

【华为OD机考 统一考试机试C卷】亲子游戏（C++ Java JavaScript Python C语言）

华为OD机考:统一考试 C卷 + D卷 + B卷 +A卷

2023年11月份，华为官方已经将 华为OD机考：OD统一考试（A卷 / B卷）切换到 OD统一考试（C卷）和 OD统一考试（D卷）。根据考友反馈：目前抽到的试卷为B卷或C卷/D卷，其中C卷居多，按照之前的经验C卷D卷部分考题会复用A卷/B卷题，博主正积极从考过的同学收集C卷和D卷真题，可以查看下面的真题目录。

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](#) 华为OD机考B卷C卷华为OD机考华为OD机考B卷华为OD机试B卷华为OD机试C卷华为OD机考C卷华为OD机考D卷题目华为OD机考C卷/D卷答案华为OD机考C卷/D卷解析华为OD机考C卷和D卷真题华为OD机考C卷和D卷题解

题目描述

宝宝和妈妈参加亲子游戏，在一个二维矩阵（ $N \times N$ ）的格子地图上，宝宝和妈妈抽签决定各自的位置，地图上每个格子有不同的糖果数量，部分格子有障碍物。

游戏规则是妈妈必须在最短的时间（每个单位时间只能走一步）到达宝宝的位置，路上的所有糖果都可以拿走，不能走障碍物的格子，只能上下左右走。

请问妈妈在最短到达宝宝位置的时间内最多拿到多少糖果（优先考虑最短时间到达的情况下尽可能多拿糖果）。

输入描述

第一行输入为 N ， N 表示二维矩阵的大小

之后 N 行，每行有 N 个值，表格矩阵每个位置的值，其中：

- -3: 妈妈
- -2: 宝宝
- -1: 障碍
- ≥ 0 : 糖果数（0表示没有糖果，但是可以走）

输出描述

输出妈妈在最短到达宝宝位置的时间内最多拿到多少糖果，行末无多余空格

用例

输入

1	4
2	3 2 1 -3
3	1 -1 1 1
4	1 1 -1 2
5	-2 1 2 3

输出

1	9
---	---

说明

此地图有两条最短路径可到达宝宝位置，绿色线和黄色线都是最短路径6步，但黄色拿到的糖果更多，9个。

输入

1	4
2	3 2 1 -3
3	-1 -1 1 1
4	1 1 -1 2
5	-2 1 -1 3

输出

1	-1
---	----

说明

此地图妈妈无法到达宝宝位置

解题思路

1. 初始化:

- 初始化 `grid` 矩阵存储输入的网格数据。
- 初始化 `visited` 三维数组来记录每个位置的最短步数和糖果数量, 初始值为 `-1`。

2. 读取网格信息:

- 通过双层循环读取网格中的每个值。
- 如果遇到起点 (值为 `-3`) , 则创建起点 `Node` 并更新 `visited` 数组中起点的步数和糖果数为 `0`。

3. 广度优先搜索 (BFS) :

- 将起点 `Node` 加入队列 `queue` 。
- 初始化最大糖果数 `maxCandies` 为 `0` , 用 `flag` 标记是否到达终点。
- 当队列不为空时, 循环执行以下步骤:
 - 从队列中取出一个节点 `current` 。
 - 如果当前节点是终点 (值为 `-2`) , 则更新 `maxCandies` 并继续循环。
 - 遍历四个可能的移动方向。
 - 对于每个方向, 计算新位置的坐标 `nx` 和 `ny` 。
 - 检查新位置是否在网格内、不是障碍物 (值不为 `-1`) 。
 - 计算到达新位置的糖果数 `newCandies` 和步数 `newSteps` 。
 - 如果新位置未访问过, 或者可以以更少的步数到达, 或者步数相同但糖果数更多, 则更新 `visited` 数组并将新位置的 `Node` 加入队列。

4. 输出结果:

