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进行反转为0，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

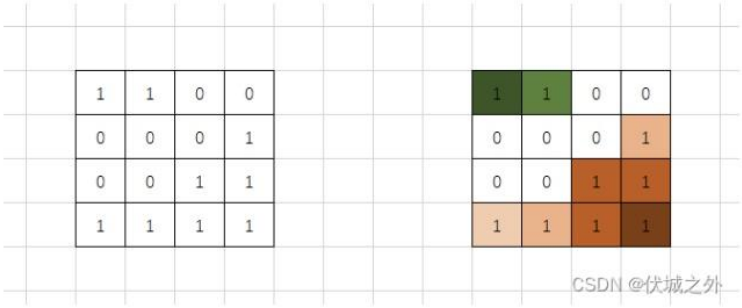
最少需要点击几次后，矩阵中所有数字均为0

用例

输入	4 4 1 1 0 0 0 0 0 1 0 0 1 1 1 1 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

```
/* JavaScript Node ACM模式 控制台输入获取 */
const readline = require("readline");

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout,
});

```

JavaScript算法源码

```javascript
/* JavaScript Node ACM模式 控制台输入获取 */
const readline = require("readline");

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout,
});

const lines = [];
let n, m;
rl.on("line", (line) => {
  lines.push(line);

  if (lines.length === 1) {
    [n, m] = lines[0].split(" ").map(Number);
  }

  if (n && lines.length === n + 1) {
    lines.shift();
    const matrix = lines.map((line) => line.split(" "));
    console.log(getResult(matrix, n, m));
    lines.length = 0;
  }
});

function getResult(matrix, n, m) {
  const ufs = new UnionFindSet(n * m);

  // 八个方向的偏移量
  const offset = [
    [-1, -1],
    [-1, 0],
    [-1, 1],
    [0, -1],
    [0, 1],
    [1, -1],
    [1, 0],
    [1, 1],
  ];

  for (let i = 0; i < n; i++) {
    for (let j = 0; j < m; j++) {
      if (matrix[i][j] != "1") {
        ufs.count--;
        continue;
      }
      for (let k = 0; k < offset.length; k++) {
        const [offsetX, offsetY] = offset[k];
        const newI = i + offsetX;
        const newJ = j + offsetY;

        if (
          newI >= 0 &&
          newI < n &&
          newJ >= 0 &&
          newJ < m &&
          matrix[newI][newJ] == "1"
        ) {
          ufs.union(i * m + j, newI * m + newJ);
        }
      }
    }
  }

  return ufs.count;
}

class UnionFindSet {
  constructor(n) {
    this.fa = new Array(n).fill(0).map((_, i) => i);
    this.count = n;
  }

  find(x) {
    if (x !== this.fa[x]) {
      return (this.fa[x] = this.find(this.fa[x]));
    }
    return x;
  }

  union(x, y) {
    const x_fa = this.find(x);
    const y_fa = this.find(y);

    if (x_fa !== y_fa) {
      this.fa[y_fa] = x_fa;
      this.count--;
    }
  }
}
```

```
}

class UnionFindSet {
  constructor(n) {
    this.fa = new Array(n).fill(0).map((_, i) => i);
    this.count = n;
  }

  find(x) {
    if (x !== this.fa[x]) {
      return (this.fa[x] = this.find(this.fa[x]));
    }
    return x;
  }

  union(x, y) {
    const x_fa = this.find(x);
    const y_fa = this.find(y);

    if (x_fa !== y_fa) {
      this.fa[y_fa] = x_fa;
      this.count--;
    }
  }
}
```

