COMS 311: Homework 7
Due: Oct 19, 11:59pm
Total Points: 60

Late submission policy. If you submit by Oct 20, 11:59PM, there will be 20% penalty. That is, if your score is x points, then your final score for this homework after the penalty will be $0.8 \times x$. Submission after Oct 20, 11:59PM will not be graded without explicit permission from the instructors.

Submission format. Your submission should be in pdf format. Name your submission file: <Your-net-id>-311-hw7.pdf. For instance, if your netid is asterix, then your submission file will be named asterix-311-hw7.pdf.

If you are planning to write your solution and scan+upload, please make sure the submission is readable (sometimes scanned versions are very difficult to read - poor scan quality to blame).

If you plan to snap pictures of your work and upload, please make sure the generated pdf is readable - in most cases, such submissions are difficult to read and more importantly, you may end up submitting parts of your work.

If you would like to type your work, then one of the options you can explore is latex (Overleaf).

Rules for Algorithm Design Problems.

1. Unless otherwise mentioned, for all algorithm design problems, part of the grade depends on runtime. Better runtime will lead to higher grade.
2. Unless otherwise mentioned, for all algorithm design problems, you are required to write some justification of why your algorithm correctly addresses the given problem.
3. You will write pseudo-code for your algorithm. It should not be some code in a specific programming language (e.g., Java, C/C++, Python).
4. You can use any operation already covered in class-lectures in your solution without explicitly writing the algorithm. However, you need to use them appropriately (i.e., indicate the inputs to and outputs from these algorithms). For instance, you can use heapifyUp with input as the index of a heap element and you do not need to write the code for heapifyUp.

Learning outcomes.

1. Single Source Shortest Paths
2. MST
3. Greedy Algorithm for Scheduling

Problems 1 – 3 are 20pts each. Extra Credit Problem is 20pt.

1. The chief engineer is in charge of deciding the mountainous road-network that will be kept open (cleared of debris and maintained regularly) during the winter months. Each road connects different small towns in the mountains, and all towns are connected to each other either directly or indirectly. Each road is associated with a value indicating the level of danger in maintaining that road during winter months. There are many subsets of roads such that the roads in such subsets keep the towns connected directly or indirectly. Each subset is assigned a cost of maintenance, which is directly proportional to the highest danger level of the road present in that subset. The engineer wants to select the smallest subset that keeps the towns connected directly or indirectly and that also has the lowest cost of maintenance. Develop an algorithm to find such a subset. Justify the correctness of your algorithm and derive its runtime.

set answer{
    Creates MST with Prim's algorithm.
    Return the set of edges.
}

MST is a tree which connects all the nodes(towns) with minimum values of weights.
Therefore, the MST will have the smallest subset that keeps the towns connected directly or indirectly.

Runtime:
Runtime of Rrim's is O((|E|+|V|)log(|V|))
Therefore, the runtime of this algorithm is also O((|E|+|V|)log(|V|))

2. You own a car repair business, and each morning you have a set of customers whose cars need to be fixed. You can fix only one car at a time. Customer i 's car needs r_i time to be fixed. Given a schedule (i.e., an ordering of n jobs), let f_i denote the finishing time of fixing customer i 's car. For example, if job j is the first to be done, we would have $f_j = r_j$, and if job i is done right after job j, we would have $f_i = f_j + r_i$. Each customer i also has a given value v_i that represents his or her importance to the business. The happiness of customer i is expected to be dependent on the finishing time of i 's job. So you have decided to schedule the jobs with the objective to minimize the weighted sum of the completion times, $\sum_{i=1}^{n} v_i f_i$. You have been told that to realize the optimization objective, the jobs should be scheduled in decreasing order of v_i / r_i. Prove/disprove that this scheduling strategy gives an optimal solution, i.e., minimizes $\sum_{i=1}^{n} v_i f_i$.

Claim: to realize the optimization objective, the jobs should be scheduled in decreasing order of v_i/r_i.

Assume the claim is false then $\exists$ i, j, k : $v_i/ r_i <= v_j/ r_j <= v_k/ r_k$ and they do not minimizes $\Sigma^n v_i f_i$ ….i

If j starts at time f_i, then k starts at f_i+r.
The time at which I complete fixing j is f_i+r_j and the product of the happiness and finishing time is $(f_i+r_j)v_j$.
The time at which I complete fixing k is $f_i+r_j+r_k$ and the product of the happiness and finishing time is $(f_i+r_j+r_k)v_k$.

