【华为OD机考 统一考试机试C卷】5G网络建设（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

需要在某城市进行 5G 网络建设，已经选取N个地点设置5G基站，编号固定为1到N，接下来需要各个基站之间使用光纤进行连接以确保基站能互联互通，不同基站之间假设光纤的成本各不相同，且有些节点之间已经存在光纤相连。

请你设计算法，计算出能联通这些基站的最小成本是多少。

注意：基站的联通具有传递性，比如基站A与基站B架设了光纤，基站B与基站C也架设了光纤，则基站A与基站C视为可以互相联通。

输入描述

第一行输入表示基站的个数N，其中：

- 0 < N ≤ 20
  第二行输入表示具备光纤直连条件的基站对的数目M，其中：
- 0 < M < N * (N - 1) / 2
  从第三行开始连续输入M行数据，格式为

```
X Y Z P
```

其中：

X，Y 表示基站的编号

- 0 < X ≤ N
- 0 < Y ≤ N
- X ≠ Y
  Z 表示在 X、Y之间架设光纤的成本
- 0 < Z < 100

P 表示是否已存在光纤连接，0 表示未连接，1表示已连接

输出描述

如果给定条件，可以建设成功互联互通的5G网络，则输出最小的建设成本

如果给定条件，无法建设成功互联互通的5G网络，则输出 -1

用例1

输入

```
3
3
1 2 3 0
1 3 1 0
2 3 5 0
```

输出

```
4
```

说明

> 只需要在1，2以及1，3基站之间铺设光纤，其成本为3+1=4

用例2

输入

```
3
1
1 2 5 0
```

输出

```
-1
```

说明

> 3基站无法与其他基站连接，输出-1

用例3

输入

```
3
3
1 2 3 0
1 3 1 0
2 3 5 1
```

输出

```
1
```

说明

> 2，3基站已有光纤相连，只要在1，3基站之间铺设光纤，其成本为1

解题思路

这个问题可以使用最小生成树算法（如Prim算法或Kruskal算法）来解决。在这个问题中，基站可以看作是图的顶点，光纤可以看作是图的边，光纤的成本可以看作是边的权重。已经存在的光纤可以看作是权重为0的边。我们的目标是找到一个最小生成树，即一个包含所有顶点且边的权重之和最小的树。

C++

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
#include <limits>

using namespace std;


// 定义边的结构体
struct Edge {
    int u, v, cost, pre;
    // 构造函数
    Edge(int u, int v, int cost, int pre) : u(u), v(v), cost(cost), pre(pre) {}
};

// 并查集数组
vector<int> parent;

// 并查集查找函数，用于查找x所在的集合
int find(int x) {
    // 如果x不是自己的父节点，那么就让x的父节点为x的父节点的父节点（路径压缩）
    if (parent[x] != x) {
        parent[x] = find(parent[x]);
    }
    // 返回x的父节点
    return parent[x];
}

// 并查集合并函数，用于合并x和y所在的集合
void unionSet(int x, int y) {
    int rootX = find(x); // 找到x的根节点
    int rootY = find(y); // 找到y的根节点
    // 如果x和y的根节点不同，那么就将x的根节点的父节点设为y的根节点

```

```
    if (rootX != rootY) {
        parent[rootX] = rootY;
    }
}

int main() {
    int N, M; // 基站的个数和具备光纤直连条件的基站对的数目
    cin >> N >> M;
    parent.resize(N + 1); // 初始化并查集数组
    for (int i = 1; i <= N; i++) {
        parent[i] = i; // 初始时每个节点的父节点就是自己
    }
    vector<Edge> edges; // 存储所有的边
    for (int i = 0; i < M; i++) {
        int X, Y, Z, P; // 基站X，基站Y，架设光纤的成本，是否已存在光纤连接
        cin >> X >> Y >> Z >> P;
        edges.push_back(Edge(X, Y, Z, P)); // 添加边
        if (P == 1) { // 如果已存在光纤连接，那么就将X和Y合并
            unionSet(X, Y);
        }
    }
    // 将所有的边按照成本从小到大排序
    sort(edges.begin(), edges.end(), [](const Edge& a, const Edge& b) {
        return a.cost < b.cost;
    });
    int cost = 0; // 总的成本
    // 遍历所有的边
    for (const Edge& edge : edges) {
        // 如果边的两个端点不在同一个集合中，那么就将这条边添加到最小生成树中
        if (find(edge.u) != find(edge.v)) {
            cost += edge.cost; // 累加成本
            unionSet(edge.u, edge.v); // 合并边的两个端点所在的集合
        }
    }
    // 检查所有的基站是否都在同一个集合中
    for (int i = 2; i <= N; i++) {
        // 如果有基站不在同一个集合中，那么就输出-1并结束程序
        if (find(i) != find(1)) {
            cout << -1 << endl;
            return 0;
```

