

【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 1 0 0
2 0 0 0 1
3 0 0 1 1
4 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

```
1 3 3
2 1 0 1
3 0 1 0
```

```
4  1 0 1
```

输出

```
1  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

```
1  4 4
2  1 1 0 0
3  0 0 0 1
4  0 0 1 1
5  1 1 1 1
```

输出

```
1  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

```
1 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

```
1 # 题目：2023Q1A-开心消消乐
2 # 分值：100
3 # 作者：闭着眼睛学数理化
4 # 算法：BFS
5 # 代码看不懂的地方，请直接在群上提问
6
7
8 from collections import deque
9
10 # 表示八个方向的数组
11 DIRECTIONS = [(0,1), (1,0), (-1,0), (0,-1), (1,1), (1,-1), (-1,1), (-1,-1)]
12
13 # 输入行数、列数
14 n, m = map(int, input().split())
15 grid = list()
16 for i in range(n):
17     row = input().split()
18     grid.append(row)
19
20
21 # 答案变量，用于记录连通块的个数
22 ans = 0
23 # 用于检查的二维矩阵
24 # 0表示没检查过，1表示检查过了
25 check_list = [[0] * m for _ in range(n)]
26
27 # 最外层的大双重循环，是用来找BFS的起始搜索位置的
28 for i in range(n):
29     for j in range(m):
30         # 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
31         # 1. 值是1      2. 没被搜索过
32         if grid[i][j] == "1" and check_list[i][j] == 0:
33             # BFS的过程
34             q = deque()
35             q.append([i, j])      # BFS的起始点
36             check_list[i][j] = 1
37             while(len(q) > 0):      # 当队列中还有元素时，持续地进行搜索
```

```

38         qSize = len(q)
39         for _ in range(qSize):
40             # 弹出队头元素，为当前点
41             x, y = q.popleft()
42             for dx, dy in DIRECTIONS:
43                 nxt_x, nxt_y = x+dx, y+dy
44                 # 若下一个点要加入队列，应该满足以下三个条件：
45                 # 1.没有越界
46                 # 2.在grid中值为"1"
47                 # 3.尚未被检查过
48                 if 0 <= nxt_x < n and 0 <= nxt_y < m:          # 越界判断
49                     # 在grid中为"1"，尚未被检查过
50                     if grid[nxt_x][nxt_y] == "1" and check_list[nxt_x]
[nxt_y] == 0:
51                         q.append([nxt_x, nxt_y])          # 入队
52                         check_list[nxt_x][nxt_y] = 1      # 标记为已检查过
53
54             # BFS搜索完成，多了一个连通块
55             ans += 1
56
57 print(ans)

```

Java

```

1 import java.util.ArrayDeque;
2 import java.util.Queue;
3 import java.util.Scanner;
4
5 public class Main {
6     static int[][] DIRECTIONS = {{0, 1}, {1, 0}, {-1, 0}, {0, -1}, {1, 1}, {1,
-1}, {-1, 1}, {-1, -1}};
7     static int n, m;
8     static String[][] grid;
9     static int[][] checkList;
10
11     public static void main(String[] args) {
12         Scanner scanner = new Scanner(System.in);
13
14         n = scanner.nextInt();
15         m = scanner.nextInt();
16         grid = new String[n][m];
17         checkList = new int[n][m];
18
19         for (int i = 0; i < n; i++) {
20             for (int j = 0; j < m; j++) {

```

```

21         grid[i][j] = scanner.next();
22     }
23 }
24
25 int ans = 0;
26
27 for (int i = 0; i < n; i++) {
28     for (int j = 0; j < m; j++) {
29         if (grid[i][j].equals("1") && checkList[i][j] == 0) {
30             bfs(i, j);
31             ans++;
32         }
33     }
34 }
35
36 System.out.println(ans);
37 }
38
39 static void bfs(int i, int j) {
40     Queue<int[]> queue = new ArrayDeque<>();
41     queue.offer(new int[]{i, j});
42     checkList[i][j] = 1;
43
44     while (!queue.isEmpty()) {
45         int[] current = queue.poll();
46         int x = current[0];
47         int y = current[1];
48
49         for (int[] dir : DIRECTIONS) {
50             int nxt_x = x + dir[0];
51             int nxt_y = y + dir[1];
52
53             if (nxt_x >= 0 && nxt_x < n && nxt_y >= 0 && nxt_y < m &&
54 [nxt_y] == 0) {
55                 queue.offer(new int[]{nxt_x, nxt_y});
56                 checkList[nxt_x][nxt_y] = 1;
57             }
58         }
59     }
60 }
61 }
62

```

