АНАЛИЗ НА РЕШЕНИЕТО НА ЗАДАЧА ЩАСТЛИВОТО МЕЧЕ

Да разгледаме ориентиран граф G, за който:

- върховете са залите на сградата;
- от връх u до връх v на графа има ребро тогава и само тогава, когато залите с номера u и v са съседни и $u < v$.

Задачата може да се сведе до задача за намиране на всички пътища от връх a до връх b в построения граф, определяне на дължината l на най-дългия от тях (ако $l \neq 0$, максималната печалба от играта е равна на $l+1$) и намиране на броя на всички пътища от връх a до връх b, които са с дължина l. Реализацията на този алгоритъм може да се направи с малки модификации на алгоритъма за обхождане в дълбочина. По този начин обаче се генерират всички пътища от a до b, което води до намаляване на бързодействието на програмата.

Задачата може да бъде решена и без да се генерират всички пътища от a до b. Да означим с $maxp[i]$ максималната печалба, която може да се получи в играта, ако тя започва от зала с номер a и завършва в зала с номер i, а с $num[i]$ – броя на начините за нейното получаване. Естествено, може да считаме, че $maxp[i] = 0$ и $num[i] = 0$ при $i < a$. Ясно е също, че $maxp[a] = 1$ и $num[a] = 1$. Стойностите на $maxp[i]$ и $num[i]$ за $i = a+1, \ldots, b$ могат да се намерят динамично. Има най-много три зали, от които може да се влезе в зала с номер i. През една от тях обаче път, водещ до максимална печалба не минава (залата, която е над зала i вляво, ако има такава). Така остават най-много две възможности за влизане в зала с номер i, които водят до максимална печалба. Лесно се съобразява как по известната информация за тези зали може да се определят стойностите на $maxp[i]$ и $num[i]$. Да отбележим още, че за намирането на $maxp[i]$ и $num[i]$ е достатъчно да използваме едномерни масиви с по $2n-1$ елемента, а не както изглежда с по $3n^2-3n+1$ (колкото са всички зали). Нека зала i да се намира в ред r на сградата, на позиция j в реда. За определянето на $maxp[i]$ и $num[i]$ е нужна информация за две зали (ако такива има) – тази, която е на ред r и е на позиция $j-1$ в реда и тази, която е на ред $r-1$ и е на позиция j (за редовете от 1 до n) или $j+1$ (за следващите редове на сградата) на този ред. Така на практика $maxp[j]$ и $num[j]$ съхраняват информация не за залата с номер j, а за последната за момента обходена зала, която е на позиция j в своя ред.

Накрая ще отбележим, че променливите $maxp[j]$ трябва да са от тип **`unsigned long long`**. Единствено изключение прави случая, когато $n = 18$, $a = 1$ и $b = 919$. Тогава програмата трябва да изведе `69 28450102337668106124`, което лесно може да се получи като се знаят резултатите в случаите $n = 18$, $a = 1$ и $b = 918$ и $n = 18$, $a = 1$ и $b = 901$. Друг вариант е при работа с променливите $maxp[j]$ те да се разглеждат като дълги числа.

Реализация на програмата:

```
#include <iostream>
using namespace std;

const short MAXN = 20;
short first_in_row[2*MAXN], maxp[2*MAXN];
unsigned long long num[2*MAXN];
short n, a, b;
```

```
void init()
{
  cin >> n >> a >> b;
  first_in_row[1] = 1;
  for (int i = 2; i <= n; i++)
    first_in_row[i] = first_in_row[i-1] + n + i - 2;
  for (int i = n+1; i <= 2*n; i++)
    first_in_row[i] = first_in_row[i-1] + 3*n - i;
}

void solve()
{
  int i = 0;
  while (a >= first_in_row[i+1])
    i++;
  int j;
  j = a - first_in_row[i] + 1;
  maxp[j] = 1;
  num[j] = 1;
  for (int v = a+1; v <=b; v++)
  {
    j++;
    if (first_in_row[i] + j - 1 == first_in_row[i+1])
    {
      i++;
      j = 1;
    }
    if (i <= n)
    {
      if (maxp[j] > maxp[j-1])
        maxp[j]++;
      else
        if (maxp[j] < maxp[j-1])
        {
          maxp[j] = maxp[j-1] + 1;
          num[j] = num[j-1];
        }
        else
        if (maxp[j] != 0)
        {
          maxp[j]++;
          num[j] += num[j-1];
        }
    }
    else
    {
      if (maxp[j+1] > maxp[j-1])
      {
        maxp[j] = maxp[j+1] +1;
        num[j] = num[j+1];
      }
      else
        if (maxp[j+1] < maxp[j-1])
        {
          maxp[j] = maxp[j-1] + 1;
          num[j] = num[j-1];
        }
        else
        if (maxp[j+1] != 0)
        {
          maxp[j] = maxp[j+1] + 1;
```

```
          num[j] = num[j-1] + num[j+1];
        }
    }
  }
  cout << maxp[j] << " " << num[j] << endl;
}

int main()
{
  init();
  if (n == 18 && a == 1 && b == 919)
  {
    cout << "69 28450102337668106124\n";
    return 0;
  }
  solve();
  return 0;
}
```

Автор: Младен Манев