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" — ЗА СТУДЕНТИТЕ ОТ СПЕЦИАЛНОСТ "КОМПЮТЪРНИ НАУКИ", 2. ПОТОК, СУ, ФМИ, ЛЕТЕН СЕМЕСТЪР НА 2019 / 2020 УЧЕБНА ГОДИНА

Задача 1. Нека $S(n; k)$ е броят на начините за разпределяне на числата 1, 2, 3, . . . , n в k неразличими непразни множества. Това, че множествата са неразличими, означава, че например следните две разпределения на числата 1, 2, 3, 4, 5 и 6 ги смятаме за еднакви: 1) {1; 2; 3; 4}, {5; 6}; 2) {5; 6}, {1; 2; 3; 4}.
Тъй като става дума за множества, а не за редици, числата от едно множество нямат наредба, тоест {5; 6} и {6; 5} е едно и също множество.

Накратко, $S(n; k)$ е броят на начините, по които можем да разбием n (различими) обекта в множество от k непразни множества. (Обаче самото множество от k непразни множества може да бъде празно; това се случва при $k = 0$.)

Допустими стойности: Числата k и n са цели неотрицателни и $k \le n$.

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

Упътване: Разгледайте две възможности за числото n: да е самò в своето множество или да участва в множеството заедно с поне още едно число. Във втория случай помислете колко възможности има за множеството, в което участва числото n.

б) Попълнете таблица със стойностите на $S(n; k)$ 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 първия) минус предишния член на подредицата дава разлика, която е точен квадрат (вкл. нула).

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 отпечатва и самата най-дълга подредица.

Както и да решавате задачата, демонстрирайте работата на алгоритъма върху следните входни данни: $A = (7; 5; 8; 9; 30; 34; 98; 50)$.

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. Има два вида разбивания на $\{1; 2; \ldots; n\}$ на k непразни множества. В първия вид числото n самò образува множество. Премахваме го и остава разбиване на $\{1; 2; \ldots; n-1\}$ на $k-1$ непразни множества. Броят на тези разбивания е $S(n-1;\ k-1)$.

Разбиванията от другия вид съдържат числото n в множество с поне още един елемент. След изтриването на n остава разбиване на числата $1, 2, \ldots, n-1$ на k непразни множества. Тези разбивания са $S(n-1; k)$ и можем да сложим n на k места — в кое да е множество.

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

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

както и следните начални условия:

$S(n; 0) = 0$ за всяко цяло $n \geq 1$ (разбиването трябва да е непразно);
$S(n; n) = 1$ за всяко цяло $n \geq 0$ (разбиването $\{\{1\}; \{2\}; \ldots; \{n\}\}$ е единствено).

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

n \ k	0	1	2	3	4	5	6	7	8	9	10
0	1										
1	0	1									
2	0	1	1								
3	0	1	3	1							
4	0	1	7	6	1						
5	0	1	15	25	10	1					
6	0	1	31	90	65	15	1				
7	0	1	63	301	350	140	21	1			
8	0	1	127	966	1701	1050	266	28	1		
9	0	1	255	3025	7770	6951	2646	462	36	1	
10	0	1	511	9330	34105	42525	22827	5880	750	45	1

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

Псевдокод на алгоритъма:

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

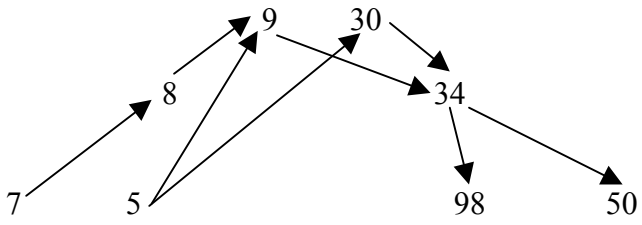
Изложеното решение има сложност $\Theta(nk)$ — колкото е размерът на динамичната таблица. Може да се постигне сложност по памет $\Theta(k)$ с помощта на следното наблюдение: числата във всеки ред от таблицата зависят само от числата в предходния ред. Затова е достатъчно да пазим само един ред от таблицата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

Псевдокод на оптимизирания алгоритъм:

```
S(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] + j × dyn[j]
return dyn[k]
```

Задача 2. По дадения масив $A[1...n]$ построяваме ориентиран граф с върхове 1, 2, . . . , n (индексите на елементите на масива). От върха i към върха j има ребро тогава и само тогава, когато $i < j$ и $A[j] - A[i]$ е точен квадрат. Така построеният ориентиран граф е ацикличен, защото ребрата сочат от връх с по-малък към връх с по-голям индекс. 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 $A = (7;\ 5;\ 8;\ 9;\ 30;\ 34;\ 98;\ 50)$ графът изглежда така:



Най-дългите пътища в графа имат дължина 4 (тоест състоят се от четири ребра и пет върха). Има два такива пътя и всеки от тях съответства на най-дълга подредица от (пет) числа, всяко от които минус предишното дава точен квадрат: (7; 8; 9; 34; 50) и (7; 8; 9; 34; 98).

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 минус всяко предишно дава точен квадрат). 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:

```
LongestSqrSubsequence(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 минус предишния
// дава точен квадрат;
// 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[i] ≤ A[j] and dyn[i] > dyn[j]
         if √(A[j]-A[i]) ∈ ℕ0 // ако A[j]-A[i] е точен квадрат
            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 минус предишния дава точен квадрат.
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 $A = (7;\ 5;\ 8;\ 9;\ 30;\ 34;\ 98;\ 50)$:

k	1	2	3	4	5	6	7	8
$A[k]$	7	5	8	9	30	34	98	50
dyn[k]	1	1	2	3	2	4	5	5
prev[k]	0	0	1	3	2	4	6	6

Най-голямото число в реда dyn е числото 5. В случая има няколко такива числа. По принцип няма значение кое от тях ще използваме. Алгоритъмът запомня индекса на първото от тях. Първата петица има индекс $k = 7$, затова алгоритъмът започва възстановяването от седмия елемент на масива A:

prev[7] = 6; prev[6] = 4; prev[4] = 3; prev[3] = 1; prev[1] = 0 (няма повече членове).

Намерената най-дълга подредица се състои от първия, третия, четвъртия, шестия и седмия елемент на масива A, тоест това е редицата (7; 8; 9; 34; 98).