【DFS/BFS】-精准核酸检测

题目描述与示例

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，...，n）`，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人数。(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`

第二行为确诊病例人员编号(确诊病例人员数量 `<N`)，用逗号分割

第三行开始，为一个 `N*N` 的矩阵，表示每个人员之间是否有接触， `0` 表示没有接触， `1` 表示有接触。

输出描述

整数：需要做核酸检测的人数

补充说明

人员编号从 `0` 开始

`0 < N < 100`

示例

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
```

补充说明

编号为 `1` 、 `2` 号的人员，为确诊病例。

`1` 号和 `0` 号有接触， `0` 号和 `3` 号有接触。

`2` 号和 `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` 人需要进行核酸检测

解题思路

> 本题让人回想那段岁月..恍若隔世

非常典型的搜索问题，很容易想到直接套用DFS/BFS模板来完成。

注意所给的无向图是以关联矩阵 `mat` 来呈现的，即 `mat[i][j] == 1` 表示 `i` 和 `j` 有关联（有接触）。

另外，需要注意进行搜索的初始节点可能有多个。若

- 进行DFS，那么需要多次进行DFS入口函数的调用
- 进行BFS，那么队列的初始状态需要储存多个节点

注意：**本题存在一个非常坑的地方**，就是原本已经确诊的人是无需再做核酸检测的，只有连通块中的其他人才需要做检测。

如果不熟悉关联矩阵，也可以将关联矩阵转化为邻接表来表示。

> 复习一下无向图关联矩阵的特点：
>
> 1. `mat[i][j] == 1` 表示 `i` 和 `j` 关联，`mat[i][j] == 0` 表示 `i` 和 `j` 无关
> 2. 对角线一定为 `1`，即 `mat[i][i] == 1` 恒成立，因为每一个人总和自己关联
> 3. 关联矩阵一定沿着对角线对称，即 `mat[i][j] == mat[j][i]`，因为 `i` 和 `j` 关联等价于 `j` 和 `i` 关联

代码

解法一：DFS

Python

```python
# 题目：2024E-精准核酸检测
# 分值：100
# 作者：闭着眼睛学数理化
# 算法：DFS
# 代码看不懂的地方，请直接在群上提问


def dfs(n, i, checkList, mat):
    global ans
    ans += 1
    # 将编号i标记为已检查过
    checkList[i] = 1
    # 遍历所有与i关联的编号j
    for j in range(n):
        # 1. i和j不是同一个人
        # 2. i和j接触过
        # 3. j尚未检查过
        if j != i and mat[i][j] == 1 and checkList[j] == 0:
            dfs(n, j, checkList, mat)
```

```python

n = int(input())
# 输入初始确诊编号
startList = list(map(int, input().split(",")))
mat = list()
# 构建关联矩阵
for _ in range(n):
    mat.append(list(map(int, input().split(","))))

# 检查数组，长度为n，checkList[i]表示编号i是否已检查过
checkList = [0] * n

# 答案变量，表示需要检测的人数
ans = 0

# 遍历所有确诊编号i
for i in startList:
    # 若i尚未检查过，则从i开始，进行dfs搜索
    if checkList[i] == 0:
        dfs(n, i, checkList, mat)

# 最后要减去原本已经确诊的人，才是所有需要检测的人数
print(ans-len(startList))
```

Java

```java
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;
import java.util.StringTokenizer;

public class Main {
    static void dfs(int n, int i, int[] checkList, int[][] mat, int[] ans) {
        ans[0]++;
        checkList[i] = 1;
        for (int j = 0; j < n; j++) {
            if (j != i && mat[i][j] == 1 && checkList[j] == 0) {
                dfs(n, j, checkList, mat, ans);
            }
        }
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
```

```
        int n = scanner.nextInt();
        scanner.nextLine(); // Consume newline
        String startListStr = scanner.nextLine();
        StringTokenizer tokenizer = new StringTokenizer(startListStr, ",");
        List<Integer> startList = new ArrayList<>();
        while (tokenizer.hasMoreTokens()) {
            startList.add(Integer.parseInt(tokenizer.nextToken()));
        }
        int[][] mat = new int[n][n];
        for (int i = 0; i < n; i++) {
            String row = scanner.nextLine();
            tokenizer = new StringTokenizer(row, ",");
            for (int j = 0; j < n; j++) {
                mat[i][j] = Integer.parseInt(tokenizer.nextToken());
            }
        }
        int[] checkList = new int[n];
        int[] ans = {0};
        for (int i : startList) {
            if (checkList[i] == 0) {
                dfs(n, i, checkList, mat, ans);
            }
        }
        System.out.println(ans[0] - startList.size());
    }
}

```

JavaScript

