

【华为OD机考 统一考试机试C卷】寻找最优的路测线路 (C++ Java JavaScript Python)

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

评估一个网络的信号质量，其中一个做法是将网络划分为栅格，然后对每个栅格的信号质量计算。

路测的时候，希望选择一条信号最好的路线（彼此相连的栅格集合）进行演示。

现给出 R 行 C 列的整数数组 Cov，每个单元格的数值 S 即为该栅格的信号质量（已归一化，无单位，值越大信号越好）。

要求从 [0, 0] 到 [R-1, C-1]设计一条最优路测路线。返回该路线得分。

规则：

- 路测路线可以上下左右四个方向，不能对角
- 路线的评分是以路线上信号最差的栅格为准的，例如路径 8→4→5→9 的值为4，该线路评分为4。线路最优表示该条线路的评分最高。

输入描述

一行表示栅格的行数 R

第二行表示栅格的列数 C

第三行开始，每一行表示栅格地图一行的信号值，如5 4 5

输出描述

最优路线的得分

备注

- $1 \leq R, C \leq 20$
- $0 \leq S \leq 65535$

用例1

输入

1	3
2	3
3	5 4 5
4	1 2 6
5	7 4 6

输出

1	4
---	---

说明

路线为：5→4→5→6→6

用例2

输入

1	6
2	5
3	3 4 6 3 4
4	0 2 1 1 7
5	8 8 3 2 7
6	3 2 4 9 8
7	

```
8 | 4 1 2 0 0
   | 4 6 5 4 3
```

输出

```
1 | 3
```

说明

路线为: 3→4→6→3→4→7→7→8→9→4→3→8→8→3→4→4→6→5→4→3

解题思路

使用 广度优先搜索 (BFS) + 二分查找。

1. 广度优先搜索 (BFS):

- `bfs` 函数实现了广度优先搜索算法。它的目的是检查是否存在一条从网格的左上角（起点）到右下角（终点）的路径，且路径上每个单元的信号强度都至少为 `minSignal`。
- 它首先检查起点和终点的信号强度，如果任何一个小于 `minSignal`，则返回false。
- 使用一个队列来存储待访问的单元格，从起点开始搜索。
- 对于队列中的每个元素，它会检查从该点出发可以到达的四个方向（上下左右）。如果相邻单元的信号强度满足要求且未被访问过，则将其添加到队列中。
- 这个过程会一直进行，直到找到一条到达终点的有效路径，或者队列变空（没有路径满足条件）。

2. 二分查找:

- `binarySearch` 函数使用二分查找来确定满足条件的最大 `minSignal` 值。
- 它在最小可能信号强度（`minSignal`）和最大可能信号强度（`maxSignal`）之间进行搜索。
- 在每次迭代中，它会计算当前范围的中间值 `mid`，然后使用BFS检查是否存在一条满足 `mid` 作为最小信号强度的路径。
- 如果存在这样的路径，它会尝试更高的信号强度；如果不存在，则降低信号强度。
- 通过不断调整搜索范围，二分查找最终确定了可以找到有效路径的最大 `minSignal` 值。

