【华为OD机考 统一考试机试C卷】园区参观路径（C++ Java JavaScript Python C语言）

华为OD机考:统一考试 C卷 + D卷 + B卷 +A卷

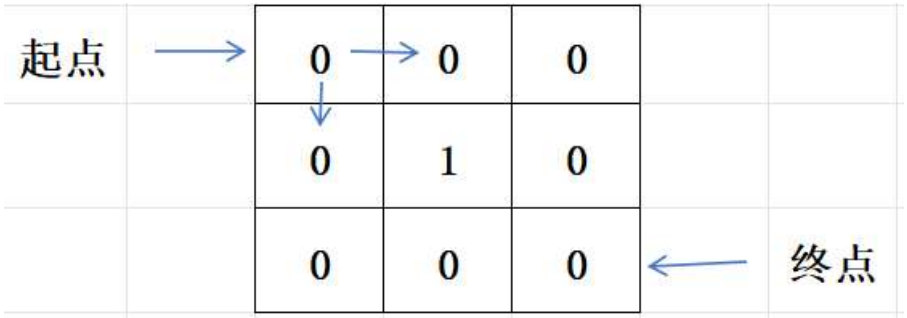
目前在考C卷，经过两个月的收集整理，C卷真题已基本整理完毕
抽到原题的概率为2/3到3/3，也就是最少抽到两道原题。请注意：大家刷完C卷真题，最好要把B卷的真题刷一下，因为C卷的部分真题来自B卷。
另外订阅专栏还可以联系笔者开通在线 OJ 进行刷题，提高刷题效率。
真题目录：华为OD机考机试 真题目录（C卷 + D卷 + B卷 + A卷） + 考点说明
专栏：2023华为OD机试(B卷+C卷+D卷)（C++JavaJSPy）
华为OD面试真题精选：华为OD面试真题精选
在线OJ：点击立即刷题，模拟真实机考环境

题目描述

园区某部门举办了Family Day，邀请员工及其家属参加；

将公司园区视为一个矩形，起始园区设置在左上角，终点园区设置在右下角；

家属参观园区时，只能向右和向下园区前进，求从起始园区到终点园区会有多少条不同的参观路径。



输入描述

第一行为园区的长和宽；

后面每一行表示该园区是否可以参观，0表示可以参观，1表示不能参观

输出描述

输出为不同的路径数量

用例

输入	3 3 0 0 0 0 1 0 0 0 0
输出	2
说明	无

解题思路

这个问题是一个典型的动态规划问题，主要解决的是在一个二维网格中，从左上角到右下角的所有可能路径的数量，其中一些单元格可能是不可达的。

动态规划的基本思想是将一个复杂的问题分解成一系列简单的子问题，并存储子问题的解，以便后续需要时直接使用，而不是重新计算。这种方法通过避免重复计算同一个子问题来节省计算时间。

在这个问题中，我们定义dp[i][j]为从起始点到网格中位置(i,j)的所有可能路径的数量。我们需要找到一个递推关系，用更小的子问题的解来表达dp[i][j]。

这里的递推关系是基于这样一个事实：到达一个特定单元格的路径只能来自其左边的单元格或其上边的单元格。因此，到达该单元格的所有可能路径的数量就是到达其左边单元格的所有可能路径的数量和到达其上边单元格的所有可能路径的数量的和。这就是我们的递推关系。

具体来说，如果我们在第一行或第一列，那么只有一种可能的路径（沿着边缘）。所以，对于第一行和第一列，dp[i][j]就等于其左边单元格或上边单元格的dp值。对于其他位置，dp[i][j] = dp[i-1][j] + dp[i][j-1]。

这就是我们如何使用动态规划来解决这个问题的。我们首先初始化dp数组，然后使用上述递推关系来填充dp数组，最后dp[m-1][n-1]就是我们的答案，也就是从起始点到终点的所有可能路径的数量。

C++

```
#include <iostream>
#include <vector>

using namespace std;

