Midterm Examination

CS 4104 (Spring 2014)

Assigned: March 19, 2014.
Due: on Scholar before the beginning of class on March 26, 2014.

Name: _____

PID: _____

Instructions

1. **You must work on your own for this examination, i.e., you are not allowed to have a partner.**

2. For every algorithm you describe, prove its correctness, and state and prove the running time of the algorithm. I am looking for clear descriptions of algorithms and for the most efficient algorithms and analysis that you can come up with. I am not specifying the desired running time for each algorithm. I will give partial credit to non-optimal algorithms, as long as they are correct.

3. You may consult the textbook, your notes, or the course web site to solve the problems in the examination. Of course, the TA and I are available to answer your questions. You **may not** work on the exam with anyone else, ask anyone questions, or consult other textbooks or sites on the Web for answers. **Do not use** concepts from Chapters 6 and later in the textbook.

4. You must prepare your solutions digitally, i.e., do not hand-write your solutions.

5. I prefer that you use LaTeX to prepare your solutions. However, I will not penalise you if you use a different system. To use LaTeX, you may find it convenient to download the LaTeX source file for this document from the link on the course web site. At the end of each problem are three commented lines that look like this:

   ```
   % \solution{
   %
   % }
   ```

   You can uncomment these lines and type in your solution within the curly braces.

Good luck!

Problem 1 (15 points) Let us start with some quickies. For each statement below, say whether it is true or false. You do not have to provide a proof or counter-example for your answer.

1. Every tree is a bipartite graph.
2. $\sum_{i=1}^{n} i \log i = \Theta(n^2 \log n)$.
3. You solve a problem on an input of size n by dividing it into three subsets of size $n/3$, solving each sub-problem recursively, and combining the solutions in $O(n)$ time. The running time of your algorithm is $O(n \log n)$.
4. In a directed graph, $G = (V, E)$, each edge has a weight between 0 and 1. To compute the length of the *longest* path that starts at u and ends at v, we can change the weight of each edge to be the reciprocal of its weight and then apply Dijkstra's algorithm.
5. A connected graph on n nodes has k cycles. We can compute its minimum spanning tree in $O(nk)$ time.

Problem 2 (10 points) Consider the recurrence relation for mergesort:

$$\begin{aligned} T(n) &\leq 2T(n/2) + cn, n > 2 \\ T(2) &\leq c \end{aligned}$$

Try to prove by induction that for all n, $T(n) \leq kn$ for some constant k that is to be determined. Either provide a correct proof or discuss why you cannot complete the proof. Please show all the steps.

Problem 3 (10 points) Consider the problem of minimising lateness that we discussed in class. We are given n jobs. For each job $i, 1 \leq i \leq n$, we are given a time $t(i)$ and a deadline $d(i)$. Let us assume that all the deadlines are distinct. We want to schedule all jobs on one resource. Our goal is to assign a starting time $s(i)$ to each job such that each job is delayed as little as possible. A job i is *delayed* if $f(i) > d(i)$; the *lateness of the job* is $\max(0, f(i) - d(i))$.

Define

1. the *lateness of a schedule* as $\max_i \left(\max \left(0, f(i) - d(i)\right)\right)$ and
2. the *delay of a schedule* as $\sum_{i=1}^{n} \left(\max \left(0, f(i) - d(i)\right)\right)$.

Note that although the words "lateness" and "delay" are synonyms, for the purpose of this problem we are defining them to mean different quantities: the lateness of a schedule is the *maximum* of the latenesses of the individual jobs, while the delay of a schedule is the *sum* of the latenesses of the individual jobs.

Consider the algorithm that we discussed in class for computing a schedule with the smallest lateness: we sorted all the jobs in increasing order of deadline and scheduled them in this order. We proved that the earliest-deadline-first algorithm correctly solves the problem of minimising lateness. If we were to use the *same proof* to try to demonstrate that the algorithm correctly solves the problem of minimising delay, where does the proof break down?

Problem 4 (30 points) Given a directed graph $G(V, E)$ where every edge $e \in E$ has a positive cost l_e, for every pair s, t of vertices in V, let us define $d(s, t)$ as the length of the shortest path between s and t. Note that there may be many shortest paths between a pair (s, t) of vertices but they will all have the same length. Let us try to represent all these shortest paths in a graph $G_{s,t}$, which we construct as follows: for every edge $e = (u, v)$ such that G contains a path from s to u and a path from v to t, we add e to $G_{s,t}$ if and only if

$$d(s, t) = d(s, u) + l_e + d(v, t),$$

i.e., if the shortest path length from s to t is the sum of the three quantities: the shortest path length from s to u, the length of the edge e, and the shortest path length from v to t. In other words, we add e to $G_{s,t}$ if and only if there is some shortest path from s to t that contains e.

Note that we are adding edges individually to $G_{s,t}$ rather than adding entire shortest paths. Therefore, we do not know whether there is a one-to-one correspondence between all the paths in $G_{s,t}$ and all the shortest paths between s and t in G. Prove the following two statements:

(a) (10 points) Every shortest path between s and t in G is a path in $G_{s,t}$.

(b) (20 points) Every path in $G_{s,t}$ is a shortest path between s and t in G. In other words, there cannot be a path in $G_{s,t}$ that has smaller or largest length than $d(s,t)$.

Problem 5 (35 points) Given a directed graph $G = (V, E)$, a positive cost length l_e on every edge e in E, and two vertices s and t, we can use Dijkstra's algorithm to compute the shortest path from s to t. Describe an algorithm to compute the *second* shortest path between s and t. Show that the running time of your algorithm is polynomial in the size of G. As an example, in the graph below, there are two shortest paths between s and t, both of length 2 (shown in green). One shortest path is the edge (s,t) itself. The other shortest path has two edges (s,u), and (u,t). Both of these are shortest paths between s and t. The second shortest path (shown in purple) has length 4 and has two edges (s,v) and (v,t). I want you to compute the second shortest path between s and t (one such path will suffice if there are many), for an arbitrary graph, of course.