```
```

C++

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

```

```cpp
using namespace std;

void dfs(int n, int i, vector<int> &checkList, vector<vector<int>> &mat, int &ans) {
    ans++;
    checkList[i] = 1;
    for (int j = 0; j < n; j++) {
        if (j != i && mat[i][j] == 1 && checkList[j] == 0) {
            dfs(n, j, checkList, mat, ans);
        }
    }
}

int main() {
    int n;
    cin >> n;
    cin.ignore(); // Consume newline
    string startListStr;
    getline(cin, startListStr);
    vector<int> startList;
    stringstream ss(startListStr);
    string token;
    while (getline(ss, token, ',')) {
        startList.push_back(stoi(token));
    }
    vector<vector<int>> mat(n, vector<int>(n));
    for (int i = 0; i < n; i++) {
        string row;
        getline(cin, row);
        stringstream ss(row);
        for (int j = 0; j < n; j++) {
            string cell;
            getline(ss, cell, ',');
            mat[i][j] = stoi(cell);
        }
    }
    vector<int> checkList(n);
    int ans = 0;
    for (int i : startList) {
        if (checkList[i] == 0) {
            dfs(n, i, checkList, mat, ans);
        }
    }
    cout << ans - startList.size() << endl;
    return 0;
}

```

时空复杂度

时间复杂度：`O(N^2)`。需要遍历整个关联矩阵。

空间复杂度：`O(N)`。`checkList`所占空间。

解法二：BFS

Python

```python
# 题目：2024E-精准核酸检测
# 分值：100
# 作者：闭着眼睛学数理化
# 算法：BFS
# 代码看不懂的地方，请直接在群上提问


from collections import deque

n = int(input())
# 输入初始确诊编号
startList = list(map(int, input().split(",")))
mat = list()
# 构建关联矩阵
for _ in range(n):
    mat.append(list(map(int, input().split(","))))

# 检查数组，长度为n，checkList[i]表示编号i是否已检查过
checkList = [0] * n

# 将startList中所有确诊人数标记为已检查过，才可以进行后续的BFS过程
for i in startList:
    checkList[i] = 1

# 答案变量，表示需要检测的人数
ans = 0

# 初始化队列，包含所有确诊的人
q = deque(startList)
while q:
    # 弹出队头元素，为当前需要搜索的编号
    i = q.popleft()
```

```
    ans += 1
    # 遍历所有与i关联的编号j
    for j in range(n):
        # 1. i和j不是同一个人
        # 2. i和j接触过
        # 3. j尚未检查过
        if j != i and mat[i][j] == 1 and checkList[j] == 0:
            checkList[j] = 1
            q.append(j)

# 最后要减去原本已经确诊的人，才是所有需要检测的人数
print(ans-len(startList))
```

Java

```
import java.util.ArrayDeque;
import java.util.Queue;
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int n = scanner.nextInt();
        scanner.nextLine(); // Consume newline
        String[] startListStr = scanner.nextLine().split(",");
        int[] startList = new int[startListStr.length];
        for (int i = 0; i < startListStr.length; i++) {
            startList[i] = Integer.parseInt(startListStr[i]);
        }
        int[][] mat = new int[n][n];
        for (int i = 0; i < n; i++) {
            String[] row = scanner.nextLine().split(",");
            for (int j = 0; j < n; j++) {
                mat[i][j] = Integer.parseInt(row[j]);
            }
        }
        int[] checkList = new int[n];
        for (int i : startList) {
            checkList[i] = 1;
        }
        int ans = 0;
        Queue<Integer> queue = new ArrayDeque<>();
        for (int i : startList) {
            queue.offer(i);
        }
```

```
        while (!queue.isEmpty()) {
            int i = queue.poll();
            ans++;
            for (int j = 0; j < n; j++) {
                if (j != i && mat[i][j] == 1 && checkList[j] == 0) {
                    checkList[j] = 1;
                    queue.offer(j);
                }
            }
        }
        System.out.println(ans - startList.length);
    }
}

```

JavaScript

```

```

C++

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

using namespace std;

int main() {
    int n;
    cin >> n;
    cin.ignore(); // Consume newline
    string startListStr;
    getline(cin, startListStr);
    vector<int> startList;
    stringstream ss(startListStr);
    string temp;
    while (getline(ss, temp, ',')) {
```

```
        startList.push_back(stoi(temp));
    }
    vector<vector<int>> mat(n, vector<int>(n));
    for (int i = 0; i < n; i++) {
        string row;
        getline(cin, row);
        stringstream ss(row);
        for (int j = 0; j < n; j++) {
            string cell;
            getline(ss, cell, ',');
            mat[i][j] = stoi(cell);
        }
    }
    vector<int> checkList(n);
    for (int i : startList) {
        checkList[i] = 1;
    }
    int ans = 0;
    queue<int> q;
    for (int i : startList) {
        q.push(i);
    }
    while (!q.empty()) {
        int i = q.front();
        q.pop();
        ans++;
        for (int j = 0; j < n; j++) {
            if (j != i && mat[i][j] == 1 && checkList[j] == 0) {
                checkList[j] = 1;
                q.push(j);
            }
        }
    }
    cout << ans - startList.size() << endl;
    return 0;
}

```

时空复杂度

时间复杂度：`O(N^2)`。需要遍历整个关联矩阵。

空间复杂度：`O(N)`。`checkList` 和 `q` 所占空间。