```java

/* * MST Minimum Spanning Tree (Basic Form)
import java.util.*;

class MST_Kruskal_Alg{
    static Node[] G;
    static int sM;
    static int N;
    static int cnt = 0;
    static int conCompCnt=0;
    static Stack<Integer> s = new Stack<Integer>();
    static List<Edge> E = new LinkedList<Edge>();
    static List<Edge> MST = new LinkedList<Edge>();

    static class DisjointSet{
        int[] pset;
        public DisjointSet(){
            pset = new int[G.length];
            for(int i = 0; i<pset.length; i++){
                pset[i] = i;
            }
        }
        public  int findSet(int i){
            return pset[i]==i ? pset[i] : (pset[i] =
findSet(pset[i]));
        }
        public  void unionSet(int i, int j){
            pset[findSet(i)]= findSet(j);
        }
        public  boolean isSameSet(int i, int j){
            return (findSet(i)==findSet(j));
        }
    }
    static class Node {
        List<Edge> adj;
        int n;
        public boolean visited;
        int layer;
        public Node(int N){
            adj = new ArrayList<Edge>();
            n=N;
            layer = -1;
            visited = false;
        }
    }
    static class Edge implements Comparable<Edge> {
        int to, weight, from;
        public Edge(int t, int w){
            to=t;
            weight = w;
        }
        public Edge(int f, int t, int w){
            to=t;
            weight = w;
            from = f;
        }
        @Override
        public int compareTo(Edge e)  {
            return this.weight - e.weight;
        }
    }
    public static void makeGraph(int n){
        G = new Node[n];
        for(int i =0; i<n; i++){
            G[i]=new Node(i);
        }
    }
    public static void addEdge(int u,int v, int w){
        G[u].adj.add(new Edge(u,v,w));
        E.add(new Edge(u,v,w));
        G[v].adj.add(new Edge(v,u,w));
    }
    public static int charN(char c){
        return c;
    }
    public static int MSTkruskal(DisjointSet ds){
        if(conComp(G)!=1){
            return -1;
        }
        Collections.sort(E);
```

```java
        int cost = 0;
        for(int i =0; i<E.size(); i++){
            Edge e = E.get(i);
            if(!(ds.isSameSet(e.from, e.to))){
                ds.unionSet(e.from, e.to);
                MST.add(e);
                cost+=e.weight;
            }
        }

        return cost;
    }
    public static void dfs(int n){
        if(G[n].visited){
            return;
        }
        G[n].visited = true;
        s.push(n);
        cnt++;
        for(Edge e : G[n].adj)
        {
            dfs(e.to);
        }
    }
    public static int conComp(Node[] g){
            for(int i = 0; i<g.length;i++){
                if(!(g[i].visited)){
                    conCompCnt++;
                    dfs(i);
                }
            }
        return conCompCnt;
    }

    public static void main(String[] args){
        Scanner scan = new Scanner(System.in);
        int K = Integer.parseInt(scan.nextLine());
        int u = -1;
        int v = -1;
        int w = -1;

        for(int k =0; k<K; k++){
            N = scan.nextInt();
            makeGraph(N);
            while((u = scan.nextInt()) !=-1){

                v = scan.nextInt();
                w = scan.nextInt();
                addEdge(u,v,w);

            }
            DisjointSet ds = new DisjointSet();
            System.out.println(MSTkruskal(ds));

        }
        scan.close();
    }
}

```