```

```cpp
int main() {
    // 读取园区的长和宽
    int m, n;
    cin >> m >> n;

    // 创建一个二维数组来存储每个园区是否可以参观
    vector<vector<int>> grid(m, vector<int>(n));

    // 使用两层for循环读取每个园区是否可以参观
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            cin >> grid[i][j];
        }
    }

    // 创建一个二维数组来存储从起始园区到每个园区的路径数量
    vector<vector<long long>> dp(m, vector<long long>(n));

    // 使用两层for循环计算从起始园区到每个园区的路径数量
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            // 如果当前园区可以参观
            if (grid[i][j] == 0) {
                if (i == 0 && j == 0) {
                    dp[i][j] = 1;
                } else if (i == 0) {
                    dp[i][j] = dp[i][j - 1];
                } else if (j == 0) {
                    dp[i][j] = dp[i - 1][j];
                } else {
                    dp[i][j] = dp[i - 1][j] + dp[i][j - 1];
                }
            }
        }
    }

    // 输出从起始园区到终点园区的路径数量
    cout << dp[m - 1][n - 1] << endl;
```

```
    return 0;
}
```

Java

```java
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        // 创建一个Scanner对象来读取输入
        Scanner sc = new Scanner(System.in);

        // 读取园区的长和宽
        int m = sc.nextInt();
        int n = sc.nextInt();

        // 创建一个二维数组来存储每个园区是否可以参观
        int[][] grid = new int[m][n];

        // 使用两层for循环读取每个园区是否可以参观
        // 外层for循环遍历每一行
        for (int i = 0; i < m; i++) {
            // 内层for循环遍历每一列
            for (int j = 0; j < n; j++) {
                // 读取并存储当前园区是否可以参观
                grid[i][j] = sc.nextInt();
            }
        }

        // 创建一个二维数组来存储从起始园区到每个园区的路径数量
        long[][] dp = new long[m][n];

        // 使用两层for循环计算从起始园区到每个园区的路径数量
        // 外层for循环遍历每一行
        for (int i = 0; i < m; i++) {
            // 内层for循环遍历每一列
            for (int j = 0; j < n; j++) {
                // 如果当前园区可以参观
                if (grid[i][j] == 0) {
                    // 如果当前园区是起始园区，那么从起始园区到这个园区的路径数量就是1
```

```
                    if (i == 0 && j == 0) {
                        dp[i][j] = 1;
                    }
                    // 如果当前园区在第一行，那么从起始园区到这个园区的路径数量就等于从起始园区到左边的园区的路径数量
                    else if (i == 0) {
                        dp[i][j] = dp[i][j - 1];
                    }
                    // 如果当前园区在第一列，那么从起始园区到这个园区的路径数量就等于从起始园区到上面的园区的路径数量
                    else if (j == 0) {
                        dp[i][j] = dp[i - 1][j];
                    }
                    // 如果当前园区不在第一行和第一列，那么从起始园区到这个园区的路径数量就等于从起始园区到上面的园区的路径数量加上从起始园区到左边的园区的路径数量
                    else {
                        dp[i][j] = dp[i - 1][j] + dp[i][j - 1];
                    }
                }
                // 如果当前园区不能参观，那么从起始园区到这个园区的路径数量就是0
            }
        }

        // 输出从起始园区到终点园区的路径数量
        System.out.println(dp[m - 1][n - 1]);
    }
}
```

javaScript

```
// 引入readline模块
const readline = require('readline');

// 创建readline接口实例
const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout
});

// 创建一个数组来存储输入的数据
let input = [];

// 当接收到用户输入的数据时，将数据存入数组
```

```javascript
rl.on('line', function(line){
    input.push(line.trim().split(' ').map(Number));
});

// 当接收完所有数据后，开始处理
rl.on('close', function(){
    // 获取园区的长和宽
    let m = input[0][0];
    let n = input[0][1];

    // 创建一个二维数组来存储每个园区是否可以参观
    let grid = input.slice(1);

    // 创建一个二维数组来存储从起始园区到每个园区的路径数量
    let dp = Array.from({length: m}, () => Array(n).fill(0));

    // 使用两层for循环计算从起始园区到每个园区的路径数量
    for (let i = 0; i < m; i++) {
        for (let j = 0; j < n; j++) {
            // 如果当前园区可以参观
            if (grid[i][j] == 0) {
                if (i == 0 && j == 0) {
                    dp[i][j] = 1;
                } else if (i == 0) {
                    dp[i][j] = dp[i][j - 1];
                } else if (j == 0) {
                    dp[i][j] = dp[i - 1][j];
                } else {
                    dp[i][j] = dp[i - 1][j] + dp[i][j - 1];
                }
            }
        }
    }

    // 输出从起始园区到终点园区的路径数量
    console.log(dp[m - 1][n - 1]);
});
```

