Kruskal (MST): Really Special Subtree

locked

Problem | Submissions | Leaderboard | Discussions

Submissions: 88
Max Score: 100
Difficulty: Medium

Rate This Challenge:

More

Given an undirected weighted connected graph, 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.
- No cycles are formed

To create the Really Special SubTree, always pick the edge with smallest weight. Determine if including it will create a cycle. If so, ignore the edge. If there are edges of equal weight available:

- Choose the edge that minimizes the sum $u + v + wt$ where u and v are vertices and wt is the edge weight.
- If there is still a collision, choose any of them.

Print the overall weight of the tree formed using the rules.

For example, given the following edges:

```
u       v       wt
1       2       2
2       3       3
3       1       5
```

First choose $1 \rightarrow 2$ at weight 2. Next choose $2 \rightarrow 3$ at weight 3. All nodes are connected without cycles for a total weight of $3 + 2 = 5$.

Function Description

Complete the $kruskals$ function in the editor below. It should return an integer that represents the total weight of the subtree formed.

kruskals has the following parameters:

- *g_nodes*: an integer that represents the number of nodes in the tree
- *g_from*: an array of integers that represent beginning edge node numbers
- *g_to*: an array of integers that represent ending edge node numbers
- *g_weight*: an array of integers that represent the weights of each edge

Input Format

The first line has two space-separated integers g_nodes and g_edges, the number of nodes and edges in the graph.

The next g_edges lines each consist of three space-separated integers g_from, g_to and g_weight, where g_from and g_to denote the two nodes between which the **undirected** edge exists and g_weight denotes the weight of that edge.

Constraints

- $2 \leq g\_nodes \leq 3000$
- $1 \leq g\_edges \leq \frac{N*(N-1)}{2}$
- $1 \leq g\_from, g\_to \leq N$
- $0 \leq g\_weight \leq 10^5$

**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 of the Really Special SubTree.

Current Buffer (saved locally, editable)

C++

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

  struct Pair{
    int index;
    int weight;
  };
  struct Node{
    list<Pair> neighbours;
    bool hasNeighbour(int index){
      for(auto it:neighbours){
        if(it.index==index){
          return true;
        }
      }
      return false;
    }
    void addNeighbour(int index,int weight){
      neighbours.push_back({index,weight});
    }
  };
  struct Edge{
    int from;
    int to;
    int weight;
    bool operator <(const Edge& right) const{
      return weight<right.weight;
    }
  };
  struct Graph{
    vector<Node> nodes;
    Graph(int nodeCount=0){
      nodes.resize(nodeCount);
    }
    void connect(int from, int to, int weight){

        nodes[from].addNeighbour(to,weight);
         nodes[to].addNeighbour(from,weight);

    }
    vector<Edge> getAllEdges(){
      vector<Edge> allEdges;
      for(int i=0;i<nodes.size();i++){
        for(auto it:nodes[i].neighbours){
          if(i<it.index){
            allEdges.push_back({i,it.index,it.weight});
          }
        }
      }
      return allEdges;
    }

    int kruskal(){
      int result=0;
      vector<Edge> allEdges=getAllEdges();
      sort(allEdges.begin(),allEdges.end());
      vector<int> components(nodes.size());
      for(int i=0;i<nodes.size();i++){
        components[i]=i;
      }
      for(auto Edge:allEdges){
        if(components[Edge.from]!=components[Edge.to]){
          result+=Edge.weight;
          int oldComp=components[Edge.from];
          int newComp=components[Edge.to];
          for(int i=0;i<components.size();i++){
            if(components[i]==oldComp){
              components[i]=newComp;
            }
          }
        }
      }
      return result;
    }
  };
  int main(){
    int nodeCount;
    int edgeCount;
    cin>>nodeCount;
    cin>>edgeCount;
    int from,to,weight;
    Graph g(nodeCount);
    for(int i=0;i<edgeCount;i++){
      cin>>from>>to>>weight;
      g.connect(from-1,to-1,weight);
    }


    cout<<g.kruskal();
  return 0;
}
```

Line: 1 Col: 1

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