

【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](#)[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：

- $0 < N \leq 20$
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

1 | X Y Z P

其中：

X, Y 表示基站的编号

- $0 < X \leq N$
- $0 < Y \leq N$
- $X \neq Y$
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

1	3
2	3
3	1 2 3 0
4	1 3 1 0
5	2 3 5 0

输出

1	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

1	3
2	1
3	1 2 5 0

输出

1	-1
---	----

说明

3基站无法与其他基站连接，输出-1

用例3

输入

1	3
2	3
3	1 2 3 0
4	1 3 1 0
5	2 3 5 1

输出

1	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++

```
1  #include <iostream>
2  #include <vector>
3  #include <algorithm>
4  #include <limits>
5
6  using namespace std;
7
8
9  // 定义边的结构体
10 struct Edge {
11     int u, v, cost, pre;
12     // 构造函数
13     Edge(int u, int v, int cost, int pre) : u(u), v(v), cost(cost), pre(pre) {}
14 };
15
16 // 并查集数组
17 vector<int> parent;
18
19 // 并查集查找函数，用于查找x所在的集合
20 int find(int x) {
21     // 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）
22     if (parent[x] != x) {
23         parent[x] = find(parent[x]);
24     }
25     // 返回x的父节点
26     return parent[x];
27 }
28
29 // 并查集合并函数，用于合并x和y所在的集合
30 void unionSet(int x, int y) {
31     int rootX = find(x); // 找到x的根节点
32     int rootY = find(y); // 找到y的根节点
33     // 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
34 }
```

```
35     if (rootX != rootY) {
36         parent[rootX] = rootY;
37     }
38 }
39
40 int main() {
41     int N, M; // 基站个数和具备光纤直连条件的基站对的数目
42     cin >> N >> M;
43     parent.resize(N + 1); // 初始化并查集数组
44     for (int i = 1; i <= N; i++) {
45         parent[i] = i; // 初始时每个节点的父节点就是自己
46     }
47     vector<Edge> edges; // 存储所有的边
48     for (int i = 0; i < M; i++) {
49         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
50         cin >> X >> Y >> Z >> P;
51         edges.push_back(Edge(X, Y, Z, P)); // 添加边
52         if (P == 1) { // 如果已存在光纤连接, 那么就将X和Y合并
53             unionSet(X, Y);
54         }
55     }
56     // 将所有的边按照成本从小到大排序
57     sort(edges.begin(), edges.end(), [](const Edge& a, const Edge& b) {
58         return a.cost < b.cost;
59     });
60     int cost = 0; // 总的成本
61     // 遍历所有的边
62     for (const Edge& edge : edges) {
63         // 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
64         if (find(edge.u) != find(edge.v)) {
65             cost += edge.cost; // 累加成本
66             unionSet(edge.u, edge.v); // 合并边的两个端点所在的集合
67         }
68     }
69     // 检查所有的基站是否都在同一个集合中
70     for (int i = 2; i <= N; i++) {
71         // 如果有基站不在同一个集合中, 那么就输出-1并结束程序
72         if (find(i) != find(1)) {
73             cout << -1 << endl;
74             return 0;
75         }
76     }
```

```
75     }
76 }
77 // 输出总的成本
78 cout << cost << endl;
79 return 0;
}
```

Java

```
1 import java.util.*;
2
3
4
5 public class Main {
6     // 并查集数组
7     static int[] parent;
8
9     // 并查集查找函数, 用于查找x所在的集合
10    static int find(int x) {
11        // 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)
12        if (parent[x] != x) {
13            parent[x] = find(parent[x]);
14        }
15        // 返回x的父节点
16        return parent[x];
17    }
18
19    // 定义边的类
20    static class Edge {
21        int u, v, cost, pre;
22
23        // 构造函数
24        public Edge(int u, int v, int cost, int pre) {
25            this.u = u; // 基站u
26            this.v = v; // 基站v
27            this.cost = cost; // 架设光纤的成本
28            this.pre = pre; // 是否已存在光纤连接
29        }
30    }
31    // 并查集合并函数, 用于合并x和y所在的集合
32}
```

```
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
}
```



```
73     for (int i = 2; i <= N; i++) {
74         // 如果有基站不在同一个集合中, 那么就输出-1并结束程序
75         if (find(i) != find(1)) {
76             System.out.println(-1);
77             return;
78         }
79     }
80     // 输出总的成本
81     System.out.println(cost);
82 }
}
```

