【DFS/BFS】-广播服务器

题目描述与示例

题目描述

服务器连接方式包括直接相连，间接连接。`A` 和 `B` 直接连接，`B` 和 `C` 直接连接，则 `A` 和 `C` 间接连接。
直接连接和间接连接都可以发送广播。
给出一个大小为 `N*N` 的二维矩阵`matrix`，代表 `N` 个服务器。`matrix[i][j] = 1`，则代表 `i` 和 `j` 直接连接；`matrix[i][j] = 0` 时，代表 `i` 和 `j` 不直接连接。`matrix[i][j]==1`，即自己和自已直接连接。
计算初始需要给几台服务器广播，才可以使每个服务器都收到广播。

输入

输入为 `N` 行，每行有 `N` 个数字，为 `0` 成 `1`，由空格分隔，构成 `N*N` 的二维矩阵`matrix`，`N` 的范围为 `1 <= N <= 40`。

输出

输出一个数字，为需要广播的服务器的数量。

示例一

输入

```
1 1 0 0
2 0 1 0
3 0 0 1
```

输出

```
1 3
```

示例二

输入

```
1 1
1 1
```

输出

```
1
```

解题思路

> 本题和LC547.省份数量不能说毫无联系，只能说一模一样。

代码

解法一：BFS

Python

```python
# 题目：2024E-广播服务器
# 分值：200
# 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

isConnected = list()
# 先输入第一行
isConnected.append(list(map(int, input().split())))
# 根据第一行的长度，得到n
n = len(isConnected[0])
# 输入剩余的n-1行
for _ in range(n-1):
    isConnected.append(list(map(int, input().split())))

```

```python
ans = 0
checkList = [0] * n      # 构建检查数组checkList

# 遍历每一个服务器
for i in range(n):
    if checkList[i] == 0:        # 若服务器i未检查过
        q = deque([i])           # 把服务器i加入q中，作为BFS的起始位置
        checkList[i] = 1         # 将服务器i标记为已检查过
        # 从服务器i开始，进行BFS
        while(q):
            # 弹出q队头的服务器x，考虑所有与其相连的服务器y
            x = q.popleft()
            # 对于服务器x，遍历所有其他服务器y，若y未检查过，且与x相连
            for y in range(n):
                if x != y and checkList[y] == 0 and isConnected[x][y] == 1:
                    q.append(y)          # 则把服务器y加入队列中
                    checkList[y] = 1     # 同时把服务器y标记为已检查过
        # 完成本次BFS，连通分量+1，即ans+1
        ans += 1

print(ans)
```

Java

```java
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        List<List<Integer>> isConnected = new ArrayList<>();

        String[] firstLine = scanner.nextLine().split(" ");
        List<Integer> firstRow = new ArrayList<>();
        for (String s : firstLine) {
            firstRow.add(Integer.parseInt(s));
        }
        isConnected.add(firstRow);

        int n = firstRow.size();
        for (int i = 1; i < n; i++) {
            String[] line = scanner.nextLine().split(" ");
            List<Integer> row = new ArrayList<>();
            for (String s : line) {
                row.add(Integer.parseInt(s));
            }
```

```
            isConnected.add(row);
        }

        int ans = 0;
        int[] checkList = new int[n];

        for (int i = 0; i < n; i++) {
            if (checkList[i] == 0) {
                Queue<Integer> q = new LinkedList<>();
                q.add(i);
                checkList[i] = 1;

                while (!q.isEmpty()) {
                    int x = q.poll();
                    for (int y = 0; y < n; y++) {
                        if (x != y && checkList[y] == 0 &&
isConnected.get(x).get(y) == 1) {
                            q.add(y);
                            checkList[y] = 1;
                        }
                    }
                }
                ans++;
            }
        }

        System.out.println(ans);
    }
}

```

C++

```
#include <iostream>
#include <sstream>
#include <vector>
#include <queue>
using namespace std;

int main() {
    vector<vector<int>> isConnected;
    string line;
    getline(cin, line);
    istringstream iss(line);
    int val;
```

```cpp
    vector<int> firstRow;
    while (iss >> val) {
        firstRow.push_back(val);
    }
    isConnected.push_back(firstRow);

    int n = firstRow.size();
    for (int i = 1; i < n; i++) {
        getline(cin, line);
        istringstream iss(line);
        vector<int> row;
        while (iss >> val) {
            row.push_back(val);
        }
        isConnected.push_back(row);
    }

    int ans = 0;
    vector<int> checkList(n, 0);

    for (int i = 0; i < n; i++) {
        if (checkList[i] == 0) {
            queue<int> q;
            q.push(i);
            checkList[i] = 1;

            while (!q.empty()) {
                int x = q.front();
                q.pop();
                for (int y = 0; y < n; y++) {
                    if (x != y && checkList[y] == 0 && isConnected[x][y] == 1)
{
                        q.push(y);
                        checkList[y] = 1;
                    }
                }
            }
            ans++;
        }
    }

    cout << ans << endl;
    return 0;
}

