Analysis and Design of Algorithms
CSE2ada Winter '19

Programming Assignment 2
Due Date : April 2, 11:59 PM

The assignment may be submitted in teams of two students. We shall be running your program on both toy examples as well as real life test cases. So make sure you test scalability of your code. Any kind of plagiarism, either between two different teams or from the internet or other sources will be strictly penalized according to the institute anti-plagiarism policies

In this assignment, you will be solving the classical Steiner Tree problem. Given a weighted graph $G(V, E)$ and a subset of vertices $R \subseteq V$, called terminals, the problem is to find the *minimum cost tree* that spans R. Note that, if $R = V$, this problem becomes our well-known minimum spanning tree problem. However, while MST admits blazingly fast algorithms, like Prim's, Kruskal's, Boruvka's etc., the Steiner Tree problem is NP-Hard, which loosely means : "It is unlikely that an algorithm exists which runs in polynomial time and produces an optimal solution to the Steiner Tree problem for all instances".

Given this state of affairs, we resort to heuristics which, although not giving optimal solutions, might actually not be too bad compared to an optimal solution. One such algorithm is the so-called **MST heuristic**. Please read about the algorithm from Section 2.1 in these lecture notes.(I had also described it in detail in the lectures)

Your task is to implement this heuristic in **java**. The input will be in STP format taken from DIMACS Implementation Challenge repo. Note that the format is much more general and can contain more information. For this assignment, we shall use undirected weighted graphs. So, the only relevant sections of the format are *Graph* & *Terminals*.

An example of the input is (SP/oddcycle3.stp) [only relevant sections]:

```
SECTION Graph
Nodes 6
Edges 9

E 1 2 1
...
E 5 6 1
END
SECTION Terminals
Terminals 3
T 1
T 3
T 5
END
```

Note that the edges are given as a 3-tuple: `E x y z`, denoting there is an undirected edge between vertices x & y, having an edge weight of z.

Your program must read the STP-format input from *stdin*, and print the result to *stdout*. The output format should be similar to the input stp format.

In addition, there should be an extra section at the end which outputs the total cost of the tree you constructed.
An example output for the input(SP/oddcycle3.stp) graph is:

```
SECTION Graph
Nodes 4
Edges 4

E 1 2 1
E 2 3 1
E 3 4 1
E 4 5 1
END
SECTION Terminals
T 1
T 3
T 5
END
SECTION Cost
4
END
```

Note: The number of nodes in output should be number of vertices included in the Steiner tree, which should necessarily contain the terminals in the input, and number of edges be the number of edges in your tree. SECTION Cost should contain a single line, consisting of a single integer, the cost of the graph, which should equal the sum of the edge weights you reported.

The test cases we shall use also come from the same source. Remember that, no matter what intermediate graphs you construct for your algorithm, the final output must only contain edges of the original graph. Your result will be accepted as correct if and only if it is **at most twice the actual optimal value and the output only contains edges from the original graph**. Your code must pass on at least 50% of the two test sets I080 and P4Z which you can find here.