

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：

1100
0001
0011
1111

现需要将矩阵中所有的1进行反转，规则如下：

- 1) 当点击一个1时，该1便被反转为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，所有值均为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？

输入描述

第一行输入两个数字N, M，分别表示二维矩阵的行数、列数，并用空格隔开

之后输入N行，每行M个数字，并用空格隔开

输出描述

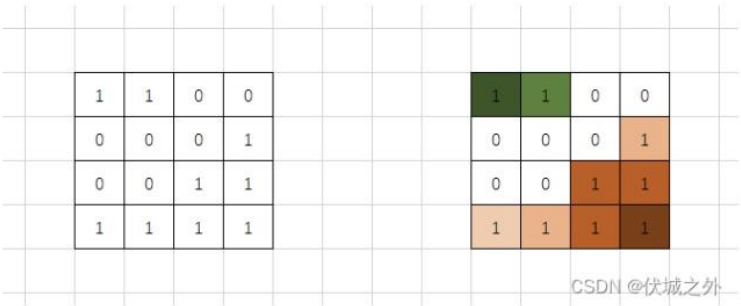
最少需要点击几次后，矩阵中所有数字均为0

用例

输入	4 4
	1 1 0 0
	0 0 0 1
	0 0 1 1
输出	2
说明	无

题目解析

用例图示如下



可以发现，只要是连接在一起的1（八个方向都算连接），点击任意1个，都会蔓延到相连的其他1。因此，本题重点不在于点击哪个1，而是有多少块连在一起的1。即孤岛问题，求解不连通的岛屿数量。孤岛问题可以使用并差集求解。

本题类似于[LeetCode - 200 岛屿数量_伏城之外的博客-CSDN博客](#)

题解可以看这个[博客](#)。

JavaScript算法源码

```
1 /* JavaScript Node ACM模式 控制台输入获取 */
2 const readline = require("readline");
3
4 const rl = readline.createInterface({
5   input: process.stdin,
6   output: process.stdout,
7 });
8
```

伏城之外 已关注

1 6 7 专栏目录 已订阅

```

1  /* JavaScript Node ACM模式 控制台输入获取 */
2  const readline = require("readline");
3
4  const rl = readline.createInterface({
5    input: process.stdin,
6    output: process.stdout,
7  });
8
9  const lines = [];
10 let n, m;
11 rl.on("line", (line) => {
12   lines.push(line);
13
14   if (lines.length === 1) {
15     [n, m] = lines[0].split(" ").map(Number);
16   }
17
18   if (n && lines.length === n + 1) {
19     lines.shift();
20     const matrix = lines.map((line) => line.split(" "));
21     console.log(getResult(matrix, n, m));
22     lines.length = 0;
23   }
24 });
25
26 function getResult(matrix, n, m) {
27   const ufs = new UnionFindSet(n * m);
28
29   // 八个方向的偏移量
30   const offset = [
31     [-1, -1],
32     [-1, 0],
33     [-1, 1],
34     [0, -1],
35     [0, 1],
36     [1, -1],
37     [1, 0],
38     [1, 1],
39   ];
40
41   for (let i = 0; i < n; i++) {
42     for (let j = 0; j < m; j++) {
43       if (matrix[i][j] !== "1") {
44         ufs.count--;
45         continue;
46       }
47       for (let k = 0; k < offset.length; k++) {
48         const [offsetX, offsetY] = offset[k];
49         const newI = i + offsetX;
50         const newJ = j + offsetY;
51
52         if (
53           newI >= 0 &&
54           newI < n &&
55           newJ >= 0 &&
56           newJ < m &&
57           matrix[newI][newJ] === "1"
58         ) {
59           ufs.union(i * m + j, newI * m + newJ);
60         }
61       }
62     }
63   }
64
65   return ufs.count;
66 }
67
68 class UnionFindSet {
69   constructor(n) {
70     this.fa = new Array(n).fill(0).map((_, i) => i);
71     this.count = n;
72   }
73
74   find(x) {
75     if (x !== this.fa[x]) {
76       return (this.fa[x] = this.find(this.fa[x]));
77     }
78     return x;
79   }
80
81   union(x, y) {
82     const x_fa = this.find(x);
83     const y_fa = this.find(y);
84
85     if (x_fa !== y_fa) {
86       this.fa[y_fa] = x_fa;
87       this.count--;
88     }
89   }
90 }

```



```

65     return visitCount;
66 }
67
68 class UnionFindSet {
69     constructor(n) {
70         this.fa = new Array(n).fill(0).map((_, i) => i);
71         this.count = n;
72     }
73
74     find(x) {
75         if (x !== this.fa[x]) {
76             return (this.fa[x] = this.find(this.fa[x]));
77         }
78         return x;
79     }
80
81     union(x, y) {
82         const x_fa = this.find(x);
83         const y_fa = this.find(y);
84
85         if (x_fa !== y_fa) {
86             this.fa[y_fa] = x_fa;
87             this.count--;
88         }
89     }
90 }