```

解法二：DFS

Python

```python
# 题目：2024E-广播服务器
# 分值：200
# 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


# dfs递归函数
def dfs(x, isConnected, checkList):
    # 对于传入的服务器x，将其标记为已检查过
    checkList[x] = 1
    # 遍历其他服务器y，若服务器y未检查过，且与服务器x相连
    for y in range(n):
        if y != x and checkList[y] == 0 and isConnected[x][y] == 1:
            dfs(y, isConnected, checkList)  # 则对y进行DFS

isConnected = list()
# 先输入第一行
isConnected.append(list(map(int, input().split())))
# 根据第一行的长度，得到n
n = len(isConnected[0])
# 输入剩余的n-1行
for _ in range(n-1):
    isConnected.append(list(map(int, input().split())))

ans = 0
checkList = [0] * n          # 构建检查数组checkList

# 遍历每一个服务器i
for i in range(n):
    # 如果该服务器没检查过，则以服务器i作为起始点，进行DFS
    if checkList[i] == 0:
        # 进行DFS搜索
        dfs(i, isConnected, checkList)
        # 完成本次DFS，连通分量+1，即ans+1
        ans += 1

print(ans)
```

Java

```java
import java.util.*;

public class Main {
    static void dfs(int x, List<List<Integer>> isConnected, int[] checkList) {
        checkList[x] = 1;
        int n = isConnected.size();
        for (int y = 0; y < n; y++) {
            if (y != x && checkList[y] == 0 && isConnected.get(x).get(y) == 1)
{
                dfs(y, isConnected, checkList);
            }
        }
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        List<List<Integer>> isConnected = new ArrayList<>();
        String[] firstLine = scanner.nextLine().split(" ");
        List<Integer> firstRow = new ArrayList<>();
        for (String s : firstLine) {
            firstRow.add(Integer.parseInt(s));
        }
        isConnected.add(firstRow);

        int n = firstRow.size();
        for (int i = 1; i < n; i++) {
            String[] line = scanner.nextLine().split(" ");
            List<Integer> row = new ArrayList<>();
            for (String s : line) {
                row.add(Integer.parseInt(s));
            }
            isConnected.add(row);
        }

        int ans = 0;
        int[] checkList = new int[n];

        for (int i = 0; i < n; i++) {
            if (checkList[i] == 0) {
                dfs(i, isConnected, checkList);
                ans++;
            }
```

```
        }

        System.out.println(ans);
    }
}

```

C++

```cpp
#include <iostream>
#include <sstream>
#include <vector>
using namespace std;

void dfs(int x, vector<vector<int>>& isConnected, vector<int>& checkList) {
    checkList[x] = 1;
    int n = isConnected.size();
    for (int y = 0; y < n; y++) {
        if (y != x && checkList[y] == 0 && isConnected[x][y] == 1) {
            dfs(y, isConnected, checkList);
        }
    }
}

int main() {
    vector<vector<int>> isConnected;
    string line;
    getline(cin, line);
    istringstream iss(line);
    int val;
    vector<int> firstRow;
    while (iss >> val) {
        firstRow.push_back(val);
    }
    isConnected.push_back(firstRow);

    int n = firstRow.size();
    for (int i = 1; i < n; i++) {
        getline(cin, line);
        istringstream iss(line);
        vector<int> row;
        while (iss >> val) {
            row.push_back(val);
        }
        isConnected.push_back(row);
```

```
    }

    int ans = 0;
    vector<int> checkList(n, 0);

    for (int i = 0; i < n; i++) {
        if (checkList[i] == 0) {
            dfs(i, isConnected, checkList);
            ans++;
        }
    }

    cout << ans << 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 `isConnected`。

空间复杂度：`O(N)`。

相同问题不同描述

2023B-快递业务站

题目

快递业务范围有 `N` 个站点，`A` 站点与 `B` 站点可以中转快递，则认为 `A-B` 站可达。

如果 `A-B` 可达，`B-C` 可达，则 `A-C` 可达。

现在给 `N` 个站点编号 `0, 1, ..., n-1`，用 `s[i][j]` 表示 `i-j` 是否可达。

`s[i][j] = 1` 表示 `i-j` 可达，`s[i][j] = 0` 表示 `i-j` 不可达。

现用二维数组给定 `N` 个站点的可达关系，请计算至少选择从几个主站点出发，才能可达所有站点（覆盖所有站点业务）。说明：`s[i][j]` 与 `s[j][i]` 取值相同。

输入描述

第一行输入为 `N`， `N` 表示站点个数。 `1 < N < 10000`

之后 `N` 行表示站点之间的可达关系，第 `i` 行第 `j` 个数值表示编号为 `i` 和 `j` 之间是否可达。

输出描述

输出站点个数，表示至少需要多少个主站点。

示例一

输入

```
4
1 1 1 1
1 1 1 0
1 1 1 0
1 0 0 1
```

输出

```
1
```

说明

选择 0 号站点作为主站点， 0 站点可达其他所有站点，
所以至少选择 1 个站点作为主站才能覆盖所有站点业务

示例二

输入

```
4
1 1 0 0
1 1 0 0
0 0 1 0
0 0 0 1
```

输出

```
1 3
```

说明

选择 0 号站点可以覆盖 0、1 站点，
选择 2 号站点可以覆盖 2 号站点，
选择 3 号站点可以覆盖 3 号站点，
所以至少选择 3 个站点作为主站才能覆盖所有站点业务