【DFS/BFS】-开心消消乐

题目描述与示例

题目描述

给定一个 `N` 行 `M` 列的二维矩阵，矩阵中每个位置的数字取值为 `0` 或 `1`，矩阵示例如：

```
1 1 0 0
0 0 0 1
0 0 1 1
1 1 1 1
```

现需要将矩阵中所有的 `1` 进行反转为 `0`，规则如下：

1. 当点击一个 `1` 时，该 `1` 被反转为 `0`，同时相邻的上、下、左、右，以及左上、左下、右上、右下 `8` 个方向的 `1` （如果存在 `1`）均会自动反转为 0；
2. 进一步地，一个位置上的 `1` 被反转为 `0` 时，与其相邻的 `8` 个方向的 `1` （如果存在 `1`）均会自动反转为 `0`。

按照上述规则示例中的矩阵只最少需要点击 `2` 次后，所有均值 `0` 。请问，给定一个矩阵，最少需要点击几次后，所有数字均为 `0`？

输入

第一行输入两个整数，分别表示矩阵的行数 `N` 和列数 `M`，取值范围均为 `[1,100]`
接下来 `N` 行表示矩阵的初始值，每行均为 `M` 个数，取值范围 `[0,1]`

输出

输出一个整数，表示最少需要点击的次数

示例一

输入

```
3 3
1 0 1
0 1 0
```

```
1 0 1
```

输出

```
1
```

说明

上述样例中，四个角上的 `1` 均在中间的 `1` 的相邻 `8` 个方向上，因此只需要点击一次即可。

示例二

输入

```
4 4
1 1 0 0
0 0 0 1
0 0 1 1
1 1 1 1
```

输出

```
2
```

解题思路

> 注意，本题和LC200. 岛屿数量几乎完全一致。唯一的区别在于，本题需要**考虑八个方向而不是四个方向**。

考虑八个方向时，我们需要定义方向数组为

```
DIRECTIONS = [(0,1), (1,0), (-1,0), (0,-1), (1,1), (1,-1), (-1,1), (-1,-1)]
```

剩余过程就是常规的DFS/BFS过程。

代码

解法一：BFS

Python

```python
# 题目：2023Q1A-开心消消乐
# 分值：100
# 作者：闭着眼睛学数理化
# 算法：BFS
# 代码看不懂的地方，请直接在群上提问


from collections import deque

# 表示八个方向的数组
DIRECTIONS = [(0,1), (1,0), (-1,0), (0,-1), (1,1), (1,-1), (-1,1), (-1,-1)]

# 输入行数、列数
n, m = map(int, input().split())
grid = list()
for i in range(n):
    row = input().split()
    grid.append(row)


# 答案变量，用于记录连通块的个数
ans = 0
# 用于检查的二维矩阵
# 0表示没检查过，1表示检查过了
check_list = [[0] * m for _ in range(n)]

# 最外层的大的双重循环，是用来找BFS的起始搜索位置的
for i in range(n):
    for j in range(m):
        # 找到一个1，并且这个1从未被搜索过：那么可以进行BFS的搜索
        # 1. 值是1        2. 没被搜索过
        if grid[i][j] == "1" and check_list[i][j] == 0:
            # BFS的过程
            q = deque()
            q.append([i, j])        # BFS的起始点
            check_list[i][j] = 1
            while(len(q) > 0):      # 当队列中还有元素时，持续地进行搜索
```

```python
                qSize = len(q)
                for _ in range(qSize):
                    # 弹出队头元素，为当前点
                    x, y = q.popleft()
                    for dx, dy in DIRECTIONS:
                        nxt_x, nxt_y = x+dx, y+dy
                        # 若下一个点要加入队列，应该满足以下三个条件：
                        # 1.没有越界
                        # 2.在grid中值为"1"
                        # 3.尚未被检查过
                        if 0 <= nxt_x < n and 0 <= nxt_y < m:       # 越界判断
                            # 在grid中为"1"，尚未被检查过
                            if grid[nxt_x][nxt_y] == "1" and check_list[nxt_x][nxt_y] == 0:
                                q.append([nxt_x, nxt_y])        # 入队
                                check_list[nxt_x][nxt_y] = 1    # 标记为已检查过

            # BFS搜索完成，多了一个连通块
            ans += 1

print(ans)
```

