

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 1 | 0 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 2 | 0 | 4 | 5 | 6 | 7 | 8 | 9 | 0 |
| 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

如何用广度优先搜索，来寻找这些细胞。细胞的特征是什么？特征很明显是非0。非0数字连起来就是一个细胞。

在这里，我们将0号当成障碍点。那么，本题就转化成了一个地图搜索。

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | 2 | 3 | 4 | 5 | | 6 | 7 |
| 1 | | 3 | 4 | 5 | 6 | 5 | |
| 2 | | 4 | 5 | 6 | | 6 | 7 |
| | | | | | | 8 | 9 |

因此，我们可以这样来思考：遍历全部的地图上的点，只要这个点非0，且未被行走过，那么它将引导出来一个全新的细胞。

编码

```
#include <bits/stdc++.h>

using namespace std;

//第一维代表行，第二维代表列
char Maps[105][105];
bool Vis[105][105];

//存储关键信息
struct Node {
    //一般来说存储坐标，步数
    int x, y;

    Node(int nx, int ny) {
        x = nx, y = ny;
    }
};

int Forward[4][2] = {
    {-1, 0}, //向上
    {1, 0}, //向下
    {0, -1}, //向左
    {0, 1}, //向右
};

int n, m; //分别代表行和列

queue<Node> queues;

//1、没有被访问过
//2、不等于0
//3、判断没有越界
bool Check(Node newNode) {
```

```

    //不能等于0;
    bool res1 = (Maps[newNode.x][newNode.y] != '0');
    //不能被访问过
    bool res2 = !Vis[newNode.x][newNode.y];
    //没有越界
    bool res3 = (newNode.x >= 0 && newNode.x < n
                 && newNode.y >= 0 && newNode.y < m);
    //三个条件必须同时满足
    return res1 && res2 && res3;
}

//深搜
void Dfs(Node start) {
    for (int i = 0; i < 4; ++i) {
        int newX = start.x + Forward[i][0];
        int newY = start.y + Forward[i][1];
        Node newNode = Node(newX, newY);
        //必须通过条件判断, 才能放入待搜索列表
        if (Check(newNode)) {
            Vis[newX][newY] = true;
            Dfs(newNode);
        }
    }
}

//广搜
void Bfs(Node start) {
    //队列清空
    while (!queues.empty()) {
        queues.pop();
    }
    //把起始点放入队列
    queues.push(start);
    while (!queues.empty()) {
        Node node = queues.front();
        queues.pop();
        //延四个方向进行搜索
        for (int i = 0; i < 4; ++i) {
            int newX = node.x + Forward[i][0];
            int newY = node.y + Forward[i][1];
            Node newNode = Node(newX, newY);
            //必须通过条件判断, 才能放入待搜索列表
            if (Check(newNode)) {
                queues.push(newNode);
                Vis[newX][newY] = true;
            }
        }
    }
}

```

//步骤:

//1、读入数据, 将点存入maps数组。

//2、数字不为0, 且没有被访问过, 这就是一个全新的细胞

//3、使用dfs将当前数字连接的数字, 标记为已访问

```
int main() {  
    int cnt; //细胞总量  
    cin >> n >> m;  
    for (int i = 0; i < n; ++i) {  
        for (int j = 0; j < m; ++j) {  
            cin >> Maps[i][j];  
        }  
    }  
    for (int i = 0; i < n; ++i) {  
        for (int j = 0; j < m; ++j) {  
            //数字不等于0, 并且没有被访问过  
            if (Maps[i][j] != '0' && !Vis[i][j]) {  
                //全新的细胞  
                cnt++;  
                //把当前点标记成已经通过  
                Vis[i][j] = true;  
                //广搜方案  
                Bfs(Node(i, j));  
                //深搜方案  
                //Dfs(Node(i, j));  
                //二者任选其一  
            }  
        }  
    }  
    cout << cnt;  
    return 0;  
}
```

逻辑航线培优教育, 信息学奥赛培训专家。

扫码添加作者获取更多内容。