- 如果 `flag` 为 `0` , 说明没有到达终点, 将 `maxCandies` 设置为 `-1` 。
- 输出最大糖果数 `maxCandies` 。如果没有到达终点, 则输出 `-1` 。

5. 辅助类 `Node` :

- 定义了一个 `Node` 类来表示 BFS 中的每个状态, 包含位置坐标 `(x, y)` 、糖果数 `candies` 和步数 `steps` 。

C++

```
1  #include <iostream>
2  #include <vector>
3  #include <queue>
4  #include <algorithm>
5  using namespace std;
6
7  // 定义四个方向移动的坐标变化 (上、右、下、左)
8  const int dx[4] = {-1, 0, 1, 0};
9  const int dy[4] = {0, 1, 0, -1};
10
11 // 节点类, 用于表示 BFS 中的每个状态
12 struct Node {
13     int x, y, candies, steps;
14     // 构造函数, 初始化节点位置、糖果数和步数
15     Node(int _x, int _y, int _candies, int _steps) : x(_x), y(_y), candies(_candies), steps(_steps) {}
16 };
17
18 int main() {
19     int N; // 矩阵的大小
20     cin >> N; // 输入矩阵的大小
21     vector<vector<int>>> grid(N, vector<int>(N)); // 创建一个二维矩阵存储输入的网格
22     // 创建一个三维数组来标记每个位置的访问状态, 包括步数和糖果数
23     vector<vector<vector<int>>> visited(N, vector<vector<int>>(N, vector<int>(2, -1)));
24     queue<Node> queue; // 创建一个队列用于 BFS
25     Node start(0, 0, 0, 0); // 初始化起点
26     int maxCandies = 0; // 最大糖果数初始化为 0
27     int flag = 0;
28     // 读取网格数据, 并找到起点
29     for (int i = 0; i < N; ++i) {
30         for (int j = 0; j < N; ++j) {
31             cin >> grid[i][j]; // 输入网格中的每个值
32             if (grid[i][j] == -3) { // 如果是起点
33                 start = Node(i, j, 0, 0); // 更新起点信息
34                 visited[i][j][0] = 0; // 标记起点的步数为 0
35                 visited[i][j][1] = 0; // 标记起点的糖果数为 0
36             }
37         }
38     }
39
40     queue.push(start); // 将起点加入队列
41 }
```

```
41  
42  
43 // BFS 搜索  
44 while (!queue.empty()) {  
45     Node current = queue.front(); // 取出队列前端的节点  
46     queue.pop(); // 弹出队列前端的节点  
47     if (grid[current.x][current.y] == -2) { // 如果到达终点  
48         flag = 1;  
49         maxCandies = max(maxCandies, current.candies); // 更新最大糖果数  
50         continue; // 继续搜索其他路径  
51     }  
52  
53     // 遍历四个方向  
54     for (int i = 0; i < 4; ++i) {  
55         int nx = current.x + dx[i]; // 计算新的 x 坐标  
56         int ny = current.y + dy[i]; // 计算新的 y 坐标  
57  
58         // 检查新坐标是否在网格内以及是否可走  
59         if (nx >= 0 && nx < N && ny >= 0 && ny < N && grid[nx][ny] != -1) {  
60             int newCandies = current.candies + max(grid[nx][ny], 0); // 计算新的糖果数  
61             int newSteps = current.steps + 1; // 计算新的步数  
62             // 如果新坐标未访问过或者有更优的路径 (步数更少或糖果数更多)  
63             if (visited[nx][ny][0] == -1 || visited[nx][ny][0] > newSteps ||  
64                 (visited[nx][ny][0] == newSteps && visited[nx][ny][1] < newCandies)) {  
65                 queue.push(Node(nx, ny, newCandies, newSteps)); // 将新节点加入队列  
66                 visited[nx][ny][0] = newSteps; // 更新访问状态的步数  
67                 visited[nx][ny][1] = newCandies; // 更新访问状态的糖果数  
68             }  
69         }  
70     }  
71 }  
72 if(flag == 0){  
73     maxCandies = -1;  
74 }  
75  
76 // 输出最大糖果数, 如果没有到达终点则输出 -1  
77 cout << (maxCandies >= 0 ? maxCandies : -1) << endl;  
78 return 0;  
}
```

