# 【华为OD机考 统一考试机试C卷】核酸检测（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

为了达到新冠疫情精准防控的需要，为了避免全员核酸检测带来的浪费，需要精准圈定可能被感染的人群。

现在根据传染病流调以及大数据分析，得到了每个人之间在时间、空间上是否存在轨迹的交叉。

现在给定一组确诊人员编号 (X1,X2,X3,…Xn)在所有人当中，找出哪些人需要进行核酸检测，输出需要进行核酸检测的数。

(注意:确诊病例自身不需要再做核酸检测)需要进行核酸检测的人，是病毒传播链条上的所有人员，即有可能通过确诊病例所能传播到的所有人。

例如:A是确诊病例，A和B有接触、B和C有接触 C和D有接触，D和E有接触。那么B、C、D、E都是需要进行核酸检测的人

## 输入描述

第一行为总人数N

第二行为确证病例人员编号（确证病例人员数量<N），用逗号隔开

接下来N行，每一行有N个数字，用逗号隔开，其中第i行的第j个数字表名编号i是否与编号j接触过。0表示没有接触，1表示有接触

备注：

人员编号从0开始

0 < N < 100 0<N<1000<N<100

## 输出描述

输出需要做核酸检测的人数

## 用例

输入

```
5
1,2
1,1,0,1,0
1,1,0,0,0
0,0,1,0,1
1,0,0,1,0
0,0,1,0,1
```

输出

```
3
```

说明

> 编号为1、2号的人员为确诊病例
> 1号与0号有接触，0号与3号有接触，2号与4号有接触。所以，需要做核酸检测的人是0号、3号、4号,总计3人要进行核酸检测。

## 解题思路

1. 初始化一个大小为N的布尔数组 `visited` ，用来记录每个人是否已经被访问过（即是否已经确定需要进行核酸检测）。

2. 初始化一个大小为N×N的布尔矩阵 `contacts`，用来表示人与人之间的接触情况。如果 `contacts[i][j]` 为 `true`，则表示编号为i的人与编号为j的人有接触。

   从输入中读取确诊病例的编号，并对每个确诊病例执行深度优先搜索（DFS）：

   - 在DFS中，首先将当前节点（即当前人员编号）标记为已访问。
   - 然后遍历该节点的所有邻接节点（即与当前人员有接触的所有人），如果邻接节点未被访问，则递归地对邻接节点执行DFS。

4. 完成DFS后，遍历 `visited` 数组，统计除确诊病例外的已访问节点的数量，即为需要进行核酸检测的人数。

### 用例模拟计算过程：

1. 初始化 `visited` 数组为 `[false, false, false, false, false]`。

   构建 `contacts` 矩阵如下：

   ```
   [true,  true,  false, true,  false]
   [true,  true,  false, false, false]
   [false, false, true,  false, true ]
   [true,  false, false, true,  false]
   [false, false, true,  false, true ]
   ```

   对于确诊病例1和2，执行DFS：

   DFS(1): 标记 `visited[1]` 为 `true`，检查与1有接触的人，发现0和3，递归DFS(0)和DFS(3)。

   - DFS(0): 标记 `visited[0]` 为 `true`，检查与0有接触的人，发现3，但3已在DFS(1)中被访问，所以不再递归。
   - DFS(3): 标记 `visited[3]` 为 `true`，检查与3有接触的人，发现0，但0已在DFS(0)中被访问，所以不再递归。

   DFS(2): 标记 `visited[2]` 为 `true`，检查与2有接触的人，发现4，递归DFS(4)。

   - DFS(4): 标记 `visited[4]` 为 `true`，检查与4有接触的人，发现2，但2已在DFS(2)中被访问，所以不再递归。

4. DFS执行完毕后， `visited` 数组为 `[true, true, true, true, true]`。
5. 统计除确诊病例外的已访问节点数量，即 `visited` 中为 `true` 的元素数量减去确诊病例的数量。在这个用例中，所有人都被访问过，但需要排除确诊病例1和2，所以需要进行核酸检测的人数为5 - 2 = 3。

### C++

```cpp
#include <iostream>
#include <vector>
#include <sstream>
```

```cpp
#include <string>
#include <set>

using namespace std;
// 深度优先搜索（DFS）算法
void dfs(const vector<vector<bool>>& contacts, vector<bool>& visited, int start) {
    visited[start] = true; // 标记当前节点为已访问
    for (size_t i = 0; i < contacts.size(); ++i) {
        // 如果当前节点与其他节点有接触，并且该节点未被访问过
        if (contacts[start][i] && !visited[i]) {
            dfs(contacts, visited, i); // 递归访问该节点
        }
    }
}

int main() {
    int N;
    cin >> N;
    cin.ignore(); // 忽略换行符

    string line;
    getline(cin, line);
    istringstream iss(line);
    string caseIndex;
    set<int> confirmedCases;

    // 读取确诊病例人员编号
    while (getline(iss, caseIndex, ',')) {
        confirmedCases.insert(stoi(caseIndex));
    }

    vector<vector<bool>> contacts(N, vector<bool>(N));
    vector<bool> visited(N, false);

    // 构建接触矩阵
    for (int i = 0; i < N; ++i) {
        getline(cin, line);
        istringstream rowStream(line);
        string contact;
        int j = 0;

