

【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 $(x_1, x_2, x_3, \dots, n)$ ，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 $N \times N$ 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

```
1 5
2 1,2
3 1,1,0,1,0
4 1,1,0,0,0
5 0,0,1,0,1
6 1,0,0,1,0
7 0,0,1,0,1
```

输出

```
1 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. $\text{mat}[i][j] == 1$ 表示 i 和 j 关联， $\text{mat}[i][j] == 0$ 表示 i 和 j 无关
2. 对角线一定为 1，即 $\text{mat}[i][i] == 1$ 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 $\text{mat}[i][j] == \text{mat}[j][i]$ ，因为 i 和 j 关联等价于 j 和 i 关联

代码

解法一：DFS

Python

```
1 # 题目：2024E-精准核酸检测
2 # 分值：100
3 # 作者：闭着眼睛学数理化
4 # 算法：DFS
5 # 代码看不懂的地方，请直接在群上提问
6
7
8 def dfs(n, i, checkList, mat):
9     global ans
10    ans += 1
11    # 将编号i标记为已检查过
12    checkList[i] = 1
13    # 遍历所有与i关联的编号j
14    for j in range(n):
15        # 1. i和j不是同一个人
16        # 2. i和j接触过
17        # 3. j尚未检查过
18        if j != i and mat[i][j] == 1 and checkList[j] == 0:
19            dfs(n, j, checkList, mat)
```

```

20
21
22 n = int(input())
23 # 输入初始确诊编号
24 startList = list(map(int, input().split(",")))
25 mat = list()
26 # 构建关联矩阵
27 for _ in range(n):
28     mat.append(list(map(int, input().split(","))))
29
30 # 检查数组，长度为n，checkList[i]表示编号i是否已检查过
31 checkList = [0] * n
32
33 # 答案变量，表示需要检测的人数
34 ans = 0
35
36 # 遍历所有确诊编号i
37 for i in startList:
38     # 若i尚未检查过，则从i开始，进行dfs搜索
39     if checkList[i] == 0:
40         dfs(n, i, checkList, mat)
41
42 # 最后要减去原本已经确诊的人，才是所有需要检测的人数
43 print(ans-len(startList))

```

Java

```

1 import java.util.ArrayList;
2 import java.util.List;
3 import java.util.Scanner;
4 import java.util.StringTokenizer;
5
6 public class Main {
7     static void dfs(int n, int i, int[] checkList, int[][] mat, int[] ans) {
8         ans[0]++;
9         checkList[i] = 1;
10        for (int j = 0; j < n; j++) {
11            if (j != i && mat[i][j] == 1 && checkList[j] == 0) {
12                dfs(n, j, checkList, mat, ans);
13            }
14        }
15    }
16
17    public static void main(String[] args) {
18        Scanner scanner = new Scanner(System.in);

```

```

19     int n = scanner.nextInt();
20     scanner.nextLine(); // Consume newline
21     String startListStr = scanner.nextLine();
22     StringTokenizer tokenizer = new StringTokenizer(startListStr, ",");
23     List<Integer> startList = new ArrayList<>();
24     while (tokenizer.hasMoreTokens()) {
25         startList.add(Integer.parseInt(tokenizer.nextToken()));
26     }
27     int[][] mat = new int[n][n];
28     for (int i = 0; i < n; i++) {
29         String row = scanner.nextLine();
30         tokenizer = new StringTokenizer(row, ",");
31         for (int j = 0; j < n; j++) {
32             mat[i][j] = Integer.parseInt(tokenizer.nextToken());
33         }
34     }
35     int[] checkList = new int[n];
36     int[] ans = {0};
37     for (int i : startList) {
38         if (checkList[i] == 0) {
39             dfs(n, i, checkList, mat, ans);
40         }
41     }
42     System.out.println(ans[0] - startList.size());
43 }
44 }
45

```

JavaScript

```

1

```

C++

```

1 #include <iostream>
2 #include <vector>
3 #include <sstream>
4

```

```

5 using namespace std;
6
7 void dfs(int n, int i, vector<int> &checkList, vector<vector<int>> &mat, int
  &ans) {
8     ans++;
9     checkList[i] = 1;
10    for (int j = 0; j < n; j++) {
11        if (j != i && mat[i][j] == 1 && checkList[j] == 0) {
12            dfs(n, j, checkList, mat, ans);
13        }
14    }
15 }
16
17 int main() {
18     int n;
19     cin >> n;
20     cin.ignore(); // Consume newline
21     string startListStr;
22     getline(cin, startListStr);
23     vector<int> startList;
24     stringstream ss(startListStr);
25     string token;
26     while (getline(ss, token, ',')) {
27         startList.push_back(stoi(token));
28     }
29     vector<vector<int>> mat(n, vector<int>(n));
30     for (int i = 0; i < n; i++) {
31         string row;
32         getline(cin, row);
33         stringstream ss(row);
34         for (int j = 0; j < n; j++) {
35             string cell;
36             getline(ss, cell, ',');
37             mat[i][j] = stoi(cell);
38         }
39     }
40     vector<int> checkList(n);
41     int ans = 0;
42     for (int i : startList) {
43         if (checkList[i] == 0) {
44             dfs(n, i, checkList, mat, ans);
45         }
46     }
47     cout << ans - startList.size() << endl;
48     return 0;
49 }
50

```

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

