CS515: Algorithms and Data Structures, Fall 2015

Homework 2*

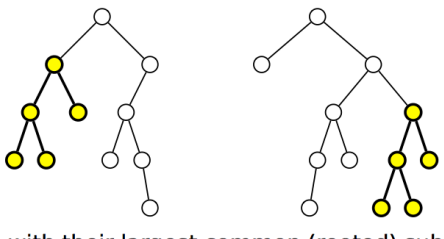
Due: Mon, Oct/26/15

Homework Policy:

1. Students should work on homework assignments in groups of preferably three people. Each group submits to TEACH one set of typeset solutions, and hands in a printed hard copy in class. The hard copy will be graded.
2. You are allowed to discuss the homework with other groups, however, you must mention their names in your submission. Also, you must cite any other source that you use.
3. The goal of the homework assignments is for you to learn solving algorithmic problems. So, I recommend spending sufficient time thinking about problems individually before discussing them with your friends.
4. *I don't know policy:* you may write "I don't know" *and nothing else* to answer a question and receive 25 percent of the total points for that problem whereas a completely wrong answer will receive zero.
5. Algorithms should be explained in plain english. Of course, you can use pseudocodes if it helps your explanation, but the grader will not try to understand a complicated pseudocode.
6. **Homework assignments must be typeset.**
7. More items might be added to this list. ☺

Problem 1. **[25 pts]** This question asks you to find efficient algorithms to compute the largest common rooted subtree of two given rooted trees. A rooted subtree consists of an arbitrary node and all its descendants. However, the precise definition of ?common? depends on which rooted trees we consider to be isomorphic.

(a) An ordered tree is either empty or a node with a *sequence* of children, which are themselves the roots of (possibly empty) ordered trees. Two ordered trees are isomorphic if they are both empty, or if their ith subtrees are isomorphic for all i. Describe an algorithm to find the largest common ordered subtree of two ordered trees T_1 and T_2.



Two binary trees, with their largest common (rooted) subtree emphasized

*Problems are from Jeff Erickson's lecture notes. Looking into similar problems from his lecture notes on dynamic programming, greedy algorithms and randomization is recommended.

(b) An unordered tree is either empty or a node with a *set* of children, which are themselves the roots of (possibly empty) **unordered** trees. Two unordered trees are isomorphic if they are both empty, or the subtrees or each tree can be ordered so that their ith subtrees are isomorphic for all i. Describe an algorithm to find the largest common unordered subtree of two unordered trees T_1 and T_2.

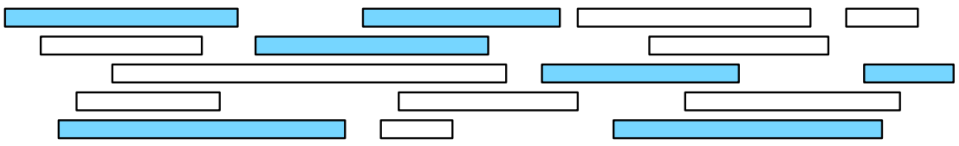
Problem 2. **[25 pts]** Let $G = (V, E)$ be a directed graph with weighted edges; edge weights could be positive, negative, or zero.

(a) How could we delete an arbitrary vertex v from this graph, without changing the shortest-path distance between any other pair of vertices? Describe an algorithm that constructs a directed graph $G' = (V', E')$ with weighted edges, where $V' = V \backslash \{v\}$, and the shortest-path distance between any two nodes in G' is equal to the shortest-path distance between the same two nodes in G, in $O(V^2)$ time.

(b) Now suppose we have already computed all shortest-path distances in G'. Describe and analyze an algorithm to compute the shortest-path distances from v to every other vertex, and from every other vertex to v, in the original graph G, in $O(V^2)$ time.

(c) Combine parts (a) and (b) into another all-pairs shortest path algorithm that runs in $O(V^3)$ time. (The resulting algorithm is not the same as Floyd-Warshall!)

Problem 3. **[25 pts]** Let X be a set of n intervals on the real line. A subset of intervals $Y \subseteq X$ is called a tiling path if the intervals in Y cover the intervals in X, that is, any real value that is contained in some interval in X is also contained in some interval in Y . The size of a tiling cover is just the number of intervals.



A set of intervals. The seven shaded intervals form a tiling path.

Describe and analyze an algorithm to compute the smallest tiling path of X as quickly as possible. Assume that your input consists of two arrays $X_L[1 \cdot \cdot n]$ and $X_R[1 \cdot \cdot n]$, representing the left and right endpoints of the intervals in X. Prove that your algorithm is correct.

Problem 4. **[25 pts]** Suppose you are given n bolts and n nuts. The bolts have distinct sizes. Also, the nuts have distinct sizes. Each bolt match exactly one nut.

Consider the following randomized algorithm for choosing the **largest** bolt. Draw a bolt uniformly at random from the set of n bolts, and draw a nut uniformly at random from the set of n nuts. If the bolt is smaller than the nut, discard the bolt, draw a new bolt uniformly at random from the unchosen bolts, and repeat. Otherwise, discard the nut, draw a new nut uniformly at random from the unchosen nuts, and repeat. Stop either when every nut has been discarded, or every bolt except the one in your hand has been discarded.

(a) What is the probability that the algorithm discards NO bolts?

(b) What is the exact expected number of discarded bolts when the algorithm terminates?

(c) What is the probability that the algorithm discards exactly one nut?

(d) What is the exact expected number of discarded nuts when the algorithm terminates?

(e) What is the exact expected number of nut-bolt tests performed by this algorithm?