Java

```java
import java.util.ArrayDeque;
import java.util.Queue;
import java.util.Scanner;

public class Main {
    static int[][] DIRECTIONS = {{0, 1}, {1, 0}, {-1, 0}, {0, -1}, {1, 1}, {1, -1}, {-1, 1}, {-1, -1}};
    static int n, m;
    static String[][] grid;
    static int[][] checkList;

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        n = scanner.nextInt();
        m = scanner.nextInt();
        grid = new String[n][m];
        checkList = new int[n][m];

        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                grid[i][j] = scanner.next();
            }
        }

        int ans = 0;

        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                if (grid[i][j].equals("1") && checkList[i][j] == 0) {
                    bfs(i, j);
                    ans++;
                }
            }
        }

        System.out.println(ans);
    }

    static void bfs(int i, int j) {
        Queue<int[]> queue = new ArrayDeque<>();
        queue.offer(new int[]{i, j});
        checkList[i][j] = 1;

        while (!queue.isEmpty()) {
            int[] current = queue.poll();
            int x = current[0];
            int y = current[1];

            for (int[] dir : DIRECTIONS) {
                int nxt_x = x + dir[0];
                int nxt_y = y + dir[1];

                if (nxt_x >= 0 && nxt_x < n && nxt_y >= 0 && nxt_y < m &&
                        grid[nxt_x][nxt_y].equals("1") && checkList[nxt_x][nxt_y] == 0) {
                    queue.offer(new int[]{nxt_x, nxt_y});
                    checkList[nxt_x][nxt_y] = 1;
                }
            }
        }
    }
}

```

C++

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

int DIRECTIONS[][2] = {{0, 1}, {1, 0}, {-1, 0}, {0, -1}, {1, 1}, {1, -1}, {-1, 1}, {-1, -1}};
int n, m;
vector<vector<string>> grid;
vector<vector<int>> checkList;

void bfs(int i, int j) {
    queue<pair<int, int>> q;
    q.push({i, j});
    checkList[i][j] = 1;

    while (!q.empty()) {
        pair<int, int> current = q.front();
        q.pop();
        int x = current.first;
        int y = current.second;

        for (auto dir : DIRECTIONS) {
            int nxt_x = x + dir[0];
            int nxt_y = y + dir[1];

            if (nxt_x >= 0 && nxt_x < n && nxt_y >= 0 && nxt_y < m &&
                grid[nxt_x][nxt_y] == "1" && checkList[nxt_x][nxt_y] == 0) {
                q.push({nxt_x, nxt_y});
                checkList[nxt_x][nxt_y] = 1;
            }
        }
    }
}

int main() {
    cin >> n >> m;
    grid.resize(n, vector<string>(m));
    checkList.resize(n, vector<int>(m, 0));

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            cin >> grid[i][j];
        }
    }

    int ans = 0;

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            if (grid[i][j] == "1" && checkList[i][j] == 0) {
                bfs(i, j);
                ans++;
            }
        }
    }

    cout << ans << endl;

    return 0;
}

```

时空复杂度

时间复杂度：`O(MN)`。

空间复杂度：`O(MN)`。

解法二：DFS

Python

```python
# 题目：2023Q1A-开心消消乐
# 分值：100
# 作者：闭着眼睛学数理化
# 算法：DFS
# 代码看不懂的地方，请直接在群上提问

# 表示八个方向的数组
DIRECTIONS = [(0,1), (1,0), (-1,0), (0,-1), (1,1), (1,-1), (-1,1), (-1,-1)]

# 输入行数、列数
n, m = map(int, input().split())
grid = list()
for i in range(n):
    row = input().split()
    grid.append(row)
```

