College of Computer & Information Science | Fall 2010
Northeastern University | Handout 20
CS7800: Advanced Algorithms | 15 December 2010

Take-Home Final Exam

Due by 5:30 PM, Thursday, December 16

- **Honor code:** This exam is open-notes, open-library, and open-web. However, *no collaboration of any kind is allowed.* ("Collaboration" includes, for example, discussion or exchange of material related to the problems on the exam with anyone other than the instructor.) If you have any questions or clarifications, please ask me.
- **Policy on Cheating:** Students who violate the above rules on scholastic honesty are subject to disciplinary penalties. Any student caught cheating will receive an **F** (failing grade) for the course, and the case will be forwarded to the Office of Student Conduct and Conflict Resolution.
- **Presentation of solutions:** While describing an algorithm, you may use any of the algorithms covered in class or in the text as a subroutine, without elaboration.
- **A note on grading:** Your grade on any problem that asks you to design an algorithm will be determined on the basis of its correctness and the clarity of the description. In case you are not able to present an algorithm that has the desired properties, give the best algorithm you have designed. Show your work, as partial credit may be given.
- **Good Luck!**

Problem 1. (4 + 5 + 6 = 15 points) Proofs and counterexamples

(a) Prove or disprove: Let G be a connected graph with a weight function w on edges. Let T be a minimum spanning tree of G, with respect to w. Let C be any cut of G, and let e and e' be two edges crossing the cut C such that e is in T and e' is not in T. Then $w(e) \leq w(e')$.

(b) We know that a sequence of n splay operations on any n-node tree incurs $O(n \log n)$ time. Recall that each splay operation is a sequence of zig, zig-zig, or zig-zag steps. Suppose we replace the zig-zig and zig-zag steps on a node x by the following step: if x has a parent y and a grandparent, first *rotate*(y) and then *rotate*(x). Give an n-node tree and a sequence of n splay operations applied on the tree for which this modified splay implementation will cost $\Omega(n^2)$ time.

You need not give a detailed proof. A precise definition of the tree, the sequence, and a brief argument for the time would suffice.

Clarification: By *rotate*(), we refer to the standard rotation operation used in binary search trees. Just to be sure, in the first illustration on the following webpage, tree on the right is obtained by applying *rotate*(P) to the tree on the left. Similarly, the tree on the left is obtained by applying *rotate*(Q) to the tree on the right.

`http://en.wikipedia.org/wiki/Tree_rotation`

(c) Consider a network flow (directed) graph G with a capacity on each edge, a source s, and a sink t. We say that an edge e is *crucial* if decreasing its capacity will decrease the maximum flow from s to t. We say that an edge e is a *bottleneck* if increasing its capacity will increase the maximum flow from s to t. Prove or disprove each of the following two statements: (i) if an edge is crucial, then it is also a bottleneck; (ii) if an edge is a bottleneck, then it is also crucial.

Clarification: The capacity of each edge is a nonnegative integer. By "decreasing" (resp., "increasing") capacity, you may assume decreasing (resp., increasing) by 1.

Problem 2 (10 + 2 = 12 points) Cheapest short paths

Let $G = (V, E)$ be a directed graph, and $\ell : E \to N$ and $c : E \to N$ be two functions, where N is the set of positive integers. We call $\ell(e)$ to be the length of edge e and $c(e)$ to be the cost of edge e. Given G, ℓ, c, a source $s \in V$, a destination $t \in V$, and an integer L, design an algorithm for finding a path from s to t with length at most L and whose cost is smallest among all paths from s to t of length at most L. (If not such path exists, then your algorithm should indicate so.) The worst-case running time of your algorithm must be polynomial in $|V|$, $|E|$, L, and the sizes of ℓ and c.

You need not prove the correctness of your algorithm; nor do you need to optimize the running time of your algorithm.

Given G, ℓ, c, s, t, and L, as above, the problem of finding the minimum-cost path from s to t with length at most L is known to be NP-complete. Explain why we cannot infer from the preceding fact and your algorithm that $P = NP$.

Clarifications: One correction – I have added s and t in two places in the first sentence of the preceding paragraph. The size of ℓ (resp., c) in the first paragraph of the problem refers to the total number of bits needed in the standard binary representation of the $\ell(e)$'s (resp., $c(e)$'s). In general, the size of a problem always refers to the number of bits needed to describe the input. Thus, for the above "cheapest short paths problem", the size is the total number of bits needed to describe each of these: G (in adjacency list form), ℓ, c, s, t, and L.

Problem 3. (8 points) Finding a fitting line

Design a polynomial-time algorithm for the following problem. Given a set of n points (x_i, y_i), $1 \le i \le n$ find a line $ax + y = c$ (i.e., find a and c) that best fits the n points in the following sense: the line minimizes

$$\max_i |ax_i + y_i - c|.$$

Assume that x_i's and y_i's are all integers.

You need not prove the correctness of your algorithm; nor do you need to optimize the running time of your algorithm.

Clarification: Note that one requirement for the (a, b, c) you compute is that a and b cannot be both 0, since $(0, 0, c)$ does not represent any line.

Correction: Some of you pointed out that a, b, and c can be made arbitrarily small, possibly leaving no solution to the above problem. This issue arises since three parameters are one too many to describe a line. I have made the following change above: set b to 1, so that the only variables are a and c.