42、迷宫问题，考点 or 实现——深度优先搜索 DFS

题目描述

定义一个二维数组 N*M，如 5 × 5 数组下所示：
int maze[5][5] = {
0, 1, 0, 0, 0,
0, 1, 1, 1, 0,
0, 0, 0, 0, 0,
0, 1, 1, 1, 0,
0, 0, 0, 1, 0,
};
它表示一个迷宫，其中的1表示墙壁，0表示可以走的路，只能横着走或竖着走，不能斜着走，要求编程序找出从左上角到右下角的路线。入口点为[0,0],既第一格是可以走的路。
数据范围： 2≤n,m≤10 ， 输入的内容只包含 0≤val≤1。

输入描述

输入两个整数，分别表示二维数组的行数，列数。再输入相应的数组，其中的1表示墙壁，0表示可以走的路。数据保证有唯一解,不考虑有多解的情况，即迷宫只有一条通道。

输出描述

左上角到右下角的最短路径，格式如样例所示。

用例

输入	5 5 0 1 0 0 0 0 1 1 1 0 0 0 0 0 0 0 1 1 1 0 0 0 0 1 0
输出	(0,0) (1,0) (2,0) (2,1) (2,2) (2,3) (2,4) (3,4) (4,4)
说明	无

输入	5 5 0 1 0 0 0 0 1 0 1 0 0 0 0 0 1 0 1 1 1 0 0 0 0 0 0

	0 0 0 0 0
输出	(0,0) (1,0) (2,0) (3,0) (4,0) (4,1) (4,2) (4,3) (4,4)
说明	注意：不能斜着走！！

题目解析

本题可以使用深度优先搜索，从起点开始，将其计入path路径中，接着将起点从”0“改为”2“，表示走过该点了，然后继续dfs其上，下，左，右四个方向的点，直到dfs到的点是终点，即(n-1,m-1)，此时可以return path。

一道很简单的深度优先搜索题。

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, matrix;
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();
    matrix = lines.map((line) => line.split(" ").map(Number));
    getResult(matrix, n, m);
    lines.length = 0;
  }
});

// 上下左右偏移
const offsets = [
```

```
const offsets = [
  [-1, 0],
  [1, 0],
  [0, -1],
  [0, 1],
];

function getResult() {
  const ans = [];
  dfs(0, 0, [], ans);
  ans.push([n - 1, m - 1]); // 由于踏入了终点就返回，因此这里要补终点进来

  for (let [x, y] of ans) {
    console.log(`(${x},${y})`);
  }
}

function dfs(x, y, path, ans) {
  if (x === n - 1 && y === m - 1) {
    ans.push(...path);
    return;
  }

  for (let offset of offsets) {
    const [offsetX, offsetY] = offset;
    const newX = x + offsetX;
    const newY = y + offsetY;

    if (
      newX >= 0 &&
      newX < n &&
      newY >= 0 &&
      newY < m &&
      matrix[newX][newY] == "0"
    ) {
      path.push([x, y]);
      matrix[x][y] = "2"; // 走过的路不再走
      dfs(newX, newY, path, ans);
      path.pop();
    }
  }
}
```

Java算法源码

```java
import java.util.LinkedList;
import java.util.Scanner;

public class Main {
  static int[][] offsets = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};

  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();
      }
    }

    getResult(n, m, matrix);
  }

  public static void getResult(int n, int m, int[][] matrix) {
    LinkedList<String> ans = new LinkedList<>();
    dfs(0, 0, new LinkedList<>(), ans, n, m, matrix);
    ans.add("(" + (n - 1) + "," + (m - 1) + ")");

    for (String an : ans) {
      System.out.println(an);
    }
  }

  public static void dfs(
      int x, int y, LinkedList<String> path, LinkedList<String> ans, int n, int m, int[][] matrix) {
    if (x == n - 1 && y == m - 1) {
      ans.addAll(path);
      return;
    }

    for (int[] offset : offsets) {
      int newX = x + offset[0];
      int newY = y + offset[1];

      if (newX >= 0 && newX < n && newY >= 0 && newY < m && matrix[newX][newY] == 0) {
        path.add("(" + x + "," + y + ")");
        matrix[x][y] = 2;
        dfs(newX, newY, path, ans, n, m, matrix);
        path.removeLast();
      }
    }
  }
}
```

Python算法源码

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

# 上下左右偏移
offsets = ((-1, 0), (1, 0), (0, -1), (0, 1))


# 深搜
def dfs(x, y, path, ans):
    if x == n - 1 and y == m - 1:
        ans.extend(path)
        return

    for offsetX, offsetY in offsets:
        newX = x + offsetX
        newY = y + offsetY

        if 0 <= newX < n and 0 <= newY < m and matrix[newX][newY] == 0:
            path.append([x, y])
            matrix[x][y] = 2  # 走过的路不再走
            dfs(newX, newY, path, ans)
            path.pop()


# 算法入口
def getResult():
    ans = []
    dfs(0, 0, [], ans)
    ans.append([n - 1, m - 1])  # 由于踏入了终点就返回，因此这里要补终点进来

    for x, y in ans:
        print(f"({x},{y})")


# 算法调用
getResult()
```