JavaScript

```
1  const readline = require('readline');
2  const rl = readline.createInterface({
3      input: process.stdin,
4      output: process.stdout
5  });
6
7  let lines = [];
8  rl.on('line', (line) => {
9      lines.push(line);
10     if(lines.length === 2 + parseInt(lines[1])) {
11         rl.close();
12     }
13 });
14
15 rl.on('close', () => {
16     // 定义边的类
17     class Edge {
18         constructor(u, v, cost, pre) {
19             this.u = u; // 基站u
20             this.v = v; // 基站v
21             this.cost = cost; // 架设光纤的成本
22             this.pre = pre; // 是否已存在光纤连接
23         }
24     }
25
26     // 并查集查找函数, 用于查找x所在的集合
27     ~
```

```
27 function find(x) {
28     if (parent[x] !== x) {
29         // 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)
30         parent[x] = find(parent[x]);
31     }
32     return parent[x]; // 返回x的父节点
33 }
34
35 // 并查集合并函数, 用于合并x和y所在的集合
36 function union(x, y) {
37     let rootX = find(x); // 找到x的根节点
38     let rootY = find(y); // 找到y的根节点
39     if (rootX !== rootY) {
40         // 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
41         parent[rootX] = rootY;
42     }
43 }
44
45 let N = parseInt(lines[0]); // 输入基站个数
46 let M = parseInt(lines[1]); // 输入具备光纤直连条件的基站对的数目
47 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
48 let edges = []; // 存储所有的边
49
50 for (let i = 2; i < lines.length; i++) {
51     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
52     edges.push(new Edge(X, Y, Z, P)); // 添加边
53     if (P === 1) { // 如果已存在光纤连接, 那么就将X和Y合并
54         union(X, Y);
55     }
56 }
57
58 // 将所有的边按照成本从小到大排序
59 edges.sort((a, b) => a.cost - b.cost);
60 let cost = 0; // 总的成本
61
62 for (let edge of edges) {
63     // 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
64     if (find(edge.u) !== find(edge.v)) {
65         cost += edge.cost; // 累加成本
66         union(edge.u, edge.v); // 合并边的两个端点所在的集合
67     }
68 }
```

```
68     }
69 }
70
71 for (let i = 2; i <= N; i++) {
72     // 检查所有的基站是否都在同一个集合中
73     if (find(i) !== find(1)) {
74         // 如果有基站不在同一个集合中, 那么就输出-1并结束程序
75         console.log(-1);
76         return;
77     }
78 }
79
80 // 输出总的成本
81 console.log(cost);
});
```

Python

```
1 class Edge:
2     # 定义边的类
3     def __init__(self, u, v, cost, pre):
4         self.u = u # 基站u
5         self.v = v # 基站v
6         self.cost = cost # 架设光纤的成本
7         self.pre = pre # 是否已存在光纤连接
8
9     def find(x):
10        # 并查集查找函数, 用于查找x所在的集合
11        if parent[x] != x:
12            # 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)
13            parent[x] = find(parent[x])
14        return parent[x] # 返回x的父节点
15
16    def union(x, y):
17        # 并查集合并函数, 用于合并x和y所在的集合
18        rootX = find(x) # 找到x的根节点
19        rootY = find(y) # 找到y的根节点
20        if rootX != rootY:
21            # 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
22            parent[rootX] = rootY
23
```

```
23
24 if __name__ == "__main__":
25     N = int(input()) # 输入基站个数
26     M = int(input()) # 输入具备光纤直连条件的基站对的数目
27     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
28     edges = [] # 存储所有的边
29
30     for _ in range(M):
31         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
32         edges.append(Edge(X, Y, Z, P)) # 添加边
33         if P == 1: # 如果已存在光纤连接, 那么就将X和Y合并
34             union(X, Y)
35
36     # 将所有的边按照成本从小到大排序
37     edges.sort(key=lambda edge: edge.cost)
38     cost = 0 # 总的成本
39
40     for edge in edges:
41         # 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
42         if find(edge.u) != find(edge.v):
43             cost += edge.cost # 累加成本
44             union(edge.u, edge.v) # 合并边的两个端点所在的集合
45
46     for i in range(2, N + 1):
47         # 检查所有的基站是否都在同一个集合中
48         if find(i) != find(1):
49             # 如果有基站不在同一个集合中, 那么就输出-1并结束程序
50             print(-1)
51             break
52     else:
53         # 输出总的成本
54         print(cost)
```

文章目录

华为OD机考:统一考试 C卷 + D卷 + B卷 +A卷

题目描述

输入描述

输出描述

用例1

用例2

用例3

解题思路

C++

Java

JavaScript

Python

机考真题 华为OD



CSDN @算法大师