Java

```
1 import java.util.*;
2
3 // 主类
4 public class Main {
5     // 定义四个方向移动的坐标变化 (上、右、下、左)
6     private static final int[] dx = {-1, 0, 1, 0};
7     private static final int[] dy = {0, 1, 0, -1};
8
9     // 主函数
10    public static void main(String[] args) {
11        // 使用 Scanner 读取输入数据
12        Scanner scanner = new Scanner(System.in);
13        // 读取矩阵的大小
14        int N = scanner.nextInt();
15        // 初始化矩阵
16        int[][] grid = new int[N][N];
17        // 初始化访问数组, 记录到达每个位置的最短步数和糖果数量
18        int[][][] visited = new int[N][N][2]; // [x][y][0] 代表步数, [x][y][1] 代表糖果数
19        // 将访问数组初始化为 -1
20        for (int[][] layer : visited) {
21            for (int[] cell : layer) {
22                Arrays.fill(cell, -1);
23            }
24        }
25        // 初始化队列, 用于 BFS 搜索
26        Queue<Node> queue = new LinkedList<>();
27        // 初始化起点
28        Node start = null;
29
30        // 读取矩阵信息, 并找到起点位置
31        for (int i = 0; i < N; i++) {
32            for (int j = 0; j < N; j++) {
33                grid[i][j] = scanner.nextInt();
34                if (grid[i][j] == -3) { // 如果是起点
35                    start = new Node(i, j, 0, 0); // 创建起点节点
36                    visited[i][j][0] = 0; // 起点的步数为 0
37                    visited[i][j][1] = 0; // 起点的糖果数为 0
38                }
39            }
40        }
41    }
42}
```

```
41 // 关闭 Scanner
42 scanner.close();
43
44 // 将起点加入队列
45 queue.add(start);
46 // 初始化最大糖果数
47 int maxCandies = 0;
48 int flag = 0;
49 // BFS 搜索
50 while (!queue.isEmpty()) {
51     Node current = queue.poll();
52     // 如果到达终点, 更新最大糖果数
53     if (grid[current.x][current.y] == -2) {
54         flag = 1;
55         maxCandies = Math.max(maxCandies, current.candies);
56         continue;
57     }
58
59     // 遍历四个方向
60     for (int i = 0; i < 4; i++) {
61         int nx = current.x + dx[i];
62         int ny = current.y + dy[i];
63
64         // 检查新位置是否有效
65         if (nx >= 0 && nx < N && ny >= 0 && ny < N && grid[nx][ny] != -1) {
66             // 计算新位置的糖果数和步数
67             int newCandies = current.candies + Math.max(grid[nx][ny], 0);
68             int newSteps = current.steps + 1;
69             // 如果新位置未访问过, 或者可以以更少的步数到达, 或者步数相同但糖果数更多, 则更新信息并加入队列
70             if (visited[nx][ny][0] == -1 || visited[nx][ny][0] > newSteps ||
71                 (visited[nx][ny][0] == newSteps && visited[nx][ny][1] < newCandies)) {
72                 queue.add(new Node(nx, ny, newCandies, newSteps));
73                 visited[nx][ny][0] = newSteps;
74                 visited[nx][ny][1] = newCandies;
75             }
76         }
77     }
78 }
79 if(flag == 0){
80     maxCandies = -1;
81 }
```



```
82     }
83     // 输出最大糖果数, 如果没有到达终点则输出 -1
84     System.out.println(maxCandies >= 0 ? maxCandies : -1);
85 }
86
87 // 节点类, 用于表示 BFS 中的每个状态
88 static class Node {
89     int x, y, candies, steps;
90
91     // 节点构造函数
92     public Node(int x, int y, int candies, int steps) {
93         this.x = x;
94         this.y = y;
95         this.candies = candies;
96         this.steps = steps;
97     }
98 }
99 }
```

