CSC 364H **Problem Set 1** **Summer 99**

Due: day section: June 9, at 2:10 (in tutorial)
evening section: June 8, at 6:10 (in tutorial)

Note: Assignments are due at the *beginning* of tutorial, and since solutions are discussed in tutorial, late assignments are *not* accepted. The work that you submit *must* be your own. You may discuss problems with each other; however, you should take no notes during these discussions, and you should prepare written solutions alone.

1. Suppose the runtime of an algorithm is $T(n)$. Indicate which of the expressions below for $T(n)$ would make the algorithm polytime.

a) 3^{n^2}	e) $2^{(\log_2 n)^2}$
b) $3^{\log_2 n}$	f) $2^{\sqrt{n}}$
c) $n^{1/n}$	g) $(0.5)^n$
d) $1000n^{17} + n^2 \log_2 n$	h) $2^{1000}n$

2. (a) Suppose that the edge costs are all distinct in an input G to the MST problem. Prove that G has a unique minimum cost spanning tree. Do this by considering the proof that Kruskal's algorithm always finds a minimum cost spanning tree.
   (b) Now suppose that all edge costs are distinct, except $c_1 = c_2$, where $c_1 = c(e_1)$ and $c_2 = c(e_2)$ are the two minimum edge costs. Prove or give a counter-example: G has a unique minimum cost spanning tree.
   (c) Finally suppose that all edge costs are distinct, except $c_1 = c_2 = c_3$, where $c_1 = c(e_1), c_2 = c(e_2), c_3 = c(e_3)$ are the three minimum edge costs. Prove or give a counter-example: G has a unique minimum cost spanning tree.

3. (Reference: Brassard and Bratley, Problem 6.6) Consider the Make-Change problem, where we have to pay a given amount using the least number of coins, using some fixed denominations. Suppose that the available denominations are $\{1, p, p^2, \ldots, p^n\}$ where $p > 1$ and $n \geq 0$ are integers. Consider the following greedy algorithm to solve the change making problem:

   function MakeChange(N) (*makes change for N units using coins from C*)
       const $C \leftarrow \{1, p, p^2, \ldots, p^n\}$
       $L \leftarrow \emptyset$ (*L is the list holding the solution*)
       $s \leftarrow 0$ (*s is the sum of the items in L*)
       while $s < N$ do
           x = largest item in C such that $s + x \leq N$
           $L \leftarrow L, x$
           $s \leftarrow s + x$
       return L

Claim: With this series of denominations, and an unlimited supply of coins of each denomination, the greedy algorithm above always finds an optimal solution.

Your task is to prove this claim. To this end, define L to be promising if it can be extended (by adding a list of coins) to an optimal solution to the Make-Change problem with input (p, n, N). Now prove that "L is promising" is a loop invariant of the above algorithm. The Basis Case is easy. In the Induction Step assume that L is promising, and show that L continues to be promising after one more execution of the loop.

4. Recall the Longest Monotone Subsequence problem defined in class. Consider the following variation of the LMS problem:

   Input: $d, a_1, a_2, \ldots, a_d \in \mathbb{N}$.

   Question: What is the length of the longest subsequence of $a_1, a_2, \ldots, a_d$, where any two consecutive members of the subsequence differ by at most 1?

   For example, the longest such subsequence of $\{7, 6, 1, 4, 7, 8, 20\}$ is $\{7, 6, 7, 8\}$, so in this case the answer would be 4.

   Give a dynamic programming solution to this problem by doing the following three steps:

   (a) Define an array R of subproblems.

   (b) Find a recurrence (including initial conditions) that could be used to compute the values in the array R.

   (c) Write an algorithm, at a high level, for computing the values in the array R.

5. Consider an input $C(i, j)$, $1 \leq i, j \leq n$, to an all pairs shortest path problem, as described in class. Suppose that these distances satisfy the restriction that $C(i, j) = \infty$ if $j < i$. (Intuitively, you should think of the nodes of the graph as ordered $1 < 2 < \ldots < n$, and it is impossible to move backwards in the graph.) Give a simple dynamic programming solution to the problem of finding the shortest distance from node 1 to every other node in the graph.

   Note that because of the restriction we are placing on the graph, your solution should be simpler than the Floyd's algorithm given in class, and operate in time $O(n^2)$ at worst.

   Specifically, you should:

   (a) Define an array D.

   (b) Give a recurrence (including initial conditions) that could be used to compute the values in the array D.

   (c) Give, at a high level, an algorithm for computing the values of the array D. Estimate the runtime of your algorithm using O-notation.