Java算法源码

```
import java.util.*;

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

        int n = sc.nextInt();
        int m = sc.nextInt();

        int[][] matrix = new int[n][m];
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                matrix[i][j] = sc.nextInt();
            }
        }

        System.out.println(getResult(matrix, n, m));
    }

    public static int getResult(int[][] matrix, int n, int m) {
        UnionFindSet ufs = new UnionFindSet(n * m);

        // 八个方向的偏移量
        Integer[][] offsets = {
                {-1, -1}, {-1, 0}, {-1, 1}, {0, -1}, {0, 1}, {1, -1}, {1, 0}, {1, 1}
        };

        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                if (matrix[i][j] != 1) {
                    ufs.count--;
                    continue;
                }

                // 不要紧张，这里固定循环8次，不会造成O(n^3)
                for (Integer[] offset : offsets) {
                    int newI = i + offset[0];
                    int newJ = j + offset[1];

                    if (newI >= 0 && newI < n && newJ >= 0 && newJ < m && matrix[newI][newJ] == 1) {
                        ufs.union(i * m + j, newI * m + newJ);
                    }
                }
            }
        }

        return ufs.count;
    }
}

// 并查集
class UnionFindSet {
    int[] fa;
    int count;

    public UnionFindSet(int n) {
        this.fa = new int[n];
        this.count = n;
        for (int i = 0; i < n; i++) this.fa[i] = i;
    }

    public int find(int x) {
        if (x != this.fa[x]) {
            return (this.fa[x] = this.find(this.fa[x]));
        }
```

Java算法源码

```java
import java.util.*;

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

        int n = sc.nextInt();
        int m = sc.nextInt();

        int[][] matrix = new int[n][m];
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                matrix[i][j] = sc.nextInt();
            }
        }

        System.out.println(getResult(matrix, n, m));
    }

    public static int getResult(int[][] matrix, int n, int m) {
        UnionFindSet ufs = new UnionFindSet(n * m);

        // 八个方向的偏移量
        Integer[][] offsets = {
                {-1, -1}, {-1, 0}, {-1, 1}, {0, -1}, {0, 1}, {1, -1}, {1, 0}, {1, 1}
        };

        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                if (matrix[i][j] != 1) {
                    ufs.count--;
                    continue;
                }

                // 不要紧张，这里固定循环8次，不会造成O(n^3)
                for (Integer[] offset : offsets) {
                    int newI = i + offset[0];
                    int newJ = j + offset[1];

                    if (newI >= 0 && newI < n && newJ >= 0 && newJ < m && matrix[newI][newJ] == 1) {
                        ufs.union(i * m + j, newI * m + newJ);
                    }
                }
            }
        }

        return ufs.count;
    }
}

// 并查集
class UnionFindSet {
    int[] fa;
    int count;

    public UnionFindSet(int n) {
        this.fa = new int[n];
        this.count = n;
        for (int i = 0; i < n; i++) this.fa[i] = i;
    }

    public int find(int x) {
        if (x != this.fa[x]) {
            return (this.fa[x] = this.find(this.fa[x]));
        }
        return x;
    }

    public void union(int x, int y) {
        int x_fa = this.find(x);
        int y_fa = this.find(y);

        if (x_fa != y_fa) {
            this.fa[y_fa] = x_fa;
            this.count--;
        }
    }
}
```

Python算法源码

```python
# 并查集
class UnionFindSet:
    def __init__(self, n):
        self.fa = [idx for idx in range(n)]
        self.count = n

    def find(self, x):
        if x != self.fa[x]:
            self.fa[x] = self.find(self.fa[x])
```

Python算法源码

```python
# 并查集
class UnionFindSet:
    def __init__(self, n):
        self.fa = [idx for idx in range(n)]
        self.count = n

    def find(self, x):
        if x != self.fa[x]:
            self.fa[x] = self.find(self.fa[x])
            return self.fa[x]
        return x

    def union(self, x, y):
        x_fa = self.find(x)
        y_fa = self.find(y)

        if x_fa != y_fa:
            self.fa[y_fa] = x_fa
            self.count -= 1


n, m = map(int, input().split())

matrix = []
for i in range(n):
    matrix.append(list(map(int, input().split())))

ufs = UnionFindSet(n * m)

# 八个方向的偏移量
offsets = ((-1, -1), (-1, 0), (-1, 1), (0, -1), (0, 1), (1, -1), (1, 0), (1, 1))

for i in range(n):
    for j in range(m):
        if matrix[i][j] != 1:
            ufs.count -= 1
            continue

        for offsetX, offsetY in offsets:
            newI = i + offsetX
            newJ = j + offsetY

            if 0 <= newI < n and 0 <= newJ < m and matrix[newI][newJ] == 1:
                ufs.union(i*m+j, newI*m+newJ)

print(ufs.count)
```