Midterm Examination

CS 5114 (Spring 2013)

Assigned: March 19, 2013.
Due: at the beginning of class on March 26, 2013.

Name: _____

PID: _____

Instructions

1. 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.

2. 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 7 and later in the textbook.

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

4. 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.

5. Submission instructions: If you are not a student in the Blacksburg campus, email a PDF version of solutions to `murali@cs.vt.edu`. Otherwise, submit a hard copy of your solutions. Do not forget to staple the hard copy you hand in.

Good luck!

Problem 1 (10 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. See pages 14 and 15 of your textbook for a definition of a bipartite graph.
2. $\sum_{i=1}^{n} i \log i = \Theta(n^2 \log n)$.
3. In a directed graph, $G = (V, E)$ where each edge has a positive weight, let $\pi(u, v)$ denote the length of the shortest path that starts at u and ends at v. If there is no such path, you can assume that $\pi(u, v) = \infty$. Since G is directed, $\pi(u, v) \neq \pi(v, u)$, in general. Then, for any three nodes $u, v, w \in V$, $\pi(u, v) + \pi(v, w) \geq \pi(u, w)$.
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 (25 points) Solve the following recurrence relation. You can assume that $T(1) = 0$, $T(2) = 1$ and c is a positive constant. Note that I am expecting you to prove the tightest bound possible on $T(n)$.

$$T(n) = \frac{1}{n} \sum_{i=1}^{n-1} \big(T(i) + T(n-i)\big) + cn.$$

Hint: Unrolling the recurrence may lead to messy expressions. You could guess the solution and prove the guess is correct using induction. To try to prove it directly, compute $nT(n) - (n-1)T(n-1)$ and see if you notice a pattern.

Problem 3 (25 points) After graduating from Virginia Tech, you start working at a company that makes games for phones. You are assigned to work on a game called PoodleJump. The goal here is for a player to move a character (a Poodle, to be precise) so that it jumps from one ledge to another. The position of a ledge is specified by its x-coordinate and its z-coordinate. The Poodle can only jump to the left. A jump is *good* if the poodle jumps to the left and if the ledge l that the poodle jumps on has the following property: every ledge to the right of ledge l, i.e., with x-coordinate larger than that of l, is below l, i.e., has a smaller z-coordinate than that of l; Otherwise, the jump is bad. See Figure 1. The poodle starts at the rightmost ledge, i.e., the one with the largest x-coordinate. The game ends as soon as the Poodle makes a jump that is bad. Given a set of n ledges (and their positions), devise an efficient algorithm to compute the largest number of good jumps a player can make and the sequence of ledges that comprise these jumps.

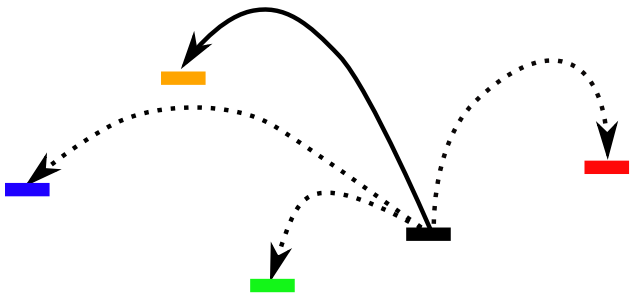

Figure 1: An illustration of good and bad jumps. The jumps from the black ledge to the red, green, and blue ledges are bad: red because the poodle jumps to the right; green because the black ledge is to the right of and above the green ledge; and blue because the orange ledge is to the right of and above the blue ledge. The only good jump from is from the black ledge to the orange ledge.

Problem 4 (15 points) The curriculum in the Department of Computer Science at the University of Draconia consists of n courses. Each course is mandatory. The prerequisite graph G has a node for each course, and an edge directed from course v to course w if and only if v is a prerequisite for w. In such a case, a student cannot take course w in the same or an earlier semester than she takes course v. A student can take any number of courses in a single semester. Design an algorithm that computes the minimum number of semesters necessary to complete the curriculum. You may assume that G does not contain cycles, i.e., it is a directed acyclic graph. Chapter 3.6 of the textbook discusses directed acyclic graphs. The graph can be split into multiple components. For example, an extreme case is when there are no pre-requisites, in which case each course is in a separate component. Of course, one semester suffices in this trivial case.

If you are concerned about how G is represented, assume that for each course, you have a list of courses for which it is a pre-requisite (the adjacency list representation). If you need, you can assume that the "list" for each course is stored in an array. if you want to use another representation, please describe it. *Do not use an adjacency matrix representation.*

Problem 5 (25 points) Many object-oriented programming language implement a class for manipulating strings. A primitive operation supported by such languages is to split a string into two pieces. This operation usually involves copying the original string. Hence, it takes n units of time to split a string of length n into two pieces, *regardless of the location of the split.* However, if we want to split a string into many pieces, the order in which we make the splits can affect the total running time of all the splits.

For example, suppose we want to split a 20-character string at positions 3 and 10. If we make the first cut at position 3, the cost of the first cut is the length of the string, which is 20. Now the cut at position 10 falls within the second string, whose length is 17, so the cost of the second cut is 17. Therefore, the total cost is $20 + 17 = 37$. Instead, if we make the first cut at position 10, the cost of this cut is still 20. However, the second cut at position 3 falls within the first string, which has length 10. Therefore, the cost of the second cut is 10, implying a total cost of $20 + 10 = 30$.

Design an algorithm that, given the locations of m cuts in a string of length n, finds the minimum total cost of breaking the string into $m + 1$ pieces at the given locations, minimised over all possible ways of breaking the string at the m locations.