搜索-Flood Fill

一、AcWing 1097. 池塘计数

【题目描述】

农夫约翰有一片$N*M$的矩形土地。

最近，由于降雨的原因，部分土地被水淹没了。

现在用一个字符矩阵来表示他的土地。

每个单元格内，如果包含雨水，则用`W`表示，如果不含雨水，则用`.`表示。

现在，约翰想知道他的土地中形成了多少片池塘。

每组相连的积水单元格集合可以看作是一片池塘。

每个单元格视为与其上、下、左、右、左上、右上、左下、右下八个邻近单元格相连。

请你输出共有多少片池塘，即矩阵中共有多少片相连的`W`块。

【输入格式】

第一行包含两个整数N和M。

接下来N行，每行包含M个字符，字符为`W`或`.`，用以表示矩形土地的积水状况，字符之间没有空格。

【输出格式】

输出一个整数，表示池塘数目。

【数据范围】

$1 \leq N, M \leq 1000$

【输入样例】

```
10 12
W........WW.
.WWW.....WWW
....WW...WW.
.........WW.
.........W..
..W......W..
.W.W.....WW.
W.W.W.....W.
.W.W......W.
..W.......W.
```

【输出样例】

```
3
```

【分析】

遍历整个图，如果遇到`W`，那么池塘数量+1，并以该坐标为起点开始搜索连通且为`W`的点，将搜索到的点全部覆盖为`.`。

【代码】

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

const int N = 1010;
char g[N][N];
int n, m, res;

void dfs(int x, int y)
{
    g[x][y] = '.';//将当前的池塘覆盖为陆地
    for (int i = -1; i <= 1; i++)
        for (int j = -1; j <= 1; j++)
        {
            int nx = x + i, ny = y + j;
            if (nx >= 0 && nx < n && ny >= 0 && ny < m && g[nx][ny] ==
'W')
                dfs(nx, ny);

        }
```

```
}

int main()
{
    cin >> n >> m;
    for (int i = 0; i < n; i++) cin >> g[i];
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            if (g[i][j] == 'W') res++, dfs(i, j);//将(i, j)连通的池塘全部覆盖
    cout << res << endl;
    return 0;
}
```

二、AcWing 1098. 城堡问题

【题目描述】

```
   1   2   3   4   5   6   7
  #############################
1 #   |   #   |   #   |   |   #
  #####---#####---#---#####---#
2 #   #   |   #   #   #   #   #
  #---#####---#####---#####---#
3 #   |   |   #   #   #   #   #
  #---#########---#####---#---#
4 #   #   |   |   |   |   #   #
  #############################
          (图 1)

  #  = Wall
  |  = No wall
  -  = No wall
```

方向：上北下南左西右东。

上图是一个城堡的地形图。

请你编写一个程序，计算城堡一共有多少房间，最大的房间有多大。

城堡被分割成$m * n$个方格区域，每个方格区域可以有$0 \sim 4$面墙。

注意：墙体厚度忽略不计。

【输入格式】

第一行包含两个整数m和n，分别表示城堡南北方向的长度和东西方向的长度。

接下来m行，每行包含n个整数，每个整数都表示平面图对应位置的方块的墙的特征。

每个方块中墙的特征由数字P来描述，我们用1表示西墙，2表示北墙，4表示东墙，8表示南墙，P为该方块包含墙的数字之和。

例如，如果一个方块的P为3，则$3 = 1 + 2$，该方块包含西墙和北墙。

城堡的内墙被计算两次，方块$(1, 1)$的南墙同时也是方块$(2, 1)$的北墙。

输入的数据保证城堡至少有两个房间。

【输出格式】

共两行，第一行输出房间总数，第二行输出最大房间的面积（方块数）。

【数据范围】

$1 \leq m, n \leq 50$

$0 \leq P \leq 15$

【输入样例】

```
4 7
11 6 11 6 3 10 6
7 9 6 13 5 15 5
1 10 12 7 13 7 5
13 11 10 8 10 12 13
```

【输出样例】

```
5
9
```

【分析】

本题可用四位的二进制数来表示每个方向的墙的状态，例如3的二进制表示为0011，说明有西墙和北墙。因此将位移方向顺序设置成西、北、东、南，如果该位置的数的第i位不为1说明那个方向没有墙，那么就走过去继续搜索。

【代码】

