ANLY550 Homework 2 Out: Feb. 11, 2018
Due: Feb. 26, 2018

For all homework problems where you are asked to give an algorithm, you must prove the correctness of your algorithm and establish the best upper bound that you can give for the running time. You should always write a clear informal description of your algorithm in English. You may also write pseudocode if you feel your informal explanation requires more precision and detail. As always, try to make your answers as clear and concise as possible.

1. Recall that when running depth first search on a directed graph, we classified edges into 4 categories: tree edges, forward edges, back edges, and cross edges (refer to Mitzenmacher's lecture notes at `http://people.cs.georgetown.edu/jthaler/ANLY550/lec3.pdf` for the definitions).

   Prove that if a graph G is undirected, then any depth first search of G will never encounter a cross edge.

2. Explain how to solve the following two problems using heaps. (No credit if you're not using heaps!) First, give an $O(n \log k)$ algorithm to merge k sorted lists with n total elements into one sorted list. Second, say that a list of numbers is k-close to sorted if each number in the list is less than k positions from its actual place in the sorted order. (Hence, a list that is 1-close to sorted is actually sorted.) Give an $O(n \log k)$ algorithm for sorting a list of n numbers that is k-close to sorted.

3. Design an efficient algorithm to find the *longest* path in a directed acyclic graph. (Partial credit will be given for a solution where each edge has weight 1; full credit for solutions that handle general real-valued weights on the edges, including *negative* values.)

4. In the shortest-path algorithm we are concerned with the *total length* of the path between a source and every other node. Suppose instead that we are concerned with the length of the *longest edge* between the source and every node. That is, the *bottleneck* of a path is defined to be the length of the longest edge in the path. Design an efficient algorithm to solve the single source smallest bottleneck problem; i.e. find the paths from a source to every other node such that each path has the smallest possible bottleneck.

5. Consider the shortest paths problem in the special case where all edge costs are non-negative integers. Describe a modification of Dijkstra's algorithm that works in time $O(|E| + |V| \cdot M)$, where M is the maximum cost of any edge in the graph.

6. The *risk-free currency exchange problem* offers a risk-free way to make money. Suppose we have currencies $c_1, \ldots, c_n$. (For example, c_1 might be dollars, c_2 rubles, c_3 yen, etc.) For every two currencies c_i and c_j there is an exchange rate $r_{i,j}$ such that you can exchange one unit of c_i for $r_{i,j}$ units of c_j. Note that if $r_{i,j} \cdot r_{j,i} > 1$, then you can make money simply by trading units of currency i into units of currency j and back again. This almost never happens, but occasionally (because the updates for exchange rates do not happen quickly enough) for very short periods of time exchange traders can find a sequence of trades that can make risk-free money. That is, if there is a sequence of currencies $c_{i_1}, c_{i_2}, \ldots, c_{i_k}$ such that $r_{i_1,i_2} \cdot r_{i_2,i_3} \ldots \cdot r_{i_{k-1},i_k} \cdot r_{i_k,i_1} > 1$, then trading one unit of c_{i_1} into c_{i_2} and trading that into c_{i_3} and so on will yield a profit.

   Design an efficient algorithm to detect if a risk-free currency exchange exists. (You need not actually find it.)