39、欢乐的周末，考点 or 实现——数据结构/并查集

题目描述

小华和小为是很要好的朋友，他们约定周末一起吃饭。

通过手机交流，他们在地图上选择了多个聚餐地点（由于自然地形等原因，部分聚餐地点不可达），求小华和小为都能到达的聚餐地点有多少个？

输入描述

第一行输入m和n，m代表地图的长度，n代表地图的宽度。

第二行开始具体输入地图信息，地图信息包含：

0 为通畅的道路

1 为障碍物（且仅1为障碍物）

2 为小华或者小为，地图中必定有且仅有2个 （非障碍物）

3 为被选中的聚餐地点（非障碍物）

输出描述

可以被两方都到达的聚餐地点数量，行末无空格。

用例

输入	4 4 2 1 0 3 0 1 2 1 0 3 0 0 0 0 0 0
输出	2
说明	第一行输入地图的长宽为3和4。 第二行开始为具体的地图，其中：3代表小华和小明选择的聚餐地点；2代表小华或者小明（确保有2个）；0代表可以通行的位置；1代表不可以通行的位置。 此时两者能都能到达的聚餐位置有2处。

输入	4 4 2 1 2 3 0 1 0 0 0 1 0 0 0 1 0 0
输出	0
说明	第一行输入地图的长宽为4和4。 第二行开始为具体的地图，其中：3代表小华和小明选择的聚餐地点；2代表小华或者小明（确保有2个）；0代表可以通行的位置；1代表不可以通行的位置。 由于图中小华和小为之间有个阻隔，此时，没有两人都能到达的聚餐地址，故而返回0。

备注：

地图的长宽为m和n，其中：

4 <= m <= 100

4 <= n <= 100

聚餐的地点数量为 k，则

1< k <= 100

题目解析

本题一开始我是考虑使用dfs来求解，但是发现递归结束条件不好定义。

后面想了一下，还是使用 并查集。

小华和小为想去同一个餐厅，那么必然小华和小为和餐厅是可以连通，如果它们不能连通，则去不了同一个餐厅。

因此，我们可以遍历矩阵中每一个元素，将它和其上下左右元素进行连接，需要注意的是如果遍历的元素本身是1，或者其上下左右的元素是1，则不进行连接。

这样的话，遍历完矩阵后，就可以得到一个 连通图。

同时在遍历矩阵过程中，记录小华、小为（值为2），以及餐厅（值为3）的位置，遍历结束后，首先看小华和小为是不是同一个祖先，若不是，则二者不可连通，就更别说去同一个餐厅了，因此返回0。若二者可以连通，则再看每一个餐厅的祖先是否和华为的祖先相同，若相同则计数++，这样就可以得到小华，小为去的同一个餐厅的数量了。

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

  if (n && lines.length === n + 1) {
    lines.shift();
    const grid = lines.map((line) => line.split(" ").map(Number));

    console.log(countAddress(grid));

    lines.length = 0;
```

```
  }
});

function countAddress(grid) {
  const n = grid.length;
  const m = grid[0].length;

  const ufs = new UnionFindSet(n * m);

  const huawei = [];
  const restrant = [];
  const offsets = [
    [-1, 0],
    [1, 0],
    [0, -1],
    [0, 1],
  ];

  for (let i = 0; i < n; i++) {
    for (let j = 0; j < m; j++) {
      if (grid[i][j] !== 1) {
        const x = i * m + j;
        if (grid[i][j] === 2) huawei.push(x);
        else if (grid[i][j] === 3) restrant.push(x);

        for (let offset of offsets) {
          const newI = i + offset[0];
```

```
          const newJ = j + offset[1];
          if (
            newI >= 0 &&
            newI < n &&
            newJ >= 0 &&
            newJ < m &&
            grid[newI][newJ] != 1
          ) {
            ufs.union(x, newI * m + newJ);
          }
        }
      }
    }
  }

  const [hua, wei] = huawei;
  const hua_fa = ufs.find(hua);
  const wei_fa = ufs.find(wei);
  if (hua_fa !== wei_fa) {
    return 0;
  }

  return restrant.filter((r) => ufs.find(r) === hua_fa).length;
}

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

```

  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;
    }
  }
}
```

Java算法源码

```java
import java.util.ArrayList;
import java.util.Scanner;

public class Main {

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

    int m = sc.nextInt();
    int n = 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(n, m, matrix));
  }

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

    ArrayList<Integer> huawei = new ArrayList<>();
    ArrayList<Integer> restaurants = new ArrayList<>();

    int[][] offsets = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};

    for (int i = 0; i < n; i++) {
      for (int j = 0; j < m; j++) {
        if (matrix[i][j] != 1) {
          int x = i * m + j;
          if (matrix[i][j] == 2) huawei.add(x);
          else if (matrix[i][j] == 3) restaurants.add(x);

          for (int[] 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(x, newI * m + newJ);
            }
          }
        }
      }
    }
```

```
47
48      int hua_fa = ufs.find(huawei.get(0));
49      int wei_fa = ufs.find(huawei.get(1));
50
51      if (hua_fa != wei_fa) {
52        return 0;
53      }
54
55      int ans = 0;
56      for (Integer restaurant : restaurants) {
57        if (ufs.find(restaurant) == hua_fa) {
58          ans++;
59        }
60      }
61
62      return ans;
63    }
64  }
65
66  // 并查集实现
67  class UnionFindSet {
68    int[] fa;
69
70    public UnionFindSet(int n) {
71      fa = new int[n];
72      for (int i = 0; i < n; i++) fa[i] = i;
73    }
74
75    public int find(int x) {
76      if (x != this.fa[x]) {
77        this.fa[x] = this.find(this.fa[x]);
78        return this.fa[x];
79      }
80      return x;
81    }
82
83    public void union(int x, int y) {
84      int x_fa = this.find(x);
85      int y_fa = this.find(y);
86
87      if (x_fa != y_fa) {
88        this.fa[y_fa] = x_fa;
89      }
90    }
91  }
```

Python算法源码

```python
# 输入获取
m, n = map(int, input().split())
matrix = [list(map(int, input().split())) for _ in range(n)]


# 并查集实现
class UnionFindSet:
    def __init__(self, n):
        self.fa = [i for i in range(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


# 算法入口
def getResult():
    ufs = UnionFindSet(n * m)

```

```
    huawei = []
    restaurants = []
    offsets = ((-1, 0), (1, 0), (0, -1), (0, 1))

    for i in range(n):
        for j in range(m):
            if matrix[i][j] != 1:
                x = i * m + j
                if matrix[i][j] == 2:
                    huawei.append(x)
                elif matrix[i][j] == 3:
                    restaurants.append(x)

                for offset in offsets:
                    newI = i + offset[0]
                    newJ = j + offset[1]

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

    hua_fa = ufs.find(huawei[0])
    wei_fa = ufs.find(huawei[1])

    if hua_fa != wei_fa:
        return 0

```

```
    ans = 0
    for r in restaurants:
        if ufs.find(r) == hua_fa:
            ans += 1

    return ans


# 算法调用
print(getResult())
```