CS 374 **Spring 2015**

Homework 8

Nikhil Unni (nunni2)

2. Pseudocode :

```
for a ∈ A do
    Remove a and all of its outgoing edges from the graph;
end
T ← ∅;
if The number of connected components of the remaining graph > 1 then
    return ∅;
end
else
    T ← MST(remaining graph);
end
for a ∈ A do
    cheapestEdge ← ∅;
    for e = (u, v) ∈ out(a) do
        if v ∉ A then
            if cheapestEdge = ∅ then
                cheapestEdge ← v
            end
            else if c(v) < cheapestEdge then
                cheapestEdge ← v
            end
        end
    end
    if cheapestEdge = ∅ then
        return ∅;
    end
    Attach a to the T via cheapestEdge;
end
return T;
```

Algorithm 1: A-leaf-respecting MST

The algorithm just removes all elements in A. If, right off the bat, this disconnects the graph, that means that at least one element in A is the bridge between two connected components, and there's no way to just have one outgoing edge coming from it, so it can never be a leaf in a spanning tree.

Then, we calculate the MST of the non-A graph, so that we get the minimum cost so far. Then we reconnect each element of A back to the MST by using its cheapest edge, again to minimize cost, making sure not to attach to any other element of A, as that would make that other element of A have two outgoing edges. If there's an element of A that only connects to other elements of A, there's no way they can all have leaves, so we return null.

From the explanation above, it's clear that the spanning tree generated by the algorithm has only elements of A as leaves. However, we can also prove that the algorithm generates the minimum cost A-leaf-respecting spanning tree. Calculating the MST of the remaining graph is for sure the minimum cost of the remaining graph.

But since the elements of A must be leaves in the spanning tree, they can only connect to the MST with one edge. So by selecting their minimum weight edge to connect with the MST, we are effectively minimizing the cost of the resultant A-leaf-respecting spanning tree.

The time complexity of the algorithm is O(V+E). We can use our linear time MST algorithm from problem 3, rather than Kruskal's. Then, we just iterate through each element of A as well as their outgoing edges. But since A is bounded by V and the sum of outgoing edges by the elements of A is bounded by E, this does not contribute to the running time.