```

1  #include <iostream>
2  #include <vector>
3  #include <queue>
4  using namespace std;
5
6  int DIRECTIONS[][2] = {{0, 1}, {1, 0}, {-1, 0}, {0, -1}, {1, 1}, {1, -1}, {-1,
    1}, {-1, -1}};
7  int n, m;
8  vector<vector<string>> grid;
9  vector<vector<int>> checkList;
10
11 void bfs(int i, int j) {
12     queue<pair<int, int>> q;
13     q.push({i, j});
14     checkList[i][j] = 1;
15
16     while (!q.empty()) {
17         pair<int, int> current = q.front();
18         q.pop();
19         int x = current.first;
20         int y = current.second;
21
22         for (auto dir : DIRECTIONS) {
23             int nxt_x = x + dir[0];
24             int nxt_y = y + dir[1];
25
26             if (nxt_x >= 0 && nxt_x < n && nxt_y >= 0 && nxt_y < m &&
27                 grid[nxt_x][nxt_y] == "1" && checkList[nxt_x][nxt_y] == 0) {
28                 q.push({nxt_x, nxt_y});
29                 checkList[nxt_x][nxt_y] = 1;
30             }
31         }
32     }
33 }
34
35 int main() {
36     cin >> n >> m;
37     grid.resize(n, vector<string>(m));
38     checkList.resize(n, vector<int>(m, 0));
39
40     for (int i = 0; i < n; i++) {
41         for (int j = 0; j < m; j++) {
42             cin >> grid[i][j];
43         }
44     }
45
46     int ans = 0;

```

```

47
48     for (int i = 0; i < n; i++) {
49         for (int j = 0; j < m; j++) {
50             if (grid[i][j] == "1" && checkList[i][j] == 0) {
51                 bfs(i, j);
52                 ans++;
53             }
54         }
55     }
56
57     cout << ans << endl;
58
59     return 0;
60 }
61

```

时空复杂度

时间复杂度： $O(MN)$ 。

空间复杂度： $O(MN)$ 。

解法二：DFS

Python

```

1  # 题目：2023Q1A-开心消消乐
2  # 分值：100
3  # 作者：闭着眼睛学数理化
4  # 算法：DFS
5  # 代码看不懂的地方，请直接在群上提问
6
7  # 表示八个方向的数组
8  DIRECTIONS = [(0,1), (1,0), (-1,0), (0,-1), (1,1), (1,-1), (-1,1), (-1,-1)]
9
10 # 输入行数、列数
11 n, m = map(int, input().split())
12 grid = list()
13 for i in range(n):
14     row = input().split()
15     grid.append(row)

```

```

16
17
18 # 答案变量，用于记录连通块的个数
19 ans = 0
20 # 用于检查的二维矩阵
21 # 0表示没检查过，1表示检查过了
22 check_list = [[0] * m for _ in range(n)]
23
24 # dfs递归函数
25 def dfs(check_list, x, y):
26     # 将点(x, y)标记为已检查过
27     check_list[x][y] = 1
28     for dx, dy in DIRECTIONS:
29         nxt_x, nxt_y = x + dx, y + dy
30         # 若下一个点继续进行dfs，应该满足以下三个条件：
31         # 1. 没有越界
32         # 2. 在grid中值为"1"
33         # 3. 尚未被检查过
34         if 0 <= nxt_x < n and 0 <= nxt_y < m:          # 越界判断
35             # 在grid中为"1"，尚未被检查过
36             # 可以进行dfs
37             if grid[nxt_x][nxt_y] == "1" and check_list[nxt_x][nxt_y] == 0:
38                 dfs(check_list, nxt_x, nxt_y)
39
40
41 # 最外层的大的双重循环，是用来找DFS的起始搜索位置的
42 for i in range(n):
43     for j in range(m):
44         # 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
45         # 1. 值得是"1"          2. 没被搜索过
46         if grid[i][j] == "1" and check_list[i][j] == 0:
47             dfs(check_list, i, j)
48             # DFS搜索完成，多了一个连通块
49             ans += 1
50
51 print(ans)

```

Java