```python


# 答案变量，用于记录连通块的个数
ans = 0
# 用于检查的二维矩阵
# 0表示没检查过，1表示检查过了
check_list = [[0] * m for _ in range(n)]

# dfs递归函数
def dfs(check_list, x, y):
    # 将点(x, y)标记为已检查过
    check_list[x][y] = 1
    for dx, dy in DIRECTIONS:
        nxt_x, nxt_y = x + dx, y + dy
        # 若下一个点继续进行dfs，应该满足以下三个条件：
        # 1.没有越界
        # 2.在grid中值为"1"
        # 3.尚未被检查过
        if 0 <= nxt_x < n and 0 <= nxt_y < m:        # 越界判断
            # 在grid中为"1"，尚未被检查过
            # 可以进行dfs
            if grid[nxt_x][nxt_y] == "1" and check_list[nxt_x][nxt_y] == 0:
                dfs(check_list, nxt_x, nxt_y)


# 最外层的大的双重循环，是用来找DFS的起始搜索位置的
for i in range(n):
    for j in range(m):
        # 找到一个"1"，并且这个"1"从未被搜索过：那么可以进行DFS的搜索
        # 1. 值得是"1"       2. 没被搜索过
        if grid[i][j] == "1" and check_list[i][j] == 0:
            dfs(check_list, i, j)
            # DFS搜索完成，多了一个连通块
            ans += 1

print(ans)
```

Java

```java
import java.util.Scanner;

public class Main {
    static int[][] DIRECTIONS = {{0, 1}, {1, 0}, {-1, 0}, {0, -1}, {1, 1}, {1, -1}, {-1, 1}, {-1, -1}};
    static int n, m;
    static String[][] grid;
    static int[][] checkList;

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        n = scanner.nextInt();
        m = scanner.nextInt();
        grid = new String[n][m];
        checkList = new int[n][m];

        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                grid[i][j] = scanner.next();
            }
        }

        int ans = 0;

        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                if (grid[i][j].equals("1") && checkList[i][j] == 0) {
                    dfs(i, j);
                    ans++;
                }
            }
        }

        System.out.println(ans);
    }

    static void dfs(int x, int y) {
        checkList[x][y] = 1;

        for (int[] dir : DIRECTIONS) {
            int nxt_x = x + dir[0];
            int nxt_y = y + dir[1];

            if (nxt_x >= 0 && nxt_x < n && nxt_y >= 0 && nxt_y < m &&
                    grid[nxt_x][nxt_y].equals("1") && checkList[nxt_x][nxt_y]
== 0) {
                dfs(nxt_x, nxt_y);
            }
        }
    }
}
```

C++

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

int DIRECTIONS[][2] = {{0, 1}, {1, 0}, {-1, 0}, {0, -1}, {1, 1}, {1, -1}, {-1, 1}, {-1, -1}};
int n, m;
vector<vector<string>> grid;
vector<vector<int>> checkList;

void dfs(int x, int y) {
    checkList[x][y] = 1;

    for (auto dir : DIRECTIONS) {
        int nxt_x = x + dir[0];
        int nxt_y = y + dir[1];

        if (nxt_x >= 0 && nxt_x < n && nxt_y >= 0 && nxt_y < m &&
            grid[nxt_x][nxt_y] == "1" && checkList[nxt_x][nxt_y] == 0) {
                dfs(nxt_x, nxt_y);
        }
    }
}

int main() {
    cin >> n >> m;
    grid.resize(n, vector<string>(m));
    checkList.resize(n, vector<int>(m, 0));

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            cin >> grid[i][j];
        }
    }

    int ans = 0;

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            if (grid[i][j] == "1" && checkList[i][j] == 0) {
                dfs(i, j);
                ans++;
            }
        }
    }

    cout << ans << endl;

    return 0;
}

```

时空复杂度

时间复杂度： `O(MN)`。

空间复杂度： `O(MN)`。