COMP 3711H Design and Analysis of Algorithms
2015 Fall Semester
Homework 3
Handed out: Nov 2
Due: Nov 13

Assignments should be submitted to the collection box located outside room 4210A in the lab area.

Problem 1. There's a natural intuition that two nodes that are far apart in a communication network—separated by many hops—have a more tenuous connection than two nodes that are close together. There are a number of algorithmic results that are based to some extent on different ways of making this notion precise. Here's one that involves the susceptibility of paths to the deletion of nodes.

Suppose that an n-node undirected graph $G = (V, E)$ contains two nodes s and t such that the distance between s and t is strictly greater than $n/2$. Show that there must exist some node v, not equal to either s and t, such that deleting v from G destroys all s-t paths. (In other words, the graph obtained from G by deleting v contains no path from s to t.) Give an algorithm with running time $O(m + n)$ to find such a node v.

Problem 2. You are given a connected undirected graph $G = (V, E)$ with the property that $|E| = |V|$. First, prove that it is always possible to assign directions to the edges so that each vertex has exactly one incoming edge. Then design an $O(|V|)$ time algorithm to orient the edges so that each vertex has exactly one incoming edge. (*Hint:* Use DFS.)

Problem 3. Recall that the length of a path in a directed graph (with no edge weights) is the number of edges on the path.

(a) Design an algorithm that computes the length of the longest simple path in a DAG $G = (V, E)$. Your algorithm should run in $O(|V| + |E|)$ time. (*Hint:* Use DFS.)

(b) Can you generalize your algorithm to compute the length of the longest simple path in arbitrary directed graphs (i.e. possibly containing cycles) in $O(|V| + |E|)$ time? Either do so, or explain why your proof of correctness does not go through.

Problem 4. Let $G = (V, E)$ be a connected undirected graph with weights on the edges. The *bottleneck weight* of any spanning tree T of G is defined to be the maximum weight of an edge in T. Prove that the minimum spanning tree (MST) minimizes the bottleneck weight over all spanning trees. (*Note:* You should prove this from first principles; i.e., do not assume the correctness of the MST Lemma or the correctness of the MST algorithms taught in class.)

Problem 5. Suppose that we are given a cable network of n sites connected by communication channels. A communication channel supports signal transmission in both directions. Unfortunately, the communication channels are not perfect. A channel c_i may fail with certain probability p_i, where $0 < p_i < 1$. The probabilities of failure may differ for different channels and they are mutually independent events. One of the n sites is the central station and your problem is to compute the most reliable paths from the central station to all other sites.

(a) Let P be a path consisting of channels $c_1, c_2, \cdots, c_k$. The failure probability of P is the probability that a message cannot be communicated through P. Similarly, the success probability of P is the probability that a message can be communicated through P? What are the failure and success probabilities of P in terms of p_i, $1 \leq i \leq k$?

(b) We define the *reliability* of P by taking the logarithm of the success probability of P, where the base of the logarithm is 2. How would you model our problem as a graph problem? Justify your model.

(c) Propose an efficient algorithm for finding the most reliable paths from the central station to all other sites. What is the running time of your algorithm in terms of the number of sites and the number of channels in the network? (Answer part (c) in at most four or five lines.)

Problem 6.

(a) Recall that in each step of Huffman's algorithm, we merge two trees with the lowest frequencies. Show that the frequencies of the two trees merged in the ith step are at least as large as the frequencies of the trees merged in any previous step.

(b) Suppose that you are given the n input characters, already sorted according to their frequencies. Show how you can now construct the Huffman code in $O(n)$ time. (*Hint:* You need to make clever use of the property given in part (a). Instead of using a priority queue, you will find it advantageous to use a simpler data structure.)