EE360C: Algorithms
University of Texas at Austin — Homework #6
Dr. Christine Julien — Due: March 8, 2018 (*in class quiz*)

Homework #6

You should try to solve these problems by yourself. I recommend that you start early and get help in office hours if needed. If you find it helpful to discuss problems with other students, go for it.

Problem 1: Minimum Spanning Trees

Let us say that a graph $G = (V, E)$ is a *near-tree* if it is connected and has at most $n + 8$ edges, where $n = |V|$. Give an algorithm with running time $O(n)$ that takes a near-tree G with costs on its edges and returns a minimum spanning tree of G. You may assume that all of the edge costs are distinct.

Solution

We run breadth first search to compute *a* spanning tree T (perhaps not a minimum one) of G. Then we examine the (up to 9) edges that are *not* included in T in turn. By definition, each such edge (u, v) forms a cycle with the path u-v in T. Find the cycle (i.e., go up in the tree from each of u and v until you find a comment ancestor, x; the cycle is formed by combining u-x, x-v, and (u, v); x could also be either u or v. Find the "heaviest" edge in this cycle; call it (s, t). If that edge is *not* (u, v), replace (s, t) in T with (u, v). Again, do this for all (up to 9) edges that were not originally included in T.

This algorithm has $O(n)$ running time. First, observe that the graph has $O(n)$ vertices (by definition of n) and $O(n)$ edges, since $|E| <= |V| + 8$. So all of our algorithms that are $O(|V| + |E|)$ time are also $O(n)$. BFS is $O(n)$. We run it once. Finding a cycle is $O(n)$ because it examines at most $O(n)$ edges to find the common ancestor of u and v. We do this at most 9 times, and 9 is a constant. So the overall running time is $O(n)$.

Problem 2: Bottleneck Edges in Minimum Spanning Trees

One of the basic motivations behind the Minimum Spanning Tree Problem is the goal of designing a spanning network for a set of nodes with minimum *total* cost. Here we explore another type of objective: designing a spanning network for which the *most expensive* edge is as cheap as possible.

Specifically, let $G = (V, E)$ be a connected graph with n vertices, m edges, and positive edge costs that you may assume are all distinct. Let $T = (V, E')$ be a spanning tree of G; we define the *bottleneck edge* of T to be the edge of T with the greatest cost.

A spanning tree T of G is a *minimum-bottleneck spanning tree* if there is no spanning tree T' of G with a cheaper bottleneck edge.

(a) Is every minimum bottleneck tree of G a minimum spanning tree of G? Prove or give a counter example.

Solution

This is false. Let G have vertices $\{v_1, v_2, v_3, v_4\}$, with edges (v_1, v_4) of weight 2, (v_1, v_2) of weight 3, (v_2, v_4) of weight 5, and (v_2, v_3) of weight 6. We see that the minimum spanning tree is the path $v_4 \to v_1 \to v_2 \to v_3$ with combined weight $2 + 3 + 6 = 11$. Notice that the path $v_1 \to v_4 \to v_2 \to v_3$ is a minimum bottleneck tree with bottleneck edge of weight 6. However, it has a combined weight of $2 + 5 + 6 = 13 > 11$. Hence, it is not a minimum spanning tree.

(b) Is every minimum spanning tree of G a minimum bottleneck tree of G? Prove or give a counter example.

Solution

This is true. Suppose that T is a minimum spanning tree of G and T' is a spanning tree with a lighter bottlenexk edge. Thus T contains an edge e that is heavier than every edge in T'. So if we add e to T', it forms a cycle C on which it is the heaviest edge (since all other edges in C belong to T'. By the cut property, then e does not belong to any minimum spanning tree, contradicting the fact that it is in T and T is a minimum spanning tree.

Problem 3: Modifying Dijkstra's Algorithm

We are given an directed graph $G = (V, E)$ on which each edge $(u, v) \in E$ has an associated value $r(u, v)$, which is a real number in the range $0 \leq r(u, v) \leq 1$ that represents the reliability of a communication channel from vertex u to vertex v. We interpret $r(u, v)$ as the probability that the channel from u to v will not fail, and we assume that these probabilities are independent. Give an efficient algorithm to find the most reliable path between two given vertices.

Solution

We create a weight function for the graph where the weight on each edge is the negative log of the reliability, i.e., $w(u, v) = -\log(r(u, v))$. Then we run Dijkstra's algorithm. Because the probabilities of reliability are independent of each other, the probability that a path will not fail is the product of the probabilities that its edges will not fail. We therefore want to maximize $\Pi_{(u,v)\in p} r(u, v)$. Maximizing this value is equivalent to maximizing $\log(\Pi_{(u,v)\in p} r(u, v)) = \sum_{(u,v)\in p} \log(r(u, v))$. (To be careful, we define $\log 0 = -\infty$).

Problem 4: Shortest Paths

Is the path between a pair of vertices in a minimum spanning tree necessarily a shortest path between the two vertices in the full graph? Give a proof or a counter example.

Solution

No. Consider the graph with three vertices v_1, v_2, v_3, where every vertex is connected to the other two with edge weights all equal to 1. Thus the shortest path between any pair of nodes is 1, but the tree will create a path between two of the nodes that is of cost 2.