C++

```
1  #include <iostream>
2  #include <vector>
3  #include <queue>
4  #include <climits>
5
6  using namespace std;
7
8  // 定义一个类表示网格中的一个单元格
9  class Cell {
10 public:
11     int row, col;
12     Cell(int r, int c) : row(r), col(c) {} // 构造函数, 初始化行和列
13 };
14
15 // 使用广度优先搜索 (BFS) 检查是否存在一条从起点到终点的路径, 路径上所有单元格的信号质量都不低于minSignal
16 bool bfs(const vector<vector<int>>& Cov, int minSignal) {
17     int R = Cov.size(), C = Cov[0].size();
18     // 如果起点或终点的信号质量低于minSignal, 直接返回false
19     if (Cov[0][0] < minSignal || Cov[R - 1][C - 1] < minSignal) {
20         return false;
21     }
22
23     // visited数组用于记录哪些单元格已经被访问过, 避免重复访问
24     vector<vector<bool>> visited(R, vector<bool>(C, false));
25     queue<Cell> queue;
26     queue.push(Cell(0, 0));
27     visited[0][0] = true;
28
29     // dr和dc数组用于表示从当前单元格向四个方向 (上下左右) 移动的行和列的变化量
30     int dr[4] = {1, -1, 0, 0};
31     int dc[4] = {0, 0, 1, -1};
32
33     while (!queue.empty()) {
34         Cell cell = queue.front();
35         queue.pop();
36         // 如果到达终点, 返回true
37         if (cell.row == R - 1 && cell.col == C - 1) {
38             return true;
39         }
40     }
41 }
```

```
41 // 否则, 尝试向四个方向移动
42 for (int i = 0; i < 4; i++) {
43     int nr = cell.row + dr[i];
44     int nc = cell.col + dc[i];
45
46     // 如果新的单元格在网格内, 且没有被访问过, 且信号质量不低于minSignal, 将其加入队列并标记为已访问
47     if (nr >= 0 && nr < R && nc >= 0 && nc < C && !visited[nr][nc] && Cov[nr][nc] >= minSignal) {
48         queue.push(Cell(nr, nc));
49         visited[nr][nc] = true;
50     }
51 }
52 }
53 }
54
55 // 如果没有找到有效路径, 返回false
56 return false;
57 }
58
59 // 使用二分搜索找到最大的满足条件的信号质量
60 int binarySearch(const vector<vector<int>>& Cov, int low, int high) {
61     while (low <= high) {
62         int mid = low + (high - low) / 2;
63         // 如果存在一条有效路径, 尝试更高的信号质量
64         if (bfs(Cov, mid)) {
65             low = mid + 1;
66         } else { // 否则, 降低信号质量
67             high = mid - 1;
68         }
69     }
70     // 返回最大的满足条件的信号质量
71     return high;
72 }
73
74 int main() {
75     int R, C;
76     cin >> R >> C;
77     vector<vector<int>> Cov(R, vector<int>(C));
78
79     int minSignal = INT_MAX;
80     int maxSignal = INT_MIN;
81
82 }
```

```
82 // 读取网格数据, 并记录信号质量的最小值和最大值
83 for (int i = 0; i < R; i++) {
84     for (int j = 0; j < C; j++) {
85         cin >> Cov[i][j];
86         minSignal = min(minSignal, Cov[i][j]);
87         maxSignal = max(maxSignal, Cov[i][j]);
88     }
89 }
90
91 // 输出最大的满足条件的信号质量
92 cout << binarySearch(Cov, minSignal, maxSignal) << endl;
93 return 0;
}
```

