CS 1510 Midterm 1
Fall 2009

1. (40 points) We consider the following problem:

   INPUT: A collection W of positive integer weights $w_1, \ldots, w_n$.

   OUTPUT: A binary tree T with n leaves, where each leaf ℓ is labeled with a unique weight $w_{\sigma(\ell)}$ from W, that minimizes the sum over the leaves ℓ in T of the product of depth $h(\ell)$ of the leaf ℓ times the weight $w_{\sigma(\ell)}$. That is the binary tree T and the labeling σ should minimize

   $$\sum_{\text{leaf } \ell \in T} h(\ell) * w_{\sigma(\ell)}$$

   We consider two greedy algorithms for solving this problem. For each algorithm, either prove or disprove that it is correct. The first sentence of a disproof should state what it means for an algorithm to be correct, and how one normally proves an algorithm incorrect. A proof of correctness must use an exchange argument, and include some reasonable explanation of what an exchange argument is, and why it is sufficient to establish correctness.

   Hoffman's Algorithm The algorithm recursively makes the heaviest remaining node the right child of the root of the current subtree, and recursively constructs the left subtree with the remaining weights.

   If the above description wasn't clear to you, here is a more technical explanation. This recursive algorithm takes two parameters, a pointer P to a node in the tree, and a list L of remaining weights. Initially the pointer P points to a single node (that will be the root of the constructed tree), and the list L of weights is W. If L has only one weight, the node that P points to is labeled with the only weight in L. Otherwise, the algorithm makes the heaviest weight $w_k \in L$ the right child of the node pointed to by P, creates a new node for the left child, and then recurses with a pointer to the left child and list $L - \{w_k\}$.

   Huffman's Algorithm The algorithm maintains a list L of weights, and a forest F. Initially L is W and F is consists of one node for each weight in W. The algorithm then iterates the following until L has only 1 weight:

   - Find the two smallest weights w_i and w_j in L
   - Create a new node in F labeled $w_i + w_j$, and make the nodes labeled w_i and w_j its children.
   - Delete w_i and w_j from L, and add $w_i + w_j$ to L

2. (20 points) We consider the longest common subsequence (LCS) problem. Recall that the input to the LCS problem is two strings $A = A_1 \dots A_m$ and $B = B_1 \dots B_n$.

   (a) Give simple recursive pseudo-code to compute the length of the longest common subsequence of two strings A and B.

   (b) Let the m by n table T be defined as follows: $T[i, j]$ is the length of the longest common subsequence of $A_1 \dots A_i$ and $B = B_1 \dots B_j$. Give pseudo-code, which runs in time $O(mn)$, to fill in this table.

   (c) Show the table T constructed by your code from the previous part for the two strings $A = yxyx$ and $B = xyxy$.

   (d) Explain how, given the filled in table T, the string A and the string B, to find the actual longest common subsequence in time $O(m + n)$.

3. (20 points) Consider the following problem:

   INPUT: A tree T with integer profits on the edges.

   OUTPUT: The most profit that one can obtain from a simple path P in T.

   The profits may be positive or negative or zero. A negative profit can be thought of as a cost. A path is simple if it doesn't repeat any edge. The profit of a path is the sum of the profits of the edges. The profit of a path consisting of one vertex, and no edges, is 0.

   Give a linear time algorithm to solve the above problem. Your algorithm should traverse the tree in a post-order fashion. So you need to explain two things:

   - Before the algorithm starts computing at a node v, what has it computed on the subtrees rooted at the children of v, and
   - how does the algorithm compute this same information for the subtree rooted at v.

4. (20 points) The input to this problem is a sequence of n points $p_1, \dots p_n$ in the Euclidean plane. The three taxis start at the origin. Each point must be visited by at least one of the three taxis. If a taxi visits a point p_i before p_j then it must be the case that $i < j$. The cost of a routing is just the total distance traveled by the first taxi plus the total distance traveled by the second taxi plus the total distance traveled by the third taxi. The output is the cost of the least cost routing for the three taxis to service these requests. Give an $O(n^3)$ time algorithm to solve this problem.