```
#include <iostream>
```

```cpp
#include <algorithm>
using namespace std;

const int N = 55;
int g[N][N];
int n, m, cnt, res;
bool st[N][N];
int dx[4] = { 0, -1, 0, 1 }, dy[4] = { -1, 0, 1, 0 };

int dfs(int x, int y)
{
    int res = 1;
    st[x][y] = true;
    for (int i = 0; i < 4; i++)
    {
        int nx = x + dx[i], ny = y + dy[i];
        if (nx >= 0 && nx < n && ny >= 0 && ny < m && !st[nx][ny] && !(g[x][y] >> i & 1))
            res += dfs(nx, ny);
    }
    return res;
}

int main()
{
    cin >> n >> m;
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            cin >> g[i][j];
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            if (!st[i][j]) cnt++, res = max(res, dfs(i, j));
    cout << cnt << endl << res << endl;
    return 0;
}
```

三、AcWing 1106. 山峰和山谷

【题目描述】

FGD小朋友特别喜欢爬山，在爬山的时候他就在研究山峰和山谷。

为了能够对旅程有一个安排，他想知道山峰和山谷的数量。

给定一个地图，为FGD想要旅行的区域，地图被分为$n \times n$的网格，每个格子(i,j)的高度$w(i,j)$是给定的。

若两个格子有公共顶点，那么它们就是相邻的格子，如与(i,j)相邻的格子有$(i-1,j-1),(i-1,j),(i-1,j+1),(i,j-1),(i,j+1),(i+1,j-1),(i+1,j),(i+1,j+1)$。

我们定义一个格子的集合S为山峰（山谷）当且仅当：

- S的所有格子都有相同的高度。
- S的所有格子都连通。
- 对于$s \in S$，与s相邻的$s' \notin S$，都有$w_s > w_{s'}$（山峰），或者$w_s < w_{s'}$（山谷）。

如果周围不存在相邻区域，则**同时将其视为山峰和山谷**。

你的任务是，对于给定的地图，求出山峰和山谷的数量，如果所有格子都有相同的高度，那么整个地图即是山峰，又是山谷。

【输入格式】

第一行包含一个正整数n，表示地图的大小。

接下来一个$n \times n$的矩阵，表示地图上每个格子的高度w。

【输出格式】

共一行，包含两个整数，表示山峰和山谷的数量。

【数据范围】

$1 \leq n \leq 1000$

$0 \leq w \leq 10^9$

【输入样例1】

```
5
8 8 8 7 7
7 7 8 8 7
7 7 7 7 7
7 8 8 7 8
7 8 8 8 8
```

【输出样例1】

```
2 1
```

【输入样例2】

```
5
5 7 8 3 1
5 5 7 6 6
6 6 6 2 8
5 7 2 5 8
7 1 0 1 7
```

【输出样例2】

```
3 3
```

【分析】

对于本题，高度相同的相邻区域为一个连通块，然后需要记录两个变量$high, low$分别表示某个连通块的相邻区域是否比自己高或低，在使用$Flood\ Fill$进行搜索的时候，如果搜到了与本连通块高度不一样的点，那么需要比较高度从而标记$high, low$，如果搜索完一个连通块后$high$为$false$，说明附近没有比这个连通块的高度更高的点了，那么山峰数量$peak + 1$；如果low为$false$，说明附近没有比这个连通块的高度更低的点了，那么山谷数量$valley + 1$。

【代码】

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

const int N = 1010;
int h[N][N];
int n, peak, valley;
bool st[N][N], high, low;

void dfs(int x, int y)
{
    st[x][y] = true;
    for (int i = -1; i <= 1; i++)
        for (int j = -1; j <= 1; j++)
        {
            int nx = x + i, ny = y + j;
            if (nx >= 0 && nx < n && ny >= 0 && ny < n)
                if (h[nx][ny] == h[x][y] && !st[nx][ny]) dfs(nx, ny);
                else if (h[nx][ny] > h[x][y]) high = true;
                else if (h[nx][ny] < h[x][y]) low = true;
```

```
        }
}

int main()
{
    cin >> n;
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            cin >> h[i][j];
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            if (!st[i][j])
            {
                high = low = false;//每次搜索前初始化high和low
                dfs(i, j);
                if (!high) peak++;//如果没有比该连通块高的那么该连通块为山峰
                if (!low) valley++;//反之该连通块为山谷
            }
    cout << peak << ' ' << valley << endl;
    return 0;
}
```