```
        }
    }
    // 输出总的成本
    cout << cost << endl;
    return 0;
}
```

Java

```
import java.util.*;



public class Main {
    // 并查集数组
    static int[] parent;

    // 并查集查找函数，用于查找x所在的集合
    static int find(int x) {
        // 如果x不是自己的父节点，那么就让x的父节点为x的父节点的父节点（路径压缩）
        if (parent[x] != x) {
            parent[x] = find(parent[x]);
        }
        // 返回x的父节点
        return parent[x];
    }

    // 定义边的类
    static class Edge {
        int u, v, cost, pre;

        // 构造函数
        public Edge(int u, int v, int cost, int pre) {
            this.u = u; // 基站u
            this.v = v; // 基站v
            this.cost = cost; // 架设光纤的成本
            this.pre = pre; // 是否已存在光纤连接
        }
    }
    // 并查集合并函数，用于合并x和y所在的集合
```

```
    static void union(int x, int y) {
        int rootX = find(x); // 找到x的根节点
        int rootY = find(y); // 找到y的根节点
        // 如果x和y的根节点不同，那么就将x的根节点的父节点设为y的根节点
        if (rootX != rootY) {
            parent[rootX] = rootY;
        }
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int N = scanner.nextInt(); // 基站的个数
        int M = scanner.nextInt(); // 具备光纤直连条件的基站对的数目
        parent = new int[N + 1]; // 初始化并查集数组
        for (int i = 1; i <= N; i++) {
            parent[i] = i; // 初始时每个节点的父节点就是自己
        }
        List<Edge> edges = new ArrayList<>(); // 存储所有的边
        for (int i = 0; i < M; i++) {
            int X = scanner.nextInt(); // 基站X
            int Y = scanner.nextInt(); // 基站Y
            int Z = scanner.nextInt(); // 架设光纤的成本
            int P = scanner.nextInt(); // 是否已存在光纤连接
            edges.add(new Edge(X, Y, Z, P)); // 添加边
            if (P == 1) { // 如果已存在光纤连接，那么就将X和Y合并
                union(X, Y);
            }
        }
        // 将所有的边按照成本从小到大排序
        edges.sort((a, b) -> a.cost - b.cost);
        int cost = 0; // 总的成本
        // 遍历所有的边
        for (Edge edge : edges) {
            // 如果边的两个端点不在同一个集合中，那么就将这条边添加到最小生成树中
            if (find(edge.u) != find(edge.v)) {
                cost += edge.cost; // 累加成本
                union(edge.u, edge.v); // 合并边的两个端点所在的集合
            }
        }
        // 检查所有的基站是否都在同一个集合中
```

