Kruskal (MST): Really Special Subtree

by pranav9413

Problem Submissions Leaderboard Discussions Editorial

Given an undirected weighted connected graph, it is required to find the Really Special SubTree in it. The Really Special SubTree is defined as a subgraph consisting of all the nodes in the graph and

- There is only one exclusive path from a node to every other node.
- The subgraph is of minimum overall weight (sum of all edges) among all such subgraphs.
- While creating the Really Special SubTree, start by picking the edge with smallest weight. If there are edges of equal weight available at an instant, then the edge to be chosen first among them is the one with minimum value of sum of the following expression :
  - u + wt + v , where u and v are the node numbers of the corresponding edge and wt is the weight.
- Even then if there is a collision, choose any one of them.
- While doing the above, ensure that no cycle is formed while picking up edges.

Finally, you need to print the overall weight of the Tree so formed using above rules.

Input Format

First line has two integers N, denoting the number of nodes in the graph and M, denoting the number of edges in the graph.

The next M lines each consist of three space separated integers $x\ y\ r$, where x and y denote the two nodes between which the **undirected** edge exists, r denotes the weight of edge between the corresponding nodes.

Constraints

- $2 \leq N \leq 3000$
- $1 \leq M \leq (N * (N-1))/2$
- $1 \leq x, y \leq N$
- $0 \leq r \leq 10^5$

Note: * If there are edges between the same pair of nodes with different weights, they are to be considered as is, like multiple edges.

Output Format

Print a single integer denoting the total weight (sum of weights of all edges in the MST) of the Really Special SubTree.

Sample Input 0

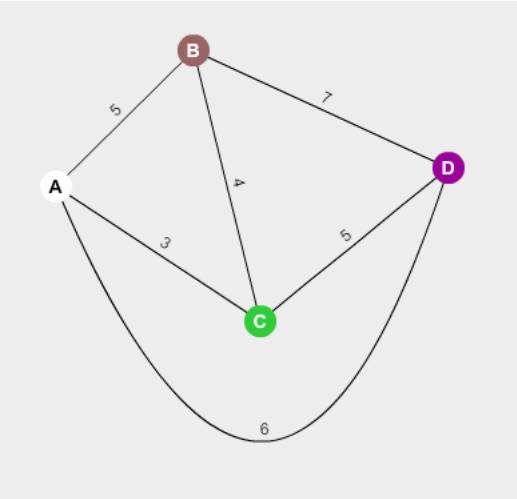
```
4 6
1 2 5
1 3 3
4 1 6
2 4 7
3 2 4
3 4 5
```

Sample Output 0

```
12
```

Explanation 0

The graph given in the test case is shown as :



- The nodes A,B,C and D denote the obvious 1,2,3 and 4 node numbers.
- The starting node is A or 1 (in the given test case)

Applying the Kruskal's algorithm, all the edges are sorted in ascending order of weight.

After sorting, the edge choices are available as :

A->C (WT. 3) , B->C (WT. 4) , A->B (WT. 5) , C->D (WT. 5) , A->D (WT. 6) and B->D (WT. 7)

Picking these edges and finalizing only if it doesnt create a cycle :

A->C : B->C

Now, when A->B edge is picked , it can be easily seen that they both belong to same set (form a cycle) and hence this edge is ignored.

The process continues and the following edge sequence is formed for the MST :

A->C : B->C : C->D

and Total weight of the hence formed Really Special SubTree is : **12**

Submissions: 7992
Max Score: 50
Difficulty: Medium

Rate This Challenge:

More

Current Buffer (saved locally, editable) Java 8

```
1 import java.io.*;
2 import java.util.*;
3
4 class DisjointSet{
5
6     long[] rank,parent;
7     int n;
```

```
 8
 9      public DisjointSet(int n){
10          this.n = n;
11          rank = new long[n];
12          parent = new long[n];
13          makeset(n);
14      }
15
16      void makeset(int n){
17          for(int i = 0 ; i < n ; i++){
18              parent[i] = (long) i;
19          }
20      }
21
22      long find(int x){
23
24          if(parent[x] != (long)x){
25              parent[x] = find((int)parent[x]);
26          }
27         return parent[x];
28      }
29
30      void union(int x, int y){
31
32          int xRoot = (int) find(x);
33          int yRoot = (int)  find(y);
34
35          if(xRoot == yRoot){
36              return;
37          }
38
39          if(rank[xRoot] < rank[yRoot]){
40              parent[xRoot] = yRoot;
41          }
42          else if(rank[xRoot] > rank[yRoot]){
43              parent[yRoot] = xRoot;
44          }
45          else{
46              parent[yRoot] = xRoot;
47              rank[xRoot] = rank[xRoot] + 1;
48          }
49      }
50  }
51
52
53  class Vertex{
54
55      private int v1,v2;
56
57      public Vertex(int v1, int v2){
58          this.v1 = v1;
59          this.v2 = v2;
60      }
61
62      public int getV1(){
63          return v1;
64      }
65
66      public int getV2(){
67          return v2;
68      }
69
70  }
71
72
73  public class Solution {
74
75      public static void main(String[] args) throws IOException{
76
77          BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
78          String line = br.readLine();
79          String[] numbers = line.split("\\s");
80
```

```
        int V = Integer.parseInt(numbers[0]);
        int E = Integer.parseInt(numbers[1]);

        DisjointSet disjoint = new DisjointSet(V);

        HashMap<Integer, LinkedList<Vertex>> adj = new HashMap<Integer, LinkedList<Vertex>>();;

        long output = 0;

        //List sortedKeys=new ArrayList(yourMap.keySet());
        //Collections.sort(sortedKeys);

        for(int i = 0 ; i < E ; i++){

            line = br.readLine();
            numbers = line.split("\\s");

            int v1 = Integer.parseInt(numbers[0]);
            int v2 = Integer.parseInt(numbers[1]);
            int weight = Integer.parseInt(numbers[2]);

            Vertex vertex = new Vertex((v1 - 1),(v2 - 1));

            if(adj.containsKey(weight)){
                //list2 = (LinkedList) list1.clone();

                LinkedList<Vertex> clone = (LinkedList) adj.get(weight).clone();
                clone.add(vertex);

                adj.put(weight,clone);
            }
            else{
                LinkedList<Vertex> link = new LinkedList<Vertex>();
                link.add(vertex);
                adj.put(weight,link);
            }

        }

        List sortedKeys=new ArrayList(adj.keySet());
        Collections.sort(sortedKeys);



        for(int g = 0 ; g < sortedKeys.size() ; g++){

            LinkedList<Vertex> clone = (LinkedList) adj.get(sortedKeys.get(g)).clone();

            Iterator itr = clone.listIterator();

            while(itr.hasNext()){

                Vertex vert = (Vertex) itr.next();
                int v1 = vert.getV1();
                int v2 = vert.getV2();

                long representativeV1 = disjoint.find(v1);
                long representativeV2 = disjoint.find(v2);

                if(representativeV1 != representativeV2){
                    disjoint.union((int)representativeV1, (int)representativeV2);
                    output = output + (Integer) sortedKeys.get(g);
                }


            }

        }

        System.out.println(output);

    }
}
```

Line: 1 Col: 1

Upload Code as File Test against custom input Run Code Submit Code