

# 【华为OD机考 统一考试机试C卷】小华地图寻宝（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

小华按照地图去寻宝，地图上被划分成 m 行和 n 列的方格，横纵坐标范围分别是 [0, n-1] 和 [0, m-1]。

在横坐标和纵坐标的数位之和不大于 k 的方格中存在黄金（每个方格中仅存在一克黄金），但横坐标和纵坐标数位之和大于 k 的方格存在危险不可进入。小华从入口 (0,0) 进入，任何时候只能向左，右，上，下四个方向移动一格。

请问小华最多能获得多少克黄金？

## 输入描述

坐标取值范围如下：

- $0 \leq m \leq 50$
- $0 \leq n \leq 50$

k 的取值范围如下：

- $0 \leq k \leq 100$

输入中包含3个字数，分别是m, n, k

## 输出描述

输出小华最多能获得多少克黄金

## 用例

输入	40 40 18
输出	1484

输入	40 40 18
说明	无

输入	5 4 7
输出	20
说明	无

解题思路

- 1. **边界条件:** 在 `dfs` 函数中，首先检查当前坐标是否越界，是否已经被访问过，以及当前坐标的数位和是否大于 `k`。如果满足这些条件中的任何一个，当前方格不可访问，返回 `0`。
- 2. **标记访问:** 创建一个二维布尔数组 `visited` 来跟踪每个方格是否已经被访问过。这是为了防止在搜索过程中重复访问相同的方格。
- 3. **递归搜索:** 从当前方格出发，递归地向四个方向（上、下、左、右）进行搜索。每次递归调用返回时，将返回值累加，表示可以收集到的黄金数量。每个可访问的方格代表一克黄金，所以在每次递归调用的返回值中加 `1`。

C++

```
1  #include <iostream>
2  #include <vector>
3
4  using namespace std;
5  // 定义地图的行数m、列数n以及数位和限制k
6  int m, n, k;
7  vector<vector<bool>> visited;
8  // 计算一个数字的数位和
9  int sumOfDigits(int num) {
10     int sum = 0; // 初始化数位和为0
11     while (num > 0) { // 当数字大于0时循环
12         sum += num % 10; // 取数字的最后一位加到数位和中
13         num /= 10; // 去掉数字的最后一位
14     }
15     return sum; // 返回计算出的数位和
16 }
17 // 深度优先搜索函数
18 int dfs(int x, int y) {
19     // 判断坐标(x, y)是否越界，或者已经被访问过，或者数位和大于k
20     if (x < 0 || y < 0 || x >= m || y >= n || visited[x][y] || sumOfDigits(x) + sumOfDigits(y) > k) {
21         return 0; // 如果是，则返回0，表示这个格子不能访问
22     }
23     visited[x][y] = true; // 标记当前格子为已访问
24     // 递归搜索当前格子的上下左右四个方向，并将结果累加，1表示当前格子的黄金
25     return 1 + dfs(x + 1, y) + dfs(x - 1, y) + dfs(x, y + 1) + dfs(x, y - 1);
26 }
27
28
29
30
```

```

31 int main() {
32     // 使用标准输入读取行数m、列数n和数位和限制k
33     cin >> m >> n >> k;
34     visited.resize(m, vector<bool>(n, false)); // 初始化访问标记数组
35
36     // 从(0,0)开始进行深度优先搜索, 并打印能够收集到的最大黄金数量
37     cout << dfs(0, 0) << endl;
38     return 0;
39 }

