COMP 3711 Design and Analysis of Algorithms
Spring 2015
Assignment 4

1. This problem is a simplified version of the "Spread of Influence" problem studied by Kempe, Kleinberg, and Tardos. We use a directed graph $G = (V, E)$ to model a social network, where each vertex represents an individual. Each vertex v is associated with a threshold $t(v)$, and each edge (u, v) has a weight $w(u, v)$, which means that u has an influence of $w(u, v)$ on v. We initially set a given vertex r to be *active* (say, adopting a new product). Then r starts to spread his/her influence to its neighbors. When the total influence a vertex v receives from all its incoming neighbors is greater than or equal to its threshold $t(v)$, v also becomes active. Your job is to design an algorithm that, given an starting vertex r, computes the number of vertices that are eventually active. For full credits, your algorithm should run in $O(V + E)$ time. All edge weights and thresholds are nonnegative.

   Please check out

   `http://course.cse.ust.hk/comp3711/homework/Spread-of-Influence.ppt`

   for some background and an example.

2. Let G be a connected undirected graph with weights on the edges. Assume that all the edge weights are distinct. Let e_i be the edge with the i-th smallest weight. Does the MST have to contain e_1? How about e_2 and e_3? If yes, give a proof; if no, give a counter example. You must prove your results from first principles, i.e., you cannot rely on the cut lemma or the correctness of Prim's or Kruskal's algorithm.

3. Let G be a connected undirected graph with distinct weights on the edges. Given an edge e of G, can you decide whether e belongs to the MST in $O(E)$ time? If you compute the MST and then check whether e belongs to the MST, this would take $O(E \log V)$ time. To design a faster algorithm, you will need the following theorem:

   Edge $e = (u, v)$ does not belong to the MST if and only if there is a path from u to v that consists of only edges cheaper than e.

   Prove this theorem (you can use the cut lemma and the cycle property in the tutorial). Then give the $O(E)$-time algorithm.

4. The longest path problem introduced in the lecture is somewhat unnatural to model jobs and the dependencies. In a more natural structure, vertices would represent jobs and edges would represent dependencies; that is, edge (u, v) would indicate that job u must be performed before job v. We would then assign weights to vertices, not edges. Let $w(v)$ be the weight of vertex u. Give an algorithm to find the longest path from a source vertex s to a destination vertex t, where the weight of a path is the sum of the weights of its vertices.