开心消消乐

算法 搜索

题目描述

近来，小明的班上风靡着一款名为"开心消消乐"的游戏，为了成为大家眼中的超人，小明开始疯狂研究这款游戏的玩法。

游戏的场景是一个$n \times m$的正方形矩阵，其中有着4种不同形状的方块，玩家要做的就是尽可能多的消除掉其中的方块。

在游戏中，各个方块和上下左右四个方向的相邻位置上形状相同的方块共同构成了一个又一个连通块，玩家选中一个位置后，若该位置所在连通块的方块总数不小于3，那么就可以消除掉这一连通块，得到分数。需要注意的是，游戏存在着重力机制，在机制作用下，每次消除后腾空的方块会下落。

显然，由于重力机制的影响，消除顺序会影响到游戏的最终得分，现在给定游戏矩阵，若简单记每个方块为1分，请试着找出最大的得分。

输入

输入格式：

- 第一行输入两个数n、m，表示矩阵大小。
- 之后n行，每行m个数，表示方块矩阵。每个数可能为1、2、3、4，分别对应一种形状。

样例输入1 复制

```
4 5
1 1 4 2 2
1 3 3 4 2
3 3 3 4 4
1 1 4 2 2
```

样例输出1 复制

```
20
```



```
/*https://blog.csdn.net/newxiaoou/article/details/136049177*/


/*pretest
4 5
1 1 4 2 2
1 3 3 4 2
3 3 3 4 4
1 1 4 2 2
```

```
*/
#include<bits/extc++.h>

using i8 = signed char;
using u8 = unsigned char;
using i16 = signed short int;
using u16 = unsigned short int;
using i32 = signed int;
using u32 = unsigned int;
using f32 = float;
using i64 = signed long long;
using u64 = unsigned long long;
using f64 = double;
using i128 = __int128_t;
using u128 = __uint128_t;
using f128 = long double;
using namespace std;

constexpr i64 mod = 999911659;
constexpr i64 maxn = 2e7 + 5;
constexpr i64 inf = 0x3f3f3f3f3f3f3f3f;

std::vector<std::pair<i64, i64>>dir = {{1, 0}, { -1, 0}, {0, 1}, {0, -1}};


int main() {
    i64 n, m; std::cin >> n >> m;
    std::vector<string>g(n + 2, std::string(m + 2, '0'));
    for (i64 i = 1; i <= n; i++) {
        for (i64 j = 1; j <= m; j++) {
            i64 v; std::cin >> v;
            g[i][j] = v + '0';
        }
        //std::cout  << g[i] << "\n";;
    }
    std::vector<i64>cnt(5, 0);
    std::vector<std::vector<bool>>vis(n + 1, std::vector<bool>(m + 1, false));
    i64 ans = 0; i64 cur = 0;
    std::function<void(i64, i64)>dfs = [&](i64 x, i64 y) {
        vis[x][y] = true;
        cnt[g[x][y] - '0']++;
        for (auto [dx, dy] : dir) {
            i64 xx = x + dx, yy = y + dy;
            if (xx <= 0 or xx > n or yy <= 0 or yy > m)continue;
            if (vis[xx][yy])continue;
            if (g[xx][yy] != (char)(cur + '0'))continue;
            dfs(xx, yy);
        }
    };
    std::function<void(i64, i64)>dfs2 = [&](i64 x, i64 y) {
        vis[x][y] = false;
        for (auto [dx, dy] : dir) {
            i64 xx = x + dx, yy = y + dy;
            if (xx <= 0 or xx > n or yy <= 0 or yy > m)continue;
            if (not vis[xx][yy])continue;
            dfs2(xx, yy);
```

```
        }
    };

    std::map<vector<string>, i64>t;
    std::function<void(i64)>dfs3 = [&](i64 depth) {

        if (t.count(g))return; t[g]++;
        i64 c = 0;
        for (i64 i = 1; i <= n; i++) {
            for (i64 j = 1; j <= m; j++) {
                if (g[i][j] == 'x')c++;
            }
        }
        ans = std::max(ans, c);
        auto tmp = g;
        for (i64 i = 1; i <= n; i++) {
            for (i64 j = 1; j <= m; j++) {
                if (isdigit(g[i][j])) {
                    cur = g[i][j] - '0';
                    dfs(i, j);
                    if (cnt[cur] < 3) {
                        dfs2(i, j);
                        cnt[cur] = 0;
                        // 不满足消除的条件, 还原
                    } else {
                        // 满足消除的条件
                        cnt[cur] = 0;
                        for (i64 i = 1; i <= n; i++) {
                            for (i64 j = 1; j <= m; j++) {
                                if (vis[i][j]) {
                                    g[i][j] = 'x';
                                }
                            }
                        }
                        dfs2(i, j);
                        // 还原vis的状态
                        for (i64 i = 1; i <= m; i++) {
                            i64 w = n;
                            while (isdigit(g[w][i]) and w >= 1)w--;
                            if (not w)continue; i64 p = w;
                            for (i64 j = w; j >= 1; j--) {
                                if (isdigit(g[j][i])) {
                                    swap(g[j][i], g[p][i]);
                                    p--;
                                }
                            }//将上块下移
                        }
                        dfs3(depth + 1);
                        g = tmp;
                        // 回溯后还原
                    }

                }
            }
        }
```

```
    };
    dfs3(0);
    std::cout  << ans << "\n";
}
```