```

1 import java.util.Scanner;
2
3 public class Main {
4     static int[][] DIRECTIONS = {{0, 1}, {1, 0}, {-1, 0}, {0, -1}, {1, 1}, {1,
5     -1}, {-1, 1}, {-1, -1}};
6     static int n, m;

```



```

6     static String[][] grid;
7     static int[][] checkList;
8
9     public static void main(String[] args) {
10         Scanner scanner = new Scanner(System.in);
11
12         n = scanner.nextInt();
13         m = scanner.nextInt();
14         grid = new String[n][m];
15         checkList = new int[n][m];
16
17         for (int i = 0; i < n; i++) {
18             for (int j = 0; j < m; j++) {
19                 grid[i][j] = scanner.next();
20             }
21         }
22
23         int ans = 0;
24
25         for (int i = 0; i < n; i++) {
26             for (int j = 0; j < m; j++) {
27                 if (grid[i][j].equals("1") && checkList[i][j] == 0) {
28                     dfs(i, j);
29                     ans++;
30                 }
31             }
32         }
33
34         System.out.println(ans);
35     }
36
37     static void dfs(int x, int y) {
38         checkList[x][y] = 1;
39
40         for (int[] dir : DIRECTIONS) {
41             int nxt_x = x + dir[0];
42             int nxt_y = y + dir[1];
43
44             if (nxt_x >= 0 && nxt_x < n && nxt_y >= 0 && nxt_y < m &&
45                 grid[nxt_x][nxt_y].equals("1") && checkList[nxt_x][nxt_y]
46 == 0) {
47                 dfs(nxt_x, nxt_y);
48             }
49         }
50     }
51

```

C++

```
1  #include <iostream>
2  #include <vector>
3  using namespace std;
4
5  int DIRECTIONS[][2] = {{0, 1}, {1, 0}, {-1, 0}, {0, -1}, {1, 1}, {1, -1}, {-1,
    1}, {-1, -1}};
6  int n, m;
7  vector<vector<string>> grid;
8  vector<vector<int>> checkList;
9
10 void dfs(int x, int y) {
11     checkList[x][y] = 1;
12
13     for (auto dir : DIRECTIONS) {
14         int nxt_x = x + dir[0];
15         int nxt_y = y + dir[1];
16
17         if (nxt_x >= 0 && nxt_x < n && nxt_y >= 0 && nxt_y < m &&
18             grid[nxt_x][nxt_y] == "1" && checkList[nxt_x][nxt_y] == 0) {
19             dfs(nxt_x, nxt_y);
20         }
21     }
22 }
23
24 int main() {
25     cin >> n >> m;
26     grid.resize(n, vector<string>(m));
27     checkList.resize(n, vector<int>(m, 0));
28
29     for (int i = 0; i < n; i++) {
30         for (int j = 0; j < m; j++) {
31             cin >> grid[i][j];
32         }
33     }
34
35     int ans = 0;
36
37     for (int i = 0; i < n; i++) {
38         for (int j = 0; j < m; j++) {
39             if (grid[i][j] == "1" && checkList[i][j] == 0) {
40                 dfs(i, j);
41                 ans++;
42             }
43         }
44     }
45 }
```

```
43     }  
44 }  
45  
46     cout << ans << endl;  
47  
48     return 0;  
49 }  
50
```

时空复杂度

时间复杂度： $O(MN)$ 。

空间复杂度： $O(MN)$ 。