```

## Java

```

1  import java.util.Scanner;
2
3  public class Main {
4      // 定义地图的行数m、列数n以及数位和限制k
5      private static int m, n, k;
6      // 定义一个二维数组visited, 用于标记格子是否已经被访问过
7      private static boolean[][] visited;
8
9      public static void main(String[] args) {
10         // 使用Scanner类读取输入
11         Scanner scanner = new Scanner(System.in);
12         m = scanner.nextInt(); // 读取行数m
13         n = scanner.nextInt(); // 读取列数n
14         k = scanner.nextInt(); // 读取数位和限制k
15         visited = new boolean[m][n]; // 初始化访问标记数组
16
17         // 从(0,0)开始进行深度优先搜索, 并打印能够收集到的最大黄金数量
18         System.out.println(dfs(0, 0));
19     }
20
21     // 深度优先搜索函数
22     private static int dfs(int x, int y) {
23         // 判断坐标(x, y)是否越界, 或者已经被访问过, 或者数位和大于k
24         if (x < 0 || y < 0 || x >= m || y >= n || visited[x][y] || sumOfDigits(x) + sumOfDigits(y) > k) {
25             return 0; // 如果是, 则返回0, 表示这个格子不能访问
26         }
27         visited[x][y] = true; // 标记当前格子为已访问
28         // 递归搜索当前格子的上下左右四个方向, 并将结果累加, 1表示当前格子的黄金
29         return 1 + dfs(x + 1, y) + dfs(x - 1, y) + dfs(x, y + 1) + dfs(x, y - 1);
30     }
31
32     // 计算一个数字的数位和
33     private static int sumOfDigits(int num) {
34         int sum = 0; // 初始化数位和为0
35         while (num > 0) { // 当数字大于0时循环
36             sum += num % 10; // 取数字的最后一位加到数位和中
37             num /= 10; // 去掉数字的最后一位
38         }
39         return sum; // 返回计算出的数位和
40     }

```

```
40 |     }
41 | }
```

## JavaScript

```
1 | const readline = require('readline');
2 |
3 | const rl = readline.createInterface({
4 |   input: process.stdin,
5 |   output: process.stdout
6 | });
7 |
8 | // 读取输入的行数m、列数n和数位和限制k
9 | rl.on('line', (input) => {
10 |   const [m, n, k] = input.split(' ').map(Number);
11 |   const visited = Array.from({ length: m }, () => Array(n).fill(false));
12 |
13 |   console.log(dfs(0, 0, m, n, k, visited));
14 |   rl.close();
15 | });
16 |
17 | // 深度优先搜索函数
18 | function dfs(x, y, m, n, k, visited) {
19 |   // 判断坐标(x, y)是否越界, 或者已经被访问过, 或者数位和大于k
20 |   if (x < 0 || y < 0 || x >= m || y >= n || visited[x][y] || sumOfDigits(x) + sumOfDigits(y) > k) {
21 |     return 0; // 如果是, 则返回0, 表示这个格子不能访问
22 |   }
23 |   visited[x][y] = true; // 标记当前格子为已访问
24 |   // 递归搜索当前格子的上下左右四个方向, 并将结果累加, 1表示当前格子的黄金
25 |   return 1 + dfs(x + 1, y, m, n, k, visited) + dfs(x - 1, y, m, n, k, visited) +
26 |     dfs(x, y + 1, m, n, k, visited) + dfs(x, y - 1, m, n, k, visited);
27 | }
28 |
29 | // 计算一个数字的数位和
30 | function sumOfDigits(num) {
31 |   let sum = 0; // 初始化数位和为0
32 |   while (num > 0) { // 当数字大于0时循环
33 |     sum += num % 10; // 取数字的最后一位加到数位和中
34 |     num = Math.floor(num / 10); // 去掉数字的最后一位
35 |   }
36 |   return sum; // 返回计算出的数位和
37 | }
```

