези две времена, тоест $\Theta(n^2)$.

Можем да използваме същата идея, без да строим явен граф. Псевдокод:

```
LongestDivSubsequence(A[1...n]: array of positive integers)
dyn[1...n]: array of positive integers
prev[1...n]: array of non-negative integers
// dyn[k] = дължината на най-дългата подредица на A[1...k],
// завършваща с A[k], всеки член на която дели следващия;
// prev[k] = индекса от A на нейния предпоследен член.
dyn[1] ← 1
prev[1] ← 0 // Елементът A[1] няма предходен.
for j ← 2 to n do
    dyn[j] ← 0
    prev[j] ← 0 // Елементът A[j] е начало на подредица.
    for i ← 1 to j - 1 do
        if A[j] mod A[i] = 0 // ако A[i] дели A[j]
            if dyn[i] > dyn[j]
                dyn[j] ← dyn[i]
                prev[j] ← i // A[j] продължава някоя подредица.
                // Понеже i < j, то prev[j] < j.
    dyn[j] ← dyn[j] + 1
bestEnd ← 1
for j ← 2 to n do
    if dyn[j] > dyn[bestEnd]
        bestEnd ← j

// Възстановяване на решението:
// индексите на най-дългата подредица се отпечатват
// в обратен ред (от най-големия към най-малкия).
j ← bestEnd
while j > 0 do
    print j
    j ← prev[j] // Понеже prev[j] < j, то j намалява строго.

// Алгоритъмът връща дължината на най-дълга подредица,
// всеки член на която (без последния) дели следващия.
return dyn[bestEnd]
```

Анализ на времевата сложност: Двата вложени цикъла, попълващи динамичната таблица, изразходват време $\Theta(n^2)$. Цикълът след тях, който обхожда попълнената таблица и търси най-голяма дължина, изисква време $\Theta(n)$. Възстановяването изисква време $O(n)$, тъй като индексът j намалява с поне една единица на всяка стъпка. Окончателно, времето за работа на целия алгоритъм е $\Theta(n^2)$.

Демонстрация на алгоритъма при $A = (7; 5; 6; 30; 14; 210; 150; 250)$:

k	1	2	3	4	5	6	7	8
$A[k]$	7	5	6	30	14	210	150	250
$\text{dyn}[k]$	1	1	1	2	2	3	3	2
$\text{prev}[k]$	0	0	0	2	1	4	4	2

Най-голямото число в реда dyn е числото 3. В случая има няколко такива числа. По принцип няма значение кое от тях ще използваме. Алгоритъмът запомня индекса на първото от тях. Първата тройка има индекс $k = 6$, затова алгоритъмът започва възстановяването от шестия елемент на масива A :

$$\text{prev}[6] = 4; \quad \text{prev}[4] = 2; \quad \text{prev}[2] = 0 \quad (\text{няма повече членове}).$$

Намерената най-дълга подредица се състои от втория, четвъртия и шестия член на масива A , тоест това е редицата $(5; 30; 210)$.