javaScript

```
1  const readline = require('readline');
2  const rl = readline.createInterface({
3      input: process.stdin,
4      output: process.stdout
5  });
6
7  // 定义四个方向移动的坐标变化 (上、右、下、左)
8  const dx = [-1, 0, 1, 0];
9  const dy = [0, 1, 0, -1];
10
11 // 节点类, 用于表示 BFS 中的每个状态
12 class Node {
13     constructor(x, y, candies, steps) {
14         this.x = x; // 当前节点的 x 坐标
15         this.y = y; // 当前节点的 y 坐标
16         this.candies = candies; // 从起点到当前节点收集到的糖果数
17         this.steps = steps; // 从起点到当前节点的步数
18     }
19 }
```

```
20
21 // 当读取到一行输入时触发
22 rl.on('line', (line) => {
23   const parts = line.split(' ').map(x => parseInt(x)); // 将输入的行分割成数组并转换为整数
24   if (parts.length === 1) {
25     // 如果只有一个数字, 则表示是矩阵的大小
26     const N = parts[0]; // 矩阵的大小
27     const grid = []; // 存储矩阵的二维数组
28     // 创建一个三维数组来标记每个位置的访问状态, 包括步数和糖果数
29     const visited = Array.from({ length: N }, () => Array.from({ length: N }, () => [-1, -1]));
30     let linesRead = 0; // 已读取的行数
31     let start = null; // 起点
32     let maxCandies = 0; // 最大糖果数
33
34     // 当读取到 N 行后, 开始处理矩阵
35     rl.on('line', (line) => {
36       grid.push(line.split(' ').map(x => parseInt(x))); // 将每行的数据添加到矩阵中
37       linesRead++; // 增加已读取的行数
38       if (linesRead === N) {
39         // 如果已读取完矩阵的所有行
40         // 初始化起点和队列
41         const queue = [];
42         for (let i = 0; i < N; i++) {
43           for (let j = 0; j < N; j++) {
44             if (grid[i][j] === -3) {
45               start = new Node(i, j, 0, 0); // 找到起点并创建起点节点
46               visited[i][j] = [0, 0]; // 标记起点为已访问
47               break;
48             }
49           }
50         }
51         queue.push(start); // 将起点加入队列
52         let flag = 0;
53         // BFS 搜索
54         while (queue.length > 0) {
55           const current = queue.shift(); // 取出队列的第一个节点
56           if (grid[current.x][current.y] === -2) {
57             // 如果当前节点是终点
58             flag = 1;
59             maxCandies = Math.max(maxCandies, current.candies); // 更新最大糖果数
60           }
61         }
62       }
63     });
64   }
65 }
```

```
61         continue;
62     }
63
64     // 遍历四个方向
65     for (let i = 0; i < 4; i++) {
66         const nx = current.x + dx[i]; // 计算新的 x 坐标
67         const ny = current.y + dy[i]; // 计算新的 y 坐标
68
69         // 检查新坐标是否在矩阵内以及是否可走
70         if (nx >= 0 && nx < N && ny >= 0 && ny < N && grid[nx][ny] !== -1) {
71             const newCandies = current.candies + Math.max(grid[nx][ny], 0); // 计算新的糖果数
72             const newSteps = current.steps + 1; // 计算新的步数
73             // 如果新坐标未访问过或者有更优的路径 (步数更少或糖果数更多)
74             if (visited[nx][ny][0] === -1 || visited[nx][ny][0] > newSteps ||
75                 (visited[nx][ny][0] === newSteps && visited[nx][ny][1] < newCandies)) {
76                 queue.push(new Node(nx, ny, newCandies, newSteps)); // 将新节点加入队列
77                 visited[nx][ny] = [newSteps, newCandies]; // 更新访问状态
78             }
79         }
80     }
81 }
82 if(flag === 0){
83     maxCandies = -1;
84 }
85 // 输出最大糖果数, 如果没有到达终点则输出 -1
86 console.log(maxCandies >= 0 ? maxCandies : -1);
87 rl.close(); // 关闭 readline 接口
88 }
89 });
90 }
```