```
        for (int i = 2; i <= N; i++) {
            // 如果有基站不在同一个集合中，那么就输出-1并结束程序
            if (find(i) != find(1)) {
                System.out.println(-1);
                return;
            }
        }
        // 输出总的成本
        System.out.println(cost);
    }
}
```

javaScript

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

let lines = [];
rl.on('line', (line) => {
    lines.push(line);
    if(lines.length === 2 + parseInt(lines[1])) {
        rl.close();
    }
});

rl.on('close', () => {
    // 定义边的类
    class Edge {
        constructor(u, v, cost, pre) {
            this.u = u;  // 基站u
            this.v = v;  // 基站v
            this.cost = cost;  // 架设光纤的成本
            this.pre = pre;  // 是否已存在光纤连接
        }
    }

    // 并查集查找函数，用于查找x所在的集合
```

```
function find(x) {
    if (parent[x] !== x) {
        // 如果x不是自己的父节点，那么就让x的父节点为x的父节点的父节点（路径压缩）
        parent[x] = find(parent[x]);
    }
    return parent[x];  // 返回x的父节点
}

// 并查集合并函数，用于合并x和y所在的集合
function union(x, y) {
    let rootX = find(x);  // 找到x的根节点
    let rootY = find(y);  // 找到y的根节点
    if (rootX !== rootY) {
        // 如果x和y的根节点不同，那么就将x的根节点的父节点设为y的根节点
        parent[rootX] = rootY;
    }
}

let N = parseInt(lines[0]);  // 输入基站的个数
let M = parseInt(lines[1]);  // 输入具备光纤直连条件的基站对的数目
let parent = Array.from({length: N + 1}, (_, i) => i);  // 初始化并查集数组，初始时每个节点的父节点就是自己
let edges = [];  // 存储所有的边

for (let i = 2; i < lines.length; i++) {
    let [X, Y, Z, P] = lines[i].split(' ').map(Number);  // 输入基站X，Y，架设光纤的成本Z，是否已存在光纤连接P
    edges.push(new Edge(X, Y, Z, P));  // 添加边
    if (P === 1) {  // 如果已存在光纤连接，那么就将X和Y合并
        union(X, Y);
    }
}

// 将所有的边按照成本从小到大排序
edges.sort((a, b) => a.cost - b.cost);
let cost = 0;  // 总的成本

for (let edge of edges) {
    // 如果边的两个端点不在同一个集合中，那么就将这条边添加到最小生成树中
    if (find(edge.u) !== find(edge.v)) {
        cost += edge.cost;  // 累加成本
        union(edge.u, edge.v);  // 合并边的两个端点所在的集合
```

```
        }
    }

    for (let i = 2; i <= N; i++) {
        // 检查所有的基站是否都在同一个集合中
        if (find(i) !== find(1)) {
            // 如果有基站不在同一个集合中，那么就输出-1并结束程序
            console.log(-1);
            return;
        }
    }

    // 输出总的成本
    console.log(cost);
});
```

Python

```
class Edge:
    # 定义边的类
    def __init__(self, u, v, cost, pre):
        self.u = u  # 基站u
        self.v = v  # 基站v
        self.cost = cost  # 架设光纤的成本
        self.pre = pre  # 是否已存在光纤连接

def find(x):
    # 并查集查找函数，用于查找x所在的集合
    if parent[x] != x:
        # 如果x不是自己的父节点，那么就让x的父节点为x的父节点的父节点（路径压缩）
        parent[x] = find(parent[x])
    return parent[x]  # 返回x的父节点

def union(x, y):
    # 并查集合并函数，用于合并x和y所在的集合
    rootX = find(x)  # 找到x的根节点
    rootY = find(y)  # 找到y的根节点
    if rootX != rootY:
        # 如果x和y的根节点不同，那么就将x的根节点的父节点设为y的根节点
        parent[rootX] = rootY
```

```python

if __name__ == "__main__":
    N = int(input())  # 输入基站的个数
    M = int(input())  # 输入具备光纤直连条件的基站对的数目
    parent = [i for i in range(N + 1)]  # 初始化并查集数组，初始时每个节点的父节点就是自己
    edges = []  # 存储所有的边

    for _ in range(M):
        X, Y, Z, P = map(int, input().split())  # 输入基站X，Y，架设光纤的成本Z，是否已存在光纤连接P
        edges.append(Edge(X, Y, Z, P))  # 添加边
        if P == 1:  # 如果已存在光纤连接，那么就将X和Y合并
            union(X, Y)

    # 将所有的边按照成本从小到大排序
    edges.sort(key=lambda edge: edge.cost)
    cost = 0  # 总的成本

    for edge in edges:
        # 如果边的两个端点不在同一个集合中，那么就将这条边添加到最小生成树中
        if find(edge.u) != find(edge.v):
            cost += edge.cost  # 累加成本
            union(edge.u, edge.v)  # 合并边的两个端点所在的集合

    for i in range(2, N + 1):
        # 检查所有的基站是否都在同一个集合中
        if find(i) != find(1):
            # 如果有基站不在同一个集合中，那么就输出-1并结束程序
            print(-1)
            break
    else:
        # 输出总的成本
        print(cost)
```

文章目录