15-451 Algorithms, Fall 2012

Homework # 7 **due: Thursday December 6, 2012**

Please hand in each problem on a separate sheet and put your **name** and **recitation** (time or letter) at the top of each sheet. You will be handing each problem into a separate box, and we will then give homeworks back in recitation.

Remember: written homeworks are to be done **individually**. Group work is only for the oral-presentation assignments.

Problems:

(25 pts) 1. [Randomized rent-or-buy]

Consider the rent-or-buy problem in the simple case that the cost to buy a tuxedo is twice the rental cost. The optimal deterministic algorithm (as discussed in class) is: "rent the first time, buy the second time" for a competitive ratio of 3/2.

For a randomized algorithm, the definition of competitive ratio is the worst case, over all possible scenarios, of the ratio of our *expected* cost under that scenario to the optimal cost for that scenario. (Scenario = how many times we need a tuxedo.)

It turns out that if the purchase cost is twice the rental cost, there are only two choices of scenario that we need to worry about: either (a) we only go to one formal event or else (b) we go to infinitely many. Because of this, there are only two deterministic strategies that make sense for us to randomize over: either (1) buy right away the first time, or (2) rent the first time and then buy if we go to a second event.

What randomized strategy has the best randomized competitive ratio and what is its ratio? Note: your randomized algorithm will be a minimax optimal strategy for a suitably-defined matrix game.

(25 pts) 2. [Inverse vertex cover]

As discussed in class, the problem of finding the fewest *vertices* needed to touch all *edges* in a graph is NP-complete (the vertex-cover problem). However, it turns out that the problem of finding the fewest *edges* needed to touch all the *vertices* in a graph can be solved in polynomial time. Give an efficient algorithm and prove that it is correct in that it indeed finds the minimum number of edges needed to touch all the vertices.

Hint 1: try some small examples to get a feel for this problem.

Hint 2: it may be helpful to recall that finding a *maximum matching* in a graph can be done in polynomial time.

(25 pts) 3. [String matching with wildcards]

In this problem you are trying to locate all instances of something like `s**rch` in a file, where the stars can match any character — so this should bring up words like "search" and "starch". We'll call the thing we are looking up the *pattern* P and we'll call the file the *text* T.

Say P has length n and T has length m. There is a simple $O(mn)$-time algorithm to solve this problem: try all $m - n$ possible starting positions, and for each one, check in time $O(n)$ to see if P matches there.

Your job is to give an algorithm that solves this problem in time $O(m \log m)$, by using the fact that we can compute convolutions quickly. We're going to assume we are working over a binary alphabet to make our life easier, and the output of the algorithm should be a list of all locations of P in T. For instance, if $P =$ `11*1` and $T =$ `10111101`, then P appears twice in T: once starting at the 3rd position in T and once starting at the 5th position in T.

(a) First solve a simpler version of the problem where (as in the example above) there are no zeroes in P. I.e., P is a string of 1s and $*$s.

(b) Now solve the problem where P also has zeroes. Hint: as a first step, in linear time, go through S and T and replace each "1" with "10", and replace each "0" with "01". Figure out what to replace stars by so that you can use the same approach as in (a).

(25 pts) 4. [Longest PreSufMid] Give an algorithm that takes as input a string S of length n and finds the longest string that is a prefix of S, a suffix of S, and also occurs in the middle of S. By "middle" we mean that the occurrence does not make use of the first or last character of S. Your algorithm should run in time $O(n \log n)$, or faster.