HW 10

Due on **Monday, April 20, 2020 at 11pm**

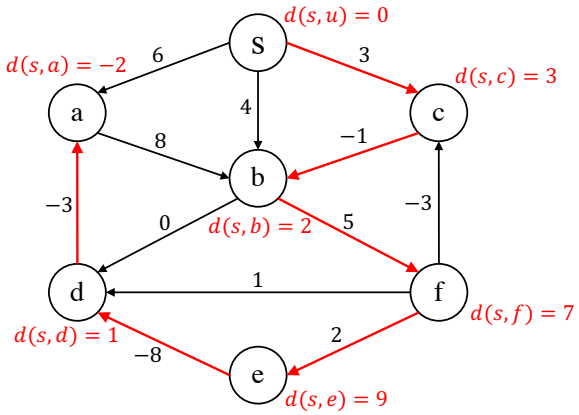
CS/ECE 374 B: Algorithms & Models of Computation, Spring 2020 Version: **1.1**

Submission instructions as in previous homeworks.

28 (100 PTS.) Changing the Weights

Let $G = (V, E)$ be a directed graph with edge lengths that can be negative. Let $\ell(e)$ denote the length of edge $e \in E$ and assume it is an integer, $\ell(e) \in \mathbb{Z}$. Assume you have found shortest path tree T rooted at a source node s that contains all the nodes in V. You also have the distance values $d(s, u)$ for each $u \in V$ in an array (thus, you can access the distance from s to u in $O(1)$ time). Note that the existence of T implies that G does not have a negative length cycle.

For example, consider the graph below, T is shown using red edges and the minimum distance $d(s, u)$ is shown next to each node.



28.A. (20 PTS.) Let $e = (p, q)$ be an edge of G that is *not* in T. Given e, show how to compute in $O(1)$ time the smallest integer amount by which we can decrease $\ell(e)$ before T is not a valid shortest path tree in G. Briefly justify the correctness of your solution. For example, in the graph above it is enough to decrease (b, d) by -2 for T to become invalid.

28.B. (80 PTS.) Let $e = (p, q)$ be an edge in the tree T. Given e, show how to compute in $O(m+n)$ time the smallest integer amount by which we can increase $\ell(e)$ such that T is no longer a valid shortest path tree. Your algorithm should output ∞ if no amount of increase will change the shortest path tree. Briefly justify the correctness of your solution. For example, in the graph above it is enough to increase (d, a) by 9 for T to become invalid.

29 (100 PTS.) 5G Cellular Deployments II

We have seen or will see in midterm 2 a question on 5G small cellular deployments. In that question, our goal was to minimize the cost of the deployment without taking into account whether it ensures coverage to all customers. In this question, we will try to ensure coverage without caring much for the cost. (Note that you do not need to have solved the midterm to solve this question nor does solving this question helps you on the midterm.)

Suppose there are n customers living on Green street which is a perfectly straight street. The ith customer lives at distance x_i meters from the beginning of the street (i.e., you are given n

numbers: $0 \leq x_1 < x_2 < \cdots < x_n$). `GlobalCell` is planning to connect all of these customers together small cell 5G base stations. A base station, which can be placed anywhere along Green street, can serve all the customers in distance r from it.

The input is $x_1, x_2, \ldots, x_n, r$. Describe a greedy efficient algorithm, as fast as possible, that computes the *smallest* number of base stations that can serve all the n customers. Namely, every one of the n customers must be in distance $\leq r$ from some base station that your algorithm decided to build.

As with all greedy algorithms, you must always prove the correctness and running time of your algorithm.

30 (100 PTS.) Minimum Spanning Tree

30.A. Consider the following "local-search" algorithm for MST. It starts with an arbitrary spanning tree T of G. Suppose $e = (u, v)$ is an edge in G that is not in T. It checks if it can add e to T and remove an edge e' on the unique path $p_T(u, v)$ from u to v in T such that tree $T' = T - e' + e$ is cheaper than T. If T' is cheaper then it replaces T by T' and repeats. Assuming all edge weights are integers one can see that the algorithm will terminate with a "local-optimum" T which means it cannot be improved further by these single-edge "swaps". Assuming all edge weghts are distinct prove that a local-optimum tree is an MST. Note that you are not concerned with the running time here.

30.B. We saw in lecture that Borouvka's algorithm for MST can be implemented in $O(m \log n)$ time where m is the number of edges and n is the number of nodes. We also saw that Prim's algorithm can be implemented in $O(m + n \log n)$ time. Obtain an algorithm for MST with running time $O(m \log \log n)$ by running Borouvka's algorithm for some number of steps and then switching to Prim's algorithm. This algorithm is better than either of the algorithms when $m = \Theta(n)$. Formalize the algorithm, specify the parameters, argue carefully about the implementation and analyze the running time details. No proof of correctness required but your algorithm should be clear.