600.363 Introduction to Algorithms / 600.463 Algorithms I **Fall 2014**
Homework #6 **Due:** October 21, 2014, 1:30pm

Please start each problem on a new page, and include your name on each problem. You can submit on blackboard, under student assessment.

Remember: you may work in groups of up to three people, but must write up your solution entirely on your own. Collaboration is limited to discussing the problems – you may not look at, compare, reuse, etc. any text from anyone else in the class. Please include your list of collaborators on the first page of your submission. You may use the internet to look up formulas, definitions, etc., but may not simply look up the answers online.

Please include proofs with all of your answers, unless stated otherwise.

1 Max Reachable Node (33 points)

Let G be a directed graph represented by an adjacency list. Suppose each node u has a weight w_u, which might be different for each node. Give an algorithm that computes, for every node, the maximum weight that is reachable from that node. So, for example, if G is strongly connected then every node can reach every other node, so for every node the maximum reachable weight is the same (the largest weight in the graph). Slightly more formally, for each vertex u let $R(u)$ denote the vertices reachable from u. Then when your algorithm is run on G, it should return an array of values where the value for node u is $\max_{v \in R(u)} w(v)$.

Your algorithm should run in $O(m + n)$ time. Prove correctness and running time.

2 Robot Planning (33 points)

Suppose that you have two robots, which each start out at different locations and want to get to different locations. At any time, one of them can move (but not both). However, if they get too close together their transmitters interfere, and they become stuck. Your goal is to design an algorithm that finds the shortest possible schedule that gets both robots to their destination, or detect that no such schedule exists.

More formally, you are given an unweighted graph $G = (V, E)$ and four special nodes $a, b, c, d \in V$. One robot starts at a and wants to get to c, and the other robot starts at b and wants to get to d. A schedule of length T specifies, for each times step in $\{1, \ldots, T\}$, which robot should move and to where (note that robots can only move along a single edge at a time, so if at time t your schedule says that the first robot should move to node u then it had better be the case that the robot was at a node adjacent to u after time $t - 1$). A schedule is non-interfering if the two robots are never at vertices that are distance at most r from each other in G, i.e. at every time point the two robots are at distance more than r from each other (where distance is the normal unweighted distance in G).

Design an algorithm that runs in time $O(n^3)$ to find a non-interfering schedule, or detect that no such schedule exists.

3 Arbitrage (33 points)

One way to make money in currency markets is to perform *arbitrage*. An arbitrage opportunity let's us convert money between currencies in such a way that in the end, we have more money than we started with. For example, suppose that 1 US dollar buys 49 Indian rupees, 1 rupee buys 2 Japanese yen, and 1 yen buys 0.0107 US dollars. Then if we start with 1 US dollar, we can perform a series of currency conversions so that we end up with $1 \cdot 49 \cdot 2 \cot 0.0107 = 1.0486$ US dollars. We made about 5 cents without doing anything other than moving between currencies!

Because arbitrage opportunities are essentially free money, it is an extremely interesting problem to find such opportunities. In this problem you will design an algorithm to do so. Suppose that there are n currencies $x_1, c_2, \ldots, c_n$, and you are given a matrix A so that the entry $A[i, j]$ is how many units of c_j you can buy with one unit of c_i. Give a polynomial-time algorithm to detect whether there is an arbitrage opportunity, i.e. detect whether there is a sequence of currencies $(c_{i_1}, c_{i_2}, \ldots, c_{i_k})$ such that

$$A[i_1, i_2] \cdot A[i_2, i_3] \ldots A[i_{k-1}, i_k] \cdot A[i_k, i_1] > 1.$$

Note: there are at least two different ways of solving this problem based on ideas from class. Either one is acceptable.