Practice Problems for Quiz 2

Problem -1. True or False, and Justify

Circle **T** or **F** for each of the following statements to indicate whether the statement is true or false, respectively. If the statement is correct, briefly state why. If the statement is wrong, explain why. Your justification is worth more points than your true-or-false designation.

T F To determine if two binary search trees are identical trees, one could perform an inorder tree walk on both and compare the output lists.

T F Constructing a binary search tree on n elements takes $\Omega(n \log n)$ time in the worst case (in the comparison model).

T F A greedy algorithm for a problem can never give an optimal solution on all inputs.

T F Suppose we have computed a minimum spanning tree of a graph and its weight. If we make a new graph by doubling the weight of every edge in the original graph, we still need $\Omega(E)$ time to compute the cost of the MST of the new graph.

T F 2-3-4 trees are a special case of B-trees.

T F You have n distinct elements stored in an augmented red-black tree with an extra field per node containing the number of elements in the subtree rooted at that node (including itself). The rank of an element is the number of elements with value less than or equal to it (including itself). The best algorithm for finding the rank of a given element runs in linear time.

T F Let $G = (V, E)$ be a connected, undirected graph with edge-weight function $w : E$ to reals. Let $u \in V$ be an arbitrary vertex, and let $(u, v) \in E$ be the least-weight edge incident on u; that is, $w(u, v) = \min \{w(u, v') : (u, v') \in E\}$. (u, v) belongs to some minimum spanning tree of G.

T F It is possible to do an amortized analysis of a heap (plain old stored-in-an-array-discussed-at-the-time-of-heapsort heap) to argue that the amortized time for insert is $O(\log n)$ (where n is the number of items in the heap at the time of insertion) and the amortized time for extract-max is $O(1)$.

Problem -2. Minimum and Maximum Spanning Trees

(a) It can be shown that in any minimum spanning tree (of a connected, weighted graph), if we remove an edge (u, v), then the two remaining trees are each MSTs on their respective sets of nodes, and the edge (u, v) is a least-weight edge crossing between those two sets.

These facts inspire the 6.046 Staff to suggest the following algorithm for finding an MST on a graph $G = (V, E)$: split the nodes arbitrarily into two (nearly) equal-sized sets, and recursively find MSTs on those sets. Then connect the two trees with a least-cost edge (which is found by iterating over E).

Would you want to use this algorithm? Why or why not?

(b) Consider an undirected, connected, weighted graph G. Suppose you want to find the *maximum* spanning tree. Give an algorithm to find the maximum spanning tree.

Problem -3. Finding Shortest Paths

We would like solve, as efficiently as possible, the single source shortest paths problem in each of the following graphs. (Recall that in this problem, we must find a shortest path from a designated source vertex to every vertex in the graph.) For each graph, state which algorithm (from among those we have studied in class and section) would be best to use, and give its running time. If you don't know the name of the algorithm, give a *brief* description of it. If the running time depends on which data structures are used in the implementation, choose an arbitrary data structure.

(a) A weighted directed acyclic graph.

(b) A weighted directed graph where all edge weights are non-negative; the graph contains a directed cycle.

(c) A weighted directed graph in which some, but not all, of the edges have negative weights; the graph contains a directed cycle.

Problem -4. Roots of a graph

A **root** of a directed graph $G = (V, E)$ is a node r such that every other node $v \in V$ is reachable from r.

(a) [3 points] Give an example of a graph which does not have a root.

(b) [10 points] Prove the following claim.

Consider the forest constructed by running depth-first-search (DFS) on the graph G. Let T be the last tree constructed by DFS in this forest and let r be the root of this tree (i.e, DFS constructed T starting from the node r.) If the graph G has a root, then r is a root of G.

(c) [7 points] Using the result proved in **(b)**, give an $O(|V|+|E|)$-algorithm which when given a graph G, finds a root of the graph if one exists, and outputs NIL otherwise.

(d) [5 points] Suppose you are a given a root r of a graph G. Give an $O(|V| + |E|)$-algorithm to find all the roots of the graph.

(Hint: u is a root of G iff r is reachable from u).

Problem -5. Test-Taking Strategies

Consider (if you haven't already!) a quiz with n questions. For each $i = 1, \ldots, n$, question i has integral point value $v_i > 0$ and requires $m_i > 0$ minutes to solve. Suppose further that no partial credit is awarded (unlike this quiz).

Your goal is to come up with an algorithm which, given v_1, v_2, ..., v_n, m_1, m_2, ..., m_n, and V, computes the minimum number of minutes required to earn at least V points on the quiz. For example, you might use this algorithm to determine how quickly you can get an A on the quiz.

(a) Let $M(i, v)$ denote the minimum number of minutes needed to earn v points when you are restricted to selecting from questions 1 through i. Give a recurrence expression for $M(i, v)$.

We shall do the base cases for you: for all i, and $v \leq 0$, $M(i, v) = 0$; for $v > 0$, $M(0, v) = \infty$.

(b) Give pseudocode for an $O(nV)$-time dynamic programming algorithm to compute the minimum number of minutes required to earn V points on the quiz.

(c) Explain how to extend your solution from the previous part to output a list S of the questions to solve, such that $V \leq \sum_{i \in S} v_i$ and $\sum_{i \in S} m_i$ is minimized.

(d) Suppose partial credit is given, so that the number of points you receive on a question is proportional to the number of minutes you spend working on it. That is, you earn v_i/m_i points per minute on question i (up to a total of v_i points), and you can work for fractions of minutes. Give an $O(n \log n)$-time algorithm to determine which questions to solve (and how much time to devote to them) in order to receive V points the fastest.