Python

```
1 from collections import deque
2
3 # 定义四个方向移动的坐标变化 (上、右、下、左)
4 dx = [-1, 0, 1, 0]
5 dy = [0, 1, 0, -1]
6
7
```

```
1  # 节点类, 用于表示 BFS 中的每个状态
2
3  class Node:
4      def __init__(self, x, y, candies, steps):
5          self.x = x # 当前节点的 x 坐标
6          self.y = y # 当前节点的 y 坐标
7          self.candies = candies # 从起点到当前节点收集到的糖果数
8          self.steps = steps # 从起点到当前节点的步数
9
10 # 主函数
11 if __name__ == "__main__":
12     N = int(input()) # 读取矩阵的大小
13     grid = [] # 存储矩阵的二维数组
14     # 创建一个三维数组来标记每个位置的访问状态, 包括步数和糖果数
15     visited = [[[-1, -1] for _ in range(N)] for _ in range(N)]
16     max_candies = 0 # 最大糖果数
17
18     # 读取矩阵数据并找到起点
19     for i in range(N):
20         grid.append(list(map(int, input().split()))) # 将每行的数据添加到矩阵中
21         for j in range(N):
22             if grid[i][j] == -3: # 如果当前位置是起点
23                 start = Node(i, j, 0, 0) # 创建起点节点
24                 visited[i][j] = [0, 0] # 标记起点为已访问
25
26     # 使用双端队列来进行 BFS 搜索
27     queue = deque([start])
28     flag = 0
29     # BFS 搜索
30     while queue:
31         current = queue.popleft() # 取出队列的第一个节点
32         if grid[current.x][current.y] == -2: # 如果当前节点是终点
33             flag = 1
34             max_candies = max(max_candies, current.candies) # 更新最大糖果数
35             continue
36
37     # 遍历四个方向
38     for i in range(4):
39         nx = current.x + dx[i] # 计算新的 x 坐标
40         ny = current.y + dy[i] # 计算新的 y 坐标
41
42
43
44
45
46
47
48
```

```

48     # 检查新坐标是否在矩阵内以及是否可走
49     if 0 <= nx < N and 0 <= ny < N and grid[nx][ny] != -1:
50         new_candies = current.candies + max(grid[nx][ny], 0) # 计算新的糖果数
51         new_steps = current.steps + 1 # 计算新的步数
52         # 如果新坐标未访问过或者有更优的路径 (步数更少或糖果数更多)
53         if visited[nx][ny][0] == -1 or visited[nx][ny][0] > new_steps or \
54             (visited[nx][ny][0] == new_steps and visited[nx][ny][1] < new_candies):
55             queue.append(Node(nx, ny, new_candies, new_steps)) # 将新节点加入队列
56             visited[nx][ny] = [new_steps, new_candies] # 更新访问状态
57     if flag == 0:
58         max_candies = -1
59     # 输出最大糖果数, 如果没有到达终点则输出 -1
    print(max_candies if max_candies >= 0 else -1)

```

C语言

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  #define MAX_N 50
6
7  // 定义四个方向移动的坐标变化 (上、右、下、左)
8  const int dx[4] = {-1, 0, 1, 0};
9  const int dy[4] = {0, 1, 0, -1};
10
11 // 节点结构体, 用于表示 BFS 中的每个状态
12 typedef struct {
13     int x, y, candies, steps;
14 } Node;
15
16 // 队列结构体, 用于 BFS 搜索
17 typedef struct {
18     Node nodes[MAX_N * MAX_N];
19     int front, rear;
20 } Queue;
21
22 // 初始化队列
23 void initQueue(Queue *q) {
24     q->front = q->rear = 0;
25 }

```

