Comprehensive Exam: Algorithms and Concrete Mathematics Autumn 2007

This is a one hour closed-book exam and the point total for all questions is 60.

In questions that ask you to provide an algorithm, please explain the algorithm in words and diagrams, no need to write code or pseudo code. Also, for any algorithm, prove correctness and prove its running time. No credit will be given for exponential-time algorithms. Polynomial but slow algorithms will get some partial credit. Amount of credit will depend on how much slower they are compared to what is achievable using the knowledge in the reading list. Correct and fast algorithms with incomplete proof of correctness will get small number of points.

For full credit, the answers should be short and precise. Long and convoluted answers will not get full credit even if they are correct.

1. **[14 pts]** Please answer "true" or "false" to each one of the following questions. Correct answers will give you **(2 pts)** each while wrong answers will reduce your score by **(2 pts)** each.

   (a) There exists $\epsilon > 0$ such that $n^\epsilon = O(\log n)$.

   Answer: FALSE

   (b) Suppose $T(n)$ is given by the recurrence $T(n) = T(\lfloor n/\log n \rfloor) + n$; $T(1) = 1$. Then, $T(n) = \Theta(n)$.

   Answer: TRUE. Lower bound is obvious. Upper bound follows from the fact that $\lfloor n/\log n \rfloor$ is bounded by $n/2$ for large enough n.

   (c) Given a heap data structure (organized with minimum on top), it is possible to find the second-smallest element in constant time.

   Answer: TRUE

   (d) There exists a deterministic linear time algorithm that, given two English words with at most n characters each, determines whether they are anagrams of each other.

   Answer: TRUE. Count the frequency of all 26 letters and compare.

   (e) In comparison model, the lower bound on finding median of n input numbers is $\Omega(n \log n)$.

   Answer: FALSE. Median can be computed in linear time.

   (f) You are given a graph $G = (V, E)$ with positive lengths on edges and a shortest path P from $v \in V$ to $u \in V$. Next, lengths are transformed by computing square of the length of each edge. (If old edge length was w, new one is w^2). Claim: it is guaranteed that P is a shortest path from v to u with respect to new lengths as well.

   Answer: FALSE. Example: graph with three nodes, $w(ab) = w(bc) = 1$, and $w(ac) = 2$. Edge ac is shortest path between a and c in the original graph. After squaring of the costs this is not true anymore.

   (g) Given a weighted graph and two nodes, it is possible to list all shortest paths between these two nodes in polynomial time.

   Answer: FALSE. There can be an exponential number of shortest paths.

2. **[18 pts]** You have a single classroom and a list of lectures that you would like to use the classroom for. Each lecture is specified by start time and end time. Your goal is to come up with a schedule that maximizes the number of lectures scheduled in the classroom under the constraint that no two scheduled lectures can overlap. For example, if one lecture needs the 1pm to 2pm slot and the other one needs 1:30pm to 3pm slot, at most one of these lectures can be scheduled. Design an efficient algorithm to solve the problem. Prove correctness and running time.

   Answer: Sketch of the algorithm. Greedy approach: sort by finish time; pick the first lecture, allocate it, delete overlapping lectures, repeat. Claim: there exists an optimum

solution that includes the first lecture we allocated. Proof: Take any optimum solution and replace the first allocated lecture in this solution with the first lecture in our solution. Our lecture finishes no later than the first allocated lecture in the OPT, thus no overlaps created, and thus the modified solution is legal. This solution has the same number of lectures scheduled and includes the first lecture in our solution. QED. The claim implies that we can greedily commit to the earliest-finish-time lecture. Once the lectures are sorted, each iteration (picking the next lecture and deciding whether we schedule it or drop it) takes constant time. Total running time is $O(n \log n)$.

3. **[12 pts]** Prove that if edge weights of a graph are unique (no two edges have the same weight), then there is unique solution to the minimum-cost spanning tree problem.

   Answer: Let T be the tree produced by Kruskal's algorithm and let T' be a different minimum-cost tree. Let uv be the first edge where Kruskal's algorithm disagrees with T', i.e. this is the smallest-weight edge that Kruskal's algorithm put into T but uv is not part of T'. [Observe that this is the only possible disagreement since if uv was rejected by Kruskal's algorithm, it closes a cycle together with already chosen edges. But all these edges are in T' as well.] Add uv to T', creating a cycle. Note that the path from u to v in T' has to include at least one edge with higher weight than $w(uv)$ since when uv was chosen by Kruskal's algorithm, there was no path from u to v using already chosen edges. Cut-and-paste argument finishes the proof.

4. **[16 pts]** Given an acyclic directed graph $G = (V, E)$ and a node $s \in V$, describe an algorithm to find the number of paths from s to each one of the nodes in V.

   Answer: First compute topological sort. Then use a simple dynamic programming. For any vertex v, let $P[v]$ denote the number of paths from s to v. Initialize $P[v] = 0$ for all $v \in V$, and set $P[s] = 1$. Compute the recurrence $P[v] = \sum_{uv \in E} P[u]$ in topological order.

   Topological sort guarantees that $P[v]$ depends only on values of $P[]$ for nodes that are before v in topological sort. Thus, when we compute $P[v]$, the $P[]$ values for all these nodes were already computed. Topological sort can be implemented to run in $O(|E|+|V|)$. Second phase of the algorithm scans nodes one-by-one, each time examining all incoming edges, again giving $O(|E| + |V|)$.