```

```
        while (getline(rowStream, contact, ',')) {
            contacts[i][j++] = contact == "1";
        }
    }

    // 对每个确诊病例执行深度优先搜索
    for (int index : confirmedCases) {
        if (!visited[index]) {
            dfs(contacts, visited, index);
        }
    }

    int count = 0; // 需要进行核酸检测的人数
    // 遍历访问记录数组，统计需要进行核酸检测的人数
    for (int i = 0; i < N; ++i) {
        if (visited[i] && confirmedCases.find(i) == confirmedCases.end()) {
            count++; // 如果该人员被访问过且不是确诊病例，则计数器加一
        }
    }

    cout << count << endl; // 输出需要进行核酸检测的人数
    return 0;
}
```

## Java

```
import java.util.*;

public class Main {
    // 深度优先搜索（DFS）算法
    public static void dfs(boolean[][] contacts, boolean[] visited, int start) {
        visited[start] = true; // 标记当前节点为已访问
        for (int i = 0; i < contacts.length; i++) {
            // 如果当前节点与其他节点有接触，并且该节点未被访问过
            if (contacts[start][i] == true && !visited[i]) {
                dfs(contacts, visited, i); // 递归访问该节点
            }
        }
    }

```

```java
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int N = Integer.parseInt(scanner.nextLine()); // 读取总人数
        String[] confirmedCases = scanner.nextLine().split(","); // 读取确诊病例人员编号
        boolean[][] contacts = new boolean[N][N]; // 创建接触矩阵
        boolean[] visited = new boolean[N]; // 创建访问记录数组

        // 构建接触矩阵
        for (int i = 0; i < N; i++) {
            String[] row = scanner.nextLine().split(",");
            for (int j = 0; j < N; j++) {
                contacts[i][j] = "1".equals(row[j]); // 将接触情况转换为布尔值存储
            }
        }

        // 对每个确诊病例执行深度优先搜索
        for (String caseIndex : confirmedCases) {
            int index = Integer.parseInt(caseIndex);
            dfs(contacts, visited, index);
        }

        int count = 0; // 需要进行核酸检测的人数
        // 遍历访问记录数组，统计需要进行核酸检测的人数
        for (int i = 0; i < N; i++) {
            if (visited[i]) { // 如果该人员被访问过
                // 检查该人员是否为确诊病例
                boolean isConfirmedCase = Arrays.asList(confirmedCases).contains(String.valueOf(i));
                if (!isConfirmedCase) { // 如果不是确诊病例，则计数器加一
                    count++;
                }
            }
        }

        System.out.println(count); // 输出需要进行核酸检测的人数
        scanner.close(); // 关闭扫描器
    }
}
```

## javaScript

```javascript
const readline = require('readline');
const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout
});

// 深度优先搜索（DFS）算法
function dfs(contacts, visited, start) {
  visited[start] = true; // 标记当前节点为已访问
  for (let i = 0; i < contacts.length; i++) {
    // 如果当前节点与其他节点有接触，并且该节点未被访问过
    if (contacts[start][i] && !visited[i]) {
      dfs(contacts, visited, i); // 递归访问该节点
    }
  }
}

// 读取输入数据
let lineCount = 0;
let N = 0;
let confirmedCases = [];
let contacts = [];
let visited = [];

rl.on('line', (line) => {
  if (lineCount === 0) {
    N = parseInt(line); // 读取总人数
    visited = new Array(N).fill(false); // 初始化访问记录数组
  } else if (lineCount === 1) {
    confirmedCases = line.split(',').map(Number); // 读取确诊病例人员编号
    contacts = new Array(N).fill(null).map(() => new Array(N).fill(false)); // 创建接触矩阵
  } else {
    let row = line.split(',').map(Number);
    contacts[lineCount - 2] = row.map(value => value === 1); // 构建接触矩阵
    if (lineCount - 2 === N - 1) {
      rl.close(); // 如果已读取完所有输入数据，则关闭读取接口
    }
  }
  lineCount++;
});
```

```
rl.on('close', () => {
  // 对每个确诊病例执行深度优先搜索
  confirmedCases.forEach((caseIndex) => {
    dfs(contacts, visited, caseIndex);
  });

  let count = 0; // 需要进行核酸检测的人数
  // 遍历访问记录数组，统计需要进行核酸检测的人数
  visited.forEach((hasVisited, i) => {
    if (hasVisited && !confirmedCases.includes(i)) {
      count++; // 如果该人员被访问过且不是确诊病例，则计数器加一
    }
  });

  console.log(count); // 输出需要进行核酸检测的人数
});
```

## Python

```
import sys

# 深度优先搜索（DFS）算法
def dfs(contacts, visited, start):
    visited[start] = True  # 标记当前节点为已访问
    for i in range(len(contacts)):
        # 如果当前节点与其他节点有接触，并且该节点未被访问过
        if contacts[start][i] and not visited[i]:
            dfs(contacts, visited, i)  # 递归访问该节点

# 读取输入数据
N = int(input())  # 读取总人数
confirmed_cases = list(map(int, input().split(',')))  # 读取确诊病例人员编号
contacts = []  # 创建接触矩阵
visited = [False] * N  # 创建访问记录数组

# 构建接触矩阵
for _ in range(N):
    row = list(map(int, input().split(',')))
    contacts.append([bool(x) for x in row])
```

```python

# 对每个确诊病例执行深度优先搜索
for case_index in confirmed_cases:
    dfs(contacts, visited, case_index)

count = 0  # 需要进行核酸检测的人数
# 遍历访问记录数组，统计需要进行核酸检测的人数
for i, has_visited in enumerate(visited):
    if has_visited and i not in confirmed_cases:
        count += 1  # 如果该人员被访问过且不是确诊病例，则计数器加一

print(count)  # 输出需要进行核酸检测的人数
```

**文章目录**


机考真题
华为OD
No No No
CSDN @算法大师