Consider the schedule, where j and k are swapped.

If k starts at time f_i, then j starts at f_i+r_k.
The time at which I complete fixing k is f_i+r_k and the product of the happiness and finishing time is $(f_i+r_k)v_k$.
The time at which I complete fixing j is $f_i+r_k+r_j$ and the product of the happiness and finishing time is $(f_i+r_k+r_j)v_j$.

∵ i
when the order of j and k is swapped, it makes the sum smaller.
∴ $(f_i+r_j)v_j + (f_i+r_j+r_k)v_k >= (f_i+r_k)v_k + (f_i+r_k+r_j)v_j$
⇔ $f_iv_i+r_iv_i+f_iv_k+r_iv_k+r_kv_k >= f_iv_k+r_kv_k+f_iv_i+r_kv_i+r_iv_i$…ii

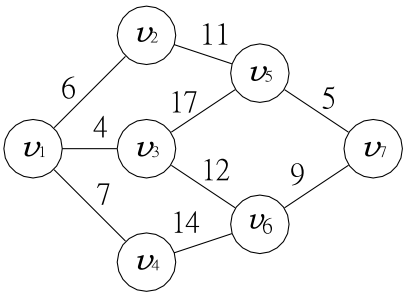
∵i
$v_ir_j <= v_jr_i$, $v_jr_k <= v_kr_j$, $v_kr_i <= v_ir_k$ ...iii

ii contradicts iii
=>The Claim is true.

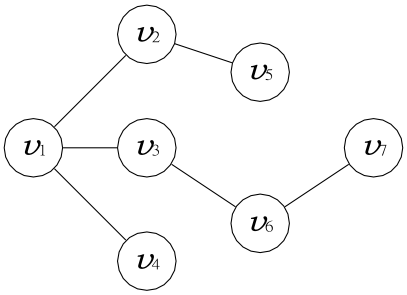
3. You are planning a long adventure trip on your newly bought Rivian R1S. Your trip will start at a town and end at some other town, and on the way you will pass through some towns (consider there are n towns including the start and destination towns). On a single charge, your Rivian R1S can go m miles. Every town has a charging station for your vehicle, and the distance between successive towns in your trip is $\leq m$. Your objective is to start your adventure with full charge and arrive at your destination with minimal number of charging-related stops. The strategy you are applying for realizing your objective is as follows: drive to the farthest town possible on the available charge and then make a charging stop at that town. Prove that this strategy will indeed minimize your charging related stops.

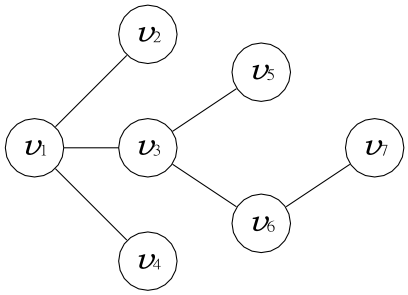
The method selects the furthest town from a town within m miles from the town. No optimal method can select a father town because then the car would run out the gas. Therefore, any other optimal solutions will select the same town or closer town. In that scenario, the claimed method will do the same performance or better than other method. Similarly, we can apply the same thing for any subsequent stop
$\therefore$ this strategy will indeed minimize your charging related stops.

4. **Extra Credit.** Given a weighted graph (with positive weights) and a tree rooted at a vertex s_0, write an algorithm to verify that the shortest paths to all vertices from s_0 are present in the tree.



For instance, if we are given the above graph and some tree (see couple of examples below) that is rooted at v_3, our algorithm should return true if the tree contains all the shortest paths from v_3; false otherwise. Note that the tree does not contain the edge weights, it just shows the edges that form the tree.





Justify the correctness of your algorithm and derive the runtime of your algorithm.

Boolean Answer(originalTree, s_0, Tree){
    Compute Dijkstra's algorithm from s_0 and obtain SPT(s).
    For each edge and vertex in SPT(s) and Tree compare if they are identical or not.
    If they are SPT(s) and Tree are identical then return true.
    Otherwise return false
}

By computing Dijkstra's algorithm from, we can have SPT(s) from s_0 which has the shortest paths to all vertices form s_0.
Then compare if the given tree is identical to SPT(s) to see if the shortest paths to all vertices form s_0 is present in tree.

Runtime:
Dijkstra's algorithm takes O((|E|+|V|)log(|V|))
To check if SPT(s) and Tree is identical, it takes O(|E|+|V|)
Therefore, the runtime of this algorithm is also O((|E|+|V|)log(|V|))