算法与复杂性报告 2：连连看

516030910259 刘欣鹏

1. 问题描述

给定一个连连看棋盘，棋盘上每个点都有各种图案（用非 0 数字表示，该位置上的数字为 0 则说明该位置上没有图案）。输入棋盘上的任意两个坐标，判断这两个坐标对应的图案是否可以消除，消除的条件是图案相同且图案间连线的转角数不得超过 2。

2. 输入数据

in.dat

x y //规模

……//x 行 y 列数据，用非负数表示棋盘各个位置的情况，0 表示没有图案

p q //第一个位置的坐标

m n //第二个位置的坐标

3. 算法思想

算法的基本框架为广度优先搜索算法。对于搜索队列中的每个状态，记录以下变量：qx，qy 表示当前状态下的坐标；face 记录当前状态下连线的朝向，向下为 1，向上为 2，向右为 3，向左为 4；change 记录从 p,q 到当前状态发生转角的次数。

对于当前状态，设其指针为 l，查询与其相邻的四个结点，若该结点被从该方向访问过的状态为未达状态，则将该节点的相应状态加入队列尾端，指针为 r。若 face[r]!=face[l]，则 change[r]=change[l+1]；否则，change[r]=change[l]。若 change[r]>2，则将其从队尾删除。否则， 标记该节点被从该方向访问过的状态为已达状态。

若当前状态坐标与 t1,t2 横纵坐标差的绝对值和为 1，则可能可以从当前状态抵达最终状态。若 change[l]<2，则可抵达最终状态，返回 Yes；若 change[l]=2，则若 face[l]与从当前状态抵达最终状态的方向一致，返回 Yes，否则返回 No。

4. 实现代码

```
#include <iostream>
#include <cmath>
using namespace std;
int main() {
	freopen("data.in", "r", stdin);
	int n, m, i, j, k, l, r, s1, s2, t1, t2, x, y;
	int qx[50001], qy[50001], change[50001], face[50001], a[102][102];
	bool used[4][102][102];
	cin >> n >> m;
	for (i = 1; i <= n; i++)
		for (j = 1; j <= m; j++)
				cin >> a[i][j];
	for (i = 0; i <= n + 1; i++) a[i][0] = a[i][m + 1] = 0;
	for (i = 0; i <= m + 1; i++) a[0][i] = a[n + 1][i] = 0;
	for (i = 0; i <= n + 1; i++)
		for (j = 0; j <= m + 1; j++)
				for (k = 0; k<4; k++)
						used[k][i][j] = false;
	cin >> s1 >> s2 >> t1 >> t2;
	//读入数据及初始化
```

```
if (a[s1][s2] != a[t1][t2]) {
        cout << "No!" << endl;
        return 0;
}
//若两位置图案不一致，则不可消除
r = 0; l = -1;
qx[0] = s1; qy[0] = s2;
used[0][s1][s2] = used[1][s1][s2] = used[2][s1][s2] = used[3][s1][s2] = true;
change[0] = 0;
while (l<r) {
        x = qx[++l]; y = qy[l];
        if (abs(x - t1) + abs(y - t2) <= 1) {
                //若坐标差绝对值之和为 1，则可能抵达最终态
                if (change[l]<2) {
                        cout << "Yes!" << endl;
                        return 0;
                }//若此前转角数小于 2，则必可抵达
                else {
                        if (face[l] == 0 && x + 1 == t1 || face[l] == 1 && x - 1 == t1 || face[l]
== 2 && y + 1 == t2 || face[l] == 3 && y - 1 == t2) {
                                cout << "Yes!" << endl;
                                return 0;
                        }//若此前转角数等于 2，则需要当前连线不增加拐角才能抵达
                }
        }
        if (x + 1 <= n + 1 && a[x + 1][y] == 0 && !used[0][x + 1][y]) {
                //若新状态未被抵达
                if (l == 0 || l != 0 && face[l] != 0 && change[l]<2 || l != 0 && face[l] == 0) {
                        //若抵达新状态后转角数<=2，则可添加此状态到搜索队列
                        r++;
                        qx[r] = x + 1; qy[r] = y;
                        face[r] = 0;
                        if (l == 0) change[r] = 0;
                        //初始特殊情况
                        else if (face[l] != 0) change[r] = change[l] + 1;
                        //若 l 与 r 对应状态朝向不同，则拐角增加
                        else change[r] = change[l];
                        //若 l 与 r 对应状态朝向相同，则拐角数不变
                        used[0][x + 1][y] = true;
                }
        }//其他情况与此相似
        if (x - 1 >= 0 && a[x - 1][y] == 0 && !used[1][x - 1][y]) {
                if (l == 0 || l != 0 && face[l] != 1 && change[l]<2 || l != 0 && face[l] == 1) {
                        r++;
```

```
                    qx[r] = x - 1; qy[r] = y;
                    face[r] = 1;
                    used[1][x - 1][y] = true;
                    if (l == 0) change[r] = 0;
                    else if (face[l] != 1) change[r] = change[l] + 1;
                    else change[r] = change[l];
                }
            }
            if (y + 1 <= m + 1 && a[x][y + 1] == 0 && !used[2][x][y + 1]) {
                if (l == 0 || l != 0 && face[l] != 2 && change[l]<2 || l != 0 && face[l] == 2) {
                    r++;
                    qx[r] = x; qy[r] = y + 1;
                    face[r] = 2;
                    used[2][x][y + 1] = true;
                    if (l == 0) change[r] = 0;
                    else if (face[l] != 2) change[r] = change[l] + 1;
                    else change[r] = change[l];
                }
            }
            if (y - 1 >= 0 && a[x][y - 1] == 0 && !used[3][x][y - 1]) {
                if (l == 0 || l != 0 && face[l] != 1 && change[l]<2 || l != 0 && face[l] == 1) {
                    r++;
                    qx[r] = x; qy[r] = y - 1;
                    face[r] = 3;
                    used[3][x][y - 1] = true;
                    if (l == 0) change[r] = 0;
                    else if (face[l] != 3) change[r] = change[l] + 1;
                    else change[r] = change[l];
                    }
            }
        }
        //广度优先搜索
        cout << "No!" << endl;
        //若未搜索到终态，则不可消除
        return 0;
}
```

5. 时间复杂度分析

状态共 4*n*m 种，每种状态入队一次，出队一次，对每个状态的操作所需时间为 O(1)，故最终总时间复杂度为 O(m*n)。

6. 实例

6.1. 实例 1

In.dat:

```
5 6
0 1 0 2 2 3
```

0 0 0 0 3 0

0 3 2 1 4 5

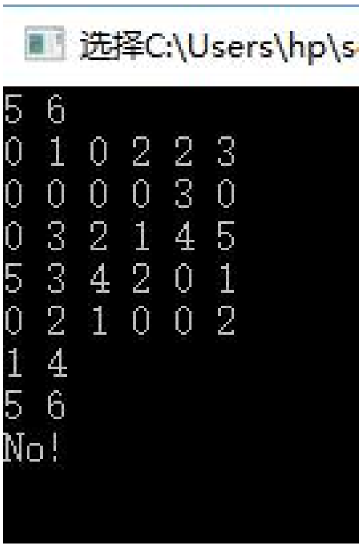
5 3 4 2 0 1

0 2 1 0 0 2

1 4

5 6

运行结果：



6.2. 实例 2

In.dat:

5 6

0 1 0 2 2 3

0 0 0 0 3 0

0 3 2 1 4 5

5 3 4 2 0 1

0 2 1 0 0 2

5 2

5 6

运行结果：