Java

```
1 import java.util.LinkedList;
2 import java.util.Queue;
3 import java.util.Scanner;
4
5 public class Main {
6
7     // 定义一个内部类表示网格中的一个单元格
8     static class Cell {
9         int row, col;
10
11         Cell(int row, int col) {
12             this.row = row;
13             this.col = col;
14         }
15     }
16
17     // 使用广度优先搜索 (BFS) 检查是否存在一条从起点到终点的路径, 路径上所有单元格的信号质量都不低于minSignal
18     private static boolean bfs(int[][] Cov, int minSignal) {
19         int R = Cov.length, C = Cov[0].length;
20         // 如果起点或终点的信号质量低于minSignal, 直接返回false
21         if (Cov[0][0] < minSignal || Cov[R - 1][C - 1] < minSignal) {
22             return false;
23         }
24     }
25 }
```

```
25 // visited数组用于记录哪些单元格已经被访问过, 避免重复访问
26 boolean[][] visited = new boolean[R][C];
27 Queue<Cell> queue = new LinkedList<>();
28 queue.add(new Cell(0, 0));
29 visited[0][0] = true;
30
31 // dr和dc数组用于表示从当前单元格向四个方向 (上下左右) 移动的行和列的变化量
32 int[] dr = {1, -1, 0, 0};
33 int[] dc = {0, 0, 1, -1};
34
35 while (!queue.isEmpty()) {
36     Cell cell = queue.poll();
37     // 如果到达终点, 返回true
38     if (cell.row == R - 1 && cell.col == C - 1) {
39         return true;
40     }
41
42     // 否则, 尝试向四个方向移动
43     for (int i = 0; i < 4; i++) {
44         int nr = cell.row + dr[i];
45         int nc = cell.col + dc[i];
46
47         // 如果新的单元格在网格内, 且没有被访问过, 且信号质量不低于minSignal, 将其加入队列并标记为已访问
48         if (nr >= 0 && nr < R && nc >= 0 && nc < C && !visited[nr][nc] && Cov[nr][nc] >= minSignal) {
49             queue.add(new Cell(nr, nc));
50             visited[nr][nc] = true;
51         }
52     }
53 }
54
55 // 如果没有找到有效路径, 返回false
56 return false;
57 }
58
59 // 使用二分搜索找到最大的满足条件的信号质量
60 private static int binarySearch(int[][] Cov, int low, int high) {
61     while (low <= high) {
62         int mid = low + (high - low) / 2;
63         // 如果存在一条有效路径, 尝试更高的信号质量
64         if (bfs(Cov, mid)) {
65             low = mid + 1;
66         } else {
67             high = mid - 1;
68         }
69     }
70     return high;
71 }
```



```
66         low = mid + 1;
67     } else { // 否则, 降低信号质量
68         high = mid - 1;
69     }
70 }
71 // 返回最大的满足条件的信号质量
72 return high;
73 }
74
75 public static void main(String[] args) {
76     Scanner scanner = new Scanner(System.in);
77     int R = scanner.nextInt();
78     int C = scanner.nextInt();
79     int[][] Cov = new int[R][C];
80
81     int minSignal = Integer.MAX_VALUE;
82     int maxSignal = Integer.MIN_VALUE;
83
84     // 读取网格数据, 并记录信号质量的最小值和最大值
85     for (int i = 0; i < R; i++) {
86         for (int j = 0; j < C; j++) {
87             Cov[i][j] = scanner.nextInt();
88             minSignal = Math.min(minSignal, Cov[i][j]);
89             maxSignal = Math.max(maxSignal, Cov[i][j]);
90         }
91     }
92     scanner.close();
93
94     // 输出最大的满足条件的信号质量
95     System.out.println(binarySearch(Cov, minSignal, maxSignal));
96 }
```

JavaScript

```
1 // 导入所需的库
2 const readline = require('readline');
3
4 // 创建readline.Interface实例
5 const rl = readline.createInterface({
```

```
6     input: process.stdin,
7     output: process.stdout
8 });
9
10 // 存储所有输入行的数组
11 let lines = [];
12 rl.on('line', (line) => {
13     lines.push(line);
14 });
15
16 // 在输入完毕后执行主要逻辑
17 rl.on('close', () => {
18
19     let R = parseInt(lines[0]);
20     let C = parseInt(lines[1]);
21     let Cov = [];
22     for (let i = 0; i < R; i++) {
23         Cov.push(lines[i + 2].split(' ').map(Number));
24     }
25
26     let minSignal = Math.min(...Cov.map(row => Math.min(...row)));
27     let maxSignal = Math.max(...Cov.map(row => Math.max(...row)));
28
29     // 输出最大的满足条件的信号质量
30     console.log(binary_search(Cov, minSignal, maxSignal));
31 });
32
33 // 定义一个类表示网格中的一个单元格
34 class Cell {
35     constructor(row, col) {
36         this.row = row;
37         this.col = col;
38     }
39 }
40
41 // 使用广度优先搜索 (BFS) 检查是否存在一条从起点到终点的路径
42 // 路径上所有单元格的信号质量都不低于minSignal
43 function bfs(Cov, minSignal) {
44     let R = Cov.length, C = Cov[0].length;
45     // 如果起点或终点的信号质量低于minSignal, 直接返回false
46     --
```

```
47     if (Cov[0][0] < minSignal || Cov[R - 1][C - 1] < minSignal) {
48         return false;
49     }
50
51     // visited数组用于记录哪些单元格已经被访问过, 避免重复访问
52     let visited = Array.from({ length: R }, () => Array(C).fill(false));
53     let queue = [];
54     queue.push(new Cell(0, 0));
55     visited[0][0] = true;
56
57     // dr和dc数组用于表示从当前单元格向四个方向 (上下左右) 移动的行和列的变化量
58     let dr = [1, -1, 0, 0];
59     let dc = [0, 0, 1, -1];
60
61     while (queue.length > 0) {
62         let cell = queue.shift();
63         // 如果到达终点, 返回True
64         if (cell.row == R - 1 && cell.col == C - 1) {
65             return true;
66         }
67
68         // 否则, 尝试向四个方向移动
69         for (let i = 0; i < 4; i++) {
70             let nr = cell.row + dr[i];
71             let nc = cell.col + dc[i];
72
73             // 如果新的单元格在网格内, 且没有被访问过, 且信号质量不低于minSignal, 将其加入队列并标记为已访问
74             if (nr >= 0 && nr < R && nc >= 0 && nc < C && !visited[nr][nc] && Cov[nr][nc] >= minSignal) {
75                 queue.push(new Cell(nr, nc));
76                 visited[nr][nc] = true;
77             }
78         }
79     }
80
81     // 如果没有找到有效路径, 返回False
82     return false;
83 }
84
85 // 使用二分搜索找到最大的满足条件的信号质量
86 function binary_search(Cov, low, high) {
87
```

```
88     while (low <= high) {
89         let mid = Math.floor(low + (high - low) / 2);
90         // 如果存在一条有效路径, 尝试更高的信号质量
91         if (bfs(Cov, mid)) {
92             low = mid + 1;
93         } else { // 否则, 降低信号质量
94             high = mid - 1;
95         }
96     }
97     // 返回最大的满足条件的信号质量
98     return high;
99 }
100
101
102
```