## Python

```
1 | import sys
2 |
3 | # 设置递归的最大深度为10000
4 | 在Python中, 默认的最大递归深度是比较小的 (通常是1000), 这意味着如果一个递归函数调用自身超过1000次, Python解释器会抛出一个`RecursionError`错误。这是为了防止无限递归导致的栈溢出, 从而可能会破坏程序或操作系统的稳定性。
5 |
6 |
```

```

7 sys.setrecursionlimit(10000)
8
9 # 定义一个函数，用于计算一个整数的数位之和
10 def sum_of_digits(num):
11     sum = 0 # 初始化数位和为0
12     while num > 0: # 当数字大于0时循环
13         sum += num % 10 # 将数字的最后一位加到数位和中
14         num //= 10 # 使用整除去掉数字的最后一位
15     return sum # 返回计算出的数位和
16
17 # 定义深度优先搜索函数
18 def dfs(x, y, m, n, k, visited):
19     # 判断坐标(x, y)是否越界，或者已经被访问过，或者(x, y)的数位和大于k
20     if x < 0 or y < 0 or x >= m or y >= n or visited[x][y] or sum_of_digits(x) + sum_of_digits(y) > k:
21         return 0 # 如果是，则返回0，表示这个格子不能访问
22     visited[x][y] = True # 标记当前格子为已访问
23     # 递归搜索当前格子的上下左右四个方向，并将结果累加，1表示当前格子可以访问
24     return 1 + dfs(x + 1, y, m, n, k, visited) + dfs(x - 1, y, m, n, k, visited) + dfs(x, y + 1, m, n, k, visited) + dfs(x, y - 1, m, n, k, visited)
25
26 # 从标准输入读取行数m、列数n和数位和限制k
27 m, n, k = map(int, input().split())
28 # 初始化访问标记数组，初始值为False，表示没有格子被访问过
29 visited = [[False for _ in range(n)] for _ in range(m)]
30
31 # 从(0,0)开始进行深度优先搜索，并打印能够访问的格子数量
32 print(dfs(0, 0, m, n, k, visited))

```

## C语言

```

1 #include <stdio.h>
2 #include <stdbool.h>
3
4 #define MAX_M 50
5 #define MAX_N 50
6
7 // 全局变量定义
8 int m, n, k;
9 bool visited[MAX_M][MAX_N] = {false}; // 访问标记数组
10
11 // 计算一个数字的数位和
12 int sumOfDigits(int num) {
13     int sum = 0; // 初始化数位和为0
14     while (num > 0) { // 当数字大于0时循环
15         sum += num % 10; // 取数字的最后一位加到数位和中
16         num /= 10; // 去掉数字的最后一位
17     }
18     return sum; // 返回计算出的数位和
19 }
20
21 // 深度优先搜索函数
22 int dfs(int x, int y) {
23     // 判断坐标(x, y)是否越界，或者已经被访问过，或者(x, y)的数位和大于k
24     if (x < 0 || y < 0 || x >= m || y >= n || visited[x][y] || sumOfDigits(x) + sumOfDigits(y) > k)
25         return 0; // 如果是，则返回0，表示这个格子不能访问
26     visited[x][y] = true; // 标记当前格子为已访问
27     // 递归搜索当前格子的上下左右四个方向，并将结果累加，1表示当前格子可以访问
28     return 1 + dfs(x + 1, y) + dfs(x - 1, y) + dfs(x, y + 1) + dfs(x, y - 1);
29 }
30
31 // 从(0,0)开始进行深度优先搜索，并打印能够访问的格子数量
32 int main() {
33     int m, n, k;
34     scanf("%d %d %d", &m, &n, &k);
35     printf("%d", dfs(0, 0));
36     return 0;
37 }

```

```
23 // 判断坐标(x, y)是否越界, 或者已经被访问过, 或者数位和大于k
24 if (x < 0 || y < 0 || x >= m || y >= n || visited[x][y] || sumOfDigits(x) + sumOfDigits(y) > k) {
25     return 0; // 如果是, 则返回0, 表示这个格子不能访问
26 }
27 visited[x][y] = true; // 标记当前格子为已访问
28 // 递归搜索当前格子的上下左右四个方向, 并将结果累加, 1表示当前格子的黄金
29 return 1 + dfs(x + 1, y) + dfs(x - 1, y) + dfs(x, y + 1) + dfs(x, y - 1);
30 }
31
32 int main() {
33     // 使用标准输入读取行数m、列数n和数位和限制k
34     scanf("%d %d %d", &m, &n, &k);
35
36     // 从(0,0)开始进行深度优先搜索, 并打印能够收集到的最大黄金数量
37     printf("%d\n", dfs(0, 0));
38     return 0;
39 }
```

## 文章目录

[华为OD机考:统一考试 C卷 + D卷 + B卷 +A卷](#)

[题目描述](#)

[输入描述](#)

[输出描述](#)

[用例](#)

[解题思路](#)

[C++](#)

[Java](#)

[javaScript](#)

[Python](#)

[C语言](#)

# 机考真题 华为OD



CSDN @算法大师