CS4102 - ALGORITHMS - S'18 — NATE BRUNELLE
DUE SATURDAY APRIL 21, 2018 AT 11PM VIA COLLAB

H7.HR2EE

Collaboration Policy You are encouraged to collaborate with up to 3 other students, but all work submitted must be your own independently written solution. List the names of all of your collaborators. Do not seek published solutions for any assignments. If you use any published resources when completing this assignment, be sure to cite them. Do not submit a solution that you are unable to explain orally to a member of the course staff.

Collaborators Sujin Park

Sources *Introduction to Algorithms 3rd edition* Cormen et al., *The Restaurant at the end of the universe* Adams, https://simple.wikipedia.org/wiki/Al-Khwarizmi

PROBLEM 1 *Minimax Spanning Tree*

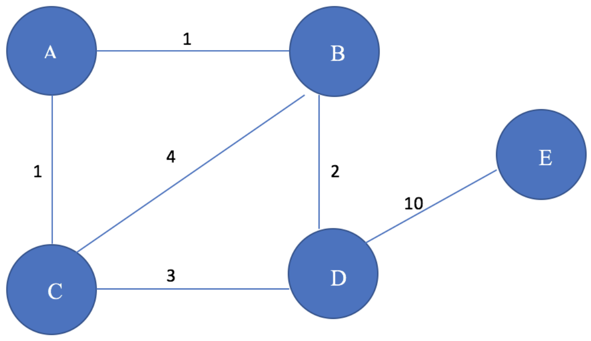
Let $G = (V, E)$ be a connected graph with distinct positive edge weights, and let T be some spanning tree of G (not necessarily a minimum spanning tree). The dominant edge of T is the edge with the greatest weight. A spanning tree is said to be a minimax spanning tree if there is no other spanning tree with a lower-weight dominant edge. In other words, a minimax tree minimizes the weight of the heaviest edge (instead of minimizing the overall sum of edge weights).

1. Is every minimum spanning tree of G also a minimax spanning tree of G?

   Proof. For the sake of contradiction proof let us assume that there is a minimum spanning tree of G is not a minimax spanning tree of G. Let us say that the minimum spanning tree is L and the minimax spanning tree of G is M. We can state that the heaviest edge of M (u, v) is lesser than the heaviest edge of L based on the definition of minimax spanning tree. We can also say that the heaviest edge for M is lesser than all the edges in L since the heaviest edge for M is less than the heaviest of L. Suppose there is cut in the graph, cut$(S, V - S)$ where S is the nodes that reachable from u by the minimum spanning tree M without crossing (u, v). Let us recall that L doesn't use (u, v), which means that there must be another point in G that crosses the cut with a lesser weight which we can call (a, b). Therefore we can theoretically make another spanning tree with swapping out u, v with a, b, which will lower the cost of the minimum spanning tree. This new tree will still have $V - 1$edges, and this tree will have the sum of the edges less than M, which contradicts are assumption. □

2. Is every minimax spanning tree of G also a minimum spanning tree of G?

(a)



As you can see a minimax spanning tree is 1-4-3-10, but the minimum spanning tree can also be 1-4-2-10. These two trees are not the same so this counter examples proves that the statement is not correct.

PROBLEM 2 *Oenology*

If you've ever been wine tasting, perhaps you're aware of the bizarre haute tasting notes given for various wines: "peppery with a hint of red fruit", "pineapple with a backdrop of tobacco", "bright with graphite minerality". Many wine sommeliers claim the ability to distinguish any vintage with superb accuracy. When subjected to scientific rigor, however, these claims rarely hold up. We wish to test the quality of a wine sommelier by having him participate in the following experiment.

We will have n samples of wine $s_1, ..., s_n$ each of which are either vintage 1985 or 1986. The sommelier will be given all possible pairs of samples (s_i, s_j). When given these samples, the sommelier will decide whether the samples are (a) both from the same vintage, (b) both from different vintages, (c) or can't decide. (All pairs will be tested, but not all pairs will have "same" or "different" decisions).

At the end of the tasting, suppose the sommelier claimed that m of the pairs were either "same" or "different". Give an algorithm that takes these m decisions and determines whether they are consistent. The decisions are consistent if there is a way to label each sample s_i with either "1985" or "1986" such that every time the sommelier decided the pair (s_i, s_j) was "same" our labels of s_i, s_j matched, and very time the sommelier decided the pair (s_i, s_j) was "different" our labels of s_i, s_j differed. Your algorithm should run in time $O(m + n)$.

1. We can first set up a graph with the nodes being the samples. If there is a decision between the samples create an edge of (s_i, s_j). Going through the graph check if there is a edge of (a, b), if (b, a) already exists. If it doesn't exist make that edge. Check whether the decisions for (a, b) is the same as (b, a). If they are not the same output, then the decisions are not consistent! For all the nodes on the graph, perform a breadth first search starting a a certain point and continuing until you have reached all nodes of the graph. When doing a breadth first search at a starting node give that node a value of 0. As doing the BFS when inspecting (s, y) if the edge is the same and y is not labeled then label it the same as s, which is 0. If it is different than label it 1 so we can tell the difference. Go through the graph once again and if the edge with same has different labels or if the edge is different and the labels for the nodes are the same we know that this is inconsistent. The running time will be $O(m + n)$ since the BFS of starting at a certain point until all the points

are looked at the labeling the nodes will take $O(m + n)$ all the other procedures will take $O(n)$ or $O(m)$.

PROBLEM 3 *Counting Shortest Paths*

Given a graph $G = (V, E)$, and a starting node s, let $\delta(s, v)$ be the length of the shortest path in terms of number of edges between s and v. Design an algorithm that computes the number of distinct paths from s to v that have length exactly $\delta(s, v)$.

1. We can develop an algorithm based on that there exists a node a that is between s and v. We could count the number of distinct paths from s to a and then from a to v. For the algorithm we are saying that a are the nodes neighboring v.
   Let us set a variable n_v for the number of shortest paths to v from s for all nodes in V. We then perform a BFS on G and for nodes of $\delta(s, v) = 1$, we set $n_v = 1$ since this means that there is only one edge between s and v. After looking for this case we can consider where $\delta(s, v) = 2, 3....V$. For node of v look for the $\delta(s, a) = 1, 2, ...V - 1$ and $\delta(a, v) = 1$. We can say this because a is a neighboring node of v.
   The overall runtime of the algorithm is $O(V + E)$since the BFS takes $O(V + E)$ and the summation also takes that time, which means the overall run time is $O(V + E)$