```

Java算法源码

```

1  import java.util.*;
2
3  public class Main {
4      public static void main(String[] args) {
5          Scanner sc = new Scanner(System.in);
6
7          int n = sc.nextInt();
8          int m = sc.nextInt();
9
10         int[][] matrix = new int[n][m];
11         for (int i = 0; i < n; i++) {
12             for (int j = 0; j < m; j++) {
13                 matrix[i][j] = sc.nextInt();
14             }
15         }
16
17         System.out.println(getResult(matrix, n, m));
18     }
19
20     public static int getResult(int[][] matrix, int n, int m) {
21         UnionFindSet ufs = new UnionFindSet(n * m);
22
23         // 八个方向的偏移量
24         Integer[][] offsets = {
25             {-1, -1}, {-1, 0}, {-1, 1}, {0, -1}, {0, 1}, {1, -1}, {1, 0}, {1, 1}
26         };
27
28         for (int i = 0; i < n; i++) {
29             for (int j = 0; j < m; j++) {
30                 if (matrix[i][j] != 1) {
31                     ufs.count--;
32                     continue;
33                 }
34
35                 // 不要紧张，这里固定循环8次，不会造成O(n^3)
36                 for (Integer[] offset : offsets) {
37                     int newI = i + offset[0];
38                     int newJ = j + offset[1];
39
40                     if (newI >= 0 && newI < n && newJ >= 0 && newJ < m && matrix[newI][newJ] == 1) {
41                         ufs.union(i * m + j, newI * m + newJ);
42                     }
43                 }
44             }
45         }
46
47         return ufs.count;
48     }
49 }
50
51 // 并查集
52 class UnionFindSet {
53     int[] fa;
54     int count;
55
56     public UnionFindSet(int n) {
57         this.fa = new int[n];
58         this.count = n;
59         for (int i = 0; i < n; i++) this.fa[i] = i;
60     }
61
62     public int find(int x) {
63         if (x != this.fa[x]) {
64             return (this.fa[x] = this.find(this.fa[x]));
65         }
66     }

```

Java算法源码

```
1 import java.util.*;
2
3 public class Main {
4     public static void main(String[] args) {
5         Scanner sc = new Scanner(System.in);
6
7         int n = sc.nextInt();
8         int m = sc.nextInt();
9
10        int[][] matrix = new int[n][m];
11        for (int i = 0; i < n; i++) {
12            for (int j = 0; j < m; j++) {
13                matrix[i][j] = sc.nextInt();
14            }
15        }
16
17        System.out.println(getResult(matrix, n, m));
18    }
19
20    public static int getResult(int[][] matrix, int n, int m) {
21        UnionFindSet ufs = new UnionFindSet(n * m);
22
23        // 八个方向的偏移量
24        Integer[][] offsets = {
25            {-1, -1}, {-1, 0}, {-1, 1}, {0, -1}, {0, 1}, {1, -1}, {1, 0}, {1, 1}
26        };
27
28        for (int i = 0; i < n; i++) {
29            for (int j = 0; j < m; j++) {
30                if (matrix[i][j] != 1) {
31                    ufs.count--;
32                    continue;
33                }
34
35                // 不要紧张, 这里固定循环8次, 不会造成O(n^3)
36                for (Integer[] offset : offsets) {
37                    int newI = i + offset[0];
38                    int newJ = j + offset[1];
39
40                    if (newI >= 0 && newI < n && newJ >= 0 && newJ < m && matrix[newI][newJ] == 1) {
41                        ufs.union(i * m + j, newI * m + newJ);
42                    }
43                }
44            }
45        }
46
47        return ufs.count;
48    }
49 }
50
51 // 并查集
52 class UnionFindSet {
53     int[] fa;
54     int count;
55
56     public UnionFindSet(int n) {
57         this.fa = new int[n];
58         this.count = n;
59         for (int i = 0; i < n; i++) this.fa[i] = i;
60     }
61
62     public int find(int x) {
63         if (x != this.fa[x]) {
64             return (this.fa[x] = this.find(this.fa[x]));
65         }
66         return x;
67     }
68
69     public void union(int x, int y) {
70         int x_fa = this.find(x);
71         int y_fa = this.find(y);
72
73         if (x_fa != y_fa) {
74             this.fa[y_fa] = x_fa;
75             this.count--;
76         }
77     }
78 }
```

Python算法源码

```
1 # 并查集
2 class UnionFindSet:
3     def __init__(self, n):
4         self.fa = [idx for idx in range(n)]
5         self.count = n
6
7     def find(self, x):
8         if x != self.fa[x]:
9             self.fa[x] = self.find(self.fa[x])
```

Python算法源码

```
1 # 并查集
2 class UnionFindSet:
3     def __init__(self, n):
4         self.fa = [idx for idx in range(n)]
5         self.count = n
6
7     def find(self, x):
8         if x != self.fa[x]:
9             self.fa[x] = self.find(self.fa[x])
10            return self.fa[x]
11        return x
12
13    def union(self, x, y):
14        x_fa = self.find(x)
15        y_fa = self.find(y)
16
17        if x_fa != y_fa:
18            self.fa[y_fa] = x_fa
19            self.count -= 1
20
21
22 n, m = map(int, input().split())
23
24 matrix = []
25 for i in range(n):
26     matrix.append(list(map(int, input().split())))
27
28 ufs = UnionFindSet(n * m)
29
30 # 八个方向的偏移量
31 offsets = ((-1, -1), (-1, 0), (-1, 1), (0, -1), (0, 1), (1, -1), (1, 0), (1, 1))
32
33 for i in range(n):
34     for j in range(m):
35         if matrix[i][j] != 1:
36             ufs.count -= 1
37             continue
38
39         for offsetX, offsetY in offsets:
40             newI = i + offsetX
41             newJ = j + offsetY
42
43             if 0 <= newI < n and 0 <= newJ < m and matrix[newI][newJ] == 1:
44                 ufs.union(i*m+j, newI*m+newJ)
45
46 print(ufs.count)
```