```
25 }
26
27 // 队列是否为空
28 int isEmpty(Queue *q) {
29     return q->front == q->rear;
30 }
31
32 // 入队操作
33 void push(Queue *q, Node node) {
34     q->nodes[q->rear++] = node;
35 }
36
37 // 出队操作
38 Node pop(Queue *q) {
39     return q->nodes[q->front++];
40 }
41
42 // 主函数
43 int main() {
44     int N; // 矩阵的大小
45     scanf("%d", &N); // 输入矩阵的大小
46     int grid[MAX_N][MAX_N]; // 创建一个二维矩阵存储输入的网格
47     int visited[MAX_N][MAX_N][2]; // 创建一个三维数组来标记每个位置的访问状态, 包括步数和糖果数
48     memset(visited, -1, sizeof(visited)); // 初始化访问状态数组为 -1
49     Queue queue; // 创建一个队列用于 BFS
50     initQueue(&queue); // 初始化队列
51     Node start = {0, 0, 0, 0}; // 初始化起点
52     int maxCandies = 0; // 最大糖果数初始化为 0
53     int flag = 0; // 标记是否到达终点
54
55     // 读取网格数据, 并找到起点
56     for (int i = 0; i < N; ++i) {
57         for (int j = 0; j < N; ++j) {
58             scanf("%d", &grid[i][j]); // 输入网格中的每个值
59             if (grid[i][j] == -3) { // 如果是起点
60                 start.x = i;
61                 start.y = j;
62                 visited[i][j][0] = 0; // 标记起点的步数为 0
63                 visited[i][j][1] = 0; // 标记起点的糖果数为 0
64             }
65         }
66     }
```

```
66     }
67 }
68
69 push(&queue, start); // 将起点加入队列
70
71 // BFS 搜索
72 while (!isEmpty(&queue)) {
73     Node current = pop(&queue); // 取出队列前端的节点
74     if (grid[current.x][current.y] == -2) { // 如果到达终点
75         flag = 1;
76         if (current.candies > maxCandies) {
77             maxCandies = current.candies; // 更新最大糖果数
78         }
79         continue; // 继续搜索其他路径
80     }
81
82     // 遍历四个方向
83     for (int i = 0; i < 4; ++i) {
84         int nx = current.x + dx[i]; // 计算新的 x 坐标
85         int ny = current.y + dy[i]; // 计算新的 y 坐标
86
87         // 检查新坐标是否在网格内以及是否可走
88         if (nx >= 0 && nx < N && ny >= 0 && ny < N && grid[nx][ny] != -1) {
89             int newCandies = current.candies + (grid[nx][ny] > 0 ? grid[nx][ny] : 0); // 计算新的糖果数
90             int newSteps = current.steps + 1; // 计算新的步数
91             // 如果新坐标未访问过或者有更优的路径 (步数更少或糖果数更多)
92             if (visited[nx][ny][0] == -1 || visited[nx][ny][0] > newSteps ||
93                 (visited[nx][ny][0] == newSteps && visited[nx][ny][1] < newCandies)) {
94                 Node newNode = {nx, ny, newCandies, newSteps};
95                 push(&queue, newNode); // 将新节点加入队列
96                 visited[nx][ny][0] = newSteps; // 更新访问状态的步数
97                 visited[nx][ny][1] = newCandies; // 更新访问状态的糖果数
98             }
99         }
100     }
101 }
102 if (flag == 0) {
103     maxCandies = -1; // 如果没有到达终点, 则最大糖果数为 -1
104 }
105
106
```

```
107 | // 输出最大糖果数, 如果没有到达终点则输出 -1
108 | printf("%d\n", maxCandies);
    | return 0;
    | }
```

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