Python

```
1  from collections import deque
2
3  # 定义一个类表示网格中的一个单元格
4  class Cell:
5      def __init__(self, row, col):
6          self.row = row
7          self.col = col
8
9  # 使用广度优先搜索 (BFS) 检查是否存在一条从起点到终点的路径
10 # 路径上所有单元格的信号质量都不低于minSignal
11 def bfs(Cov, minSignal):
12     R, C = len(Cov), len(Cov[0])
13     # 如果起点或终点的信号质量低于minSignal, 直接返回false
14     if Cov[0][0] < minSignal or Cov[R - 1][C - 1] < minSignal:
15         return False
16
17     # visited数组用于记录哪些单元格已经被访问过, 避免重复访问
18     visited = [[False for _ in range(C)] for _ in range(R)]
19     queue = deque()
20     queue.append(Cell(0, 0))
21
```

```
21 visited[0][0] = True
22
23 # dr和dc数组用于表示从当前单元格向四个方向（上下左右）移动的行和列的变化量
24 dr = [1, -1, 0, 0]
25 dc = [0, 0, 1, -1]
26
27 while queue:
28     cell = queue.popleft()
29     # 如果到达终点, 返回True
30     if cell.row == R - 1 and cell.col == C - 1:
31         return True
32
33     # 否则, 尝试向四个方向移动
34     for i in range(4):
35         nr, nc = cell.row + dr[i], cell.col + dc[i]
36
37         # 如果新的单元格在网格内, 且没有被访问过, 且信号质量不低于minSignal, 将其加入队列并标记为已访问
38         if 0 <= nr < R and 0 <= nc < C and not visited[nr][nc] and Cov[nr][nc] >= minSignal:
39             queue.append(Cell(nr, nc))
40             visited[nr][nc] = True
41
42     # 如果没有找到有效路径, 返回False
43     return False
44
45 # 使用二分搜索找到最大的满足条件的信号质量
46 def binary_search(Cov, low, high):
47     while low <= high:
48         mid = low + (high - low) // 2
49         # 如果存在一条有效路径, 尝试更高的信号质量
50         if bfs(Cov, mid):
51             low = mid + 1
52         else: # 否则, 降低信号质量
53             high = mid - 1
54     # 返回最大的满足条件的信号质量
55     return high
56
57 # 主函数
58 def main():
59     R = int(input())
60     C = int(input())
61     --
```

```
62     Cov = [list(map(int, input().split())) for _ in range(R)]
63
64     minSignal = min(min(row) for row in Cov)
65     maxSignal = max(max(row) for row in Cov)
66
67     # 输出最大的满足条件的信号质量
68     print(binary_search(Cov, minSignal, maxSignal))
69
70 if __name__ == "__main__":
    main()
```

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1](#)

[用例2](#)

[解题思路](#)

[C++](#)

[Java](#)

[javaScript](#)

[Python](#)