Python

```python
# 使用input()函数读取输入的数据
m, n = map(int, input().split())

# 创建一个二维列表来存储每个园区是否可以参观
grid = [list(map(int, input().split())) for _ in range(m)]

# 创建一个二维列表来存储从起始园区到每个园区的路径数量
dp = [[0]*n for _ in range(m)]

# 使用两层for循环计算从起始园区到每个园区的路径数量
for i in range(m):
    for j in range(n):
        # 如果当前园区可以参观
        if grid[i][j] == 0:
            if i == 0 and j == 0:
                dp[i][j] = 1
            elif i == 0:
                dp[i][j] = dp[i][j - 1]
            elif j == 0:
                dp[i][j] = dp[i - 1][j]
            else:
                dp[i][j] = dp[i - 1][j] + dp[i][j - 1]

# 输出从起始园区到终点园区的路径数量
print(dp[m - 1][n - 1])
```

C语言

```c
#include <stdio.h>

int main() {
    // 读取园区的长和宽
    int m, n;
    scanf("%d %d", &m, &n);

    // 创建一个二维数组来存储每个园区是否可以参观
    int grid[m][n];

    // 使用两层for循环读取每个园区是否可以参观
    for (int i = 0; i < m; i++) {
```

```
        for (int j = 0; j < n; j++) {
            scanf("%d", &grid[i][j]);
        }
    }

    // 创建一个二维数组来存储从起始园区到每个园区的路径数量
    long long dp[m][n];

    // 使用两层for循环计算从起始园区到每个园区的路径数量
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            // 如果当前园区可以参观
            if (grid[i][j] == 0) {
                if (i == 0 && j == 0) {
                    dp[i][j] = 1;
                } else if (i == 0) {
                    dp[i][j] = dp[i][j - 1];
                } else if (j == 0) {
                    dp[i][j] = dp[i - 1][j];
                } else {
                    dp[i][j] = dp[i - 1][j] + dp[i][j - 1];
                }
            } else {
                dp[i][j] = 0;
            }
        }
    }

    // 输出从起始园区到终点园区的路径数量
    printf("%lld\n", dp[m - 1][n - 1]);

    return 0;
}
```

完整用例

用例1

```
3 3
0 0 0
```

```
0 0 0
0 0 0
```

用例2

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

用例3

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

用例4

```
1 1
0
```

用例5

```
1 3
0 0 0
```

用例6

```
3 1
0
0
0
```

用例7

```
5 5
0 0 0 0 0
```

```
0 1 1 1 0
0 1 0 0 0
0 0 0 1 1
0 0 0 0 0
```

用例8

```
6 6
0 0 0 0 0 0
0 1 1 0 1 0
0 1 0 0 0 0
0 0 0 1 1 0
0 1 0 0 0 0
0 0 0 0 0 0
```

用例9

```
3 7
0 0 0 1 0 0 0
0 1 0 0 0 1 0
0 0 0 0 1 0 0
```

用例10

```
8 8
0 0 0 0 0 0 0 0
0 1 1 0 1 1 0 0
0 1 0 0 0 0 1 0
0 0 0 1 1 0 0 0
0 1 0 0 0 1 0 0
0 0 0 1 0 0 0 0
0 1 0 0 1 0 1 0
0 0 0 0 0 0 0 0
```

文章目录


机考真题
华为OD
No No No
CSDN @算法大师