```
1 # 题目：2024E-精准核酸检测
2 # 分值：100
3 # 作者：闭着眼睛学数理化
4 # 算法：BFS
5 # 代码看不懂的地方，请直接在群上提问
6
7
8 from collections import deque
9
10 n = int(input())
11 # 输入初始确诊编号
12 startList = list(map(int, input().split(",")))
13 mat = list()
14 # 构建关联矩阵
15 for _ in range(n):
16     mat.append(list(map(int, input().split(","))))
17
18 # 检查数组，长度为n，checkList[i]表示编号i是否已检查过
19 checkList = [0] * n
20
21 # 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
22 for i in startList:
23     checkList[i] = 1
24
25 # 答案变量，表示需要检测的人数
26 ans = 0
27
28 # 初始化队列，包含所有确诊的人
29 q = deque(startList)
30 while q:
31     # 弹出队头元素，为当前需要搜索的编号
32     i = q.popleft()
```

```

33     ans += 1
34     # 遍历所有与i关联的编号j
35     for j in range(n):
36         # 1. i和j不是同一个人
37         # 2. i和j接触过
38         # 3. j尚未检查过
39         if j != i and mat[i][j] == 1 and checkList[j] == 0:
40             checkList[j] = 1
41             q.append(j)
42
43 # 最后要减去原本已经确诊的人，才是所有需要检测的人数
44 print(ans-len(startList))

```

Java

```

1 import java.util.ArrayDeque;
2 import java.util.Queue;
3 import java.util.Scanner;
4
5 public class Main {
6     public static void main(String[] args) {
7         Scanner scanner = new Scanner(System.in);
8         int n = scanner.nextInt();
9         scanner.nextLine(); // Consume newline
10        String[] startListStr = scanner.nextLine().split(",");
11        int[] startList = new int[startListStr.length];
12        for (int i = 0; i < startListStr.length; i++) {
13            startList[i] = Integer.parseInt(startListStr[i]);
14        }
15        int[][] mat = new int[n][n];
16        for (int i = 0; i < n; i++) {
17            String[] row = scanner.nextLine().split(",");
18            for (int j = 0; j < n; j++) {
19                mat[i][j] = Integer.parseInt(row[j]);
20            }
21        }
22        int[] checkList = new int[n];
23        for (int i : startList) {
24            checkList[i] = 1;
25        }
26        int ans = 0;
27        Queue<Integer> queue = new ArrayDeque<>();
28        for (int i : startList) {
29            queue.offer(i);
30        }

```



```

31         while (!queue.isEmpty()) {
32             int i = queue.poll();
33             ans++;
34             for (int j = 0; j < n; j++) {
35                 if (j != i && mat[i][j] == 1 && checkList[j] == 0) {
36                     checkList[j] = 1;
37                     queue.offer(j);
38                 }
39             }
40         }
41         System.out.println(ans - startList.length);
42     }
43 }
44

```

JavaScript

```

1

```

C++

```

1 #include <iostream>
2 #include <vector>
3 #include <queue>
4 #include <sstream>
5
6 using namespace std;
7
8 int main() {
9     int n;
10    cin >> n;
11    cin.ignore(); // Consume newline
12    string startListStr;
13    getline(cin, startListStr);
14    vector<int> startList;
15    stringstream ss(startListStr);
16    string temp;
17    while (getline(ss, temp, ',')) {

```

```

18     startList.push_back(stoi(temp));
19 }
20 vector<vector<int>> mat(n, vector<int>(n));
21 for (int i = 0; i < n; i++) {
22     string row;
23     getline(cin, row);
24     stringstream ss(row);
25     for (int j = 0; j < n; j++) {
26         string cell;
27         getline(ss, cell, ',');
28         mat[i][j] = stoi(cell);
29     }
30 }
31 vector<int> checkList(n);
32 for (int i : startList) {
33     checkList[i] = 1;
34 }
35 int ans = 0;
36 queue<int> q;
37 for (int i : startList) {
38     q.push(i);
39 }
40 while (!q.empty()) {
41     int i = q.front();
42     q.pop();
43     ans++;
44     for (int j = 0; j < n; j++) {
45         if (j != i && mat[i][j] == 1 && checkList[j] == 0) {
46             checkList[j] = 1;
47             q.push(j);
48         }
49     }
50 }
51 cout << ans - startList.size() << endl;
52 return 0;
53 }
54

```

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。