Prof. Sergio A. Alvarez
21 Campanella Way, room 569
Computer Science Department
Boston College
Chestnut Hill, MA 02467 USA

http://www.cs.bc.edu/∼alvarez/
alvarez@cs.bc.edu
voice: (617) 552-4333
fax: (617) 552-6790

CS383, Algorithms
Spring 2009
HW7

1. Consider the computational task stated in Algorithm 1. Notice that the only constraint on the coefficients d_i is non-negativity. In particular, there is no predetermined upper bound either on the total number of d_i nor on their individual magnitudes. As an example, both $d_0 = 35,\ d_1 = 4$ and $d_0 = 25,\ d_1 = 5$ are valid representations of the same value $n = \sum_{i=0,1} d_i 10^i$.

   Algorithm 1: Powers of Ten
   Input: Non-negative integer, n.
   Output: Non-negative integers $d_0 ... d_k$ (number k not specified in advance) such that $n = \sum_{i=0...k} d_i 10^i$, and $\sum_{i=0...k} d_i \leq \sum_{i=0...k'} c_i$ whenever $n = \sum_{i=0...k'} c_i 10^i$.

   (a) Cast the computational task in Algorithm 1 as an optimization task of the sort discussed in class in the context of greedy algorithms. Specify each of the following elements in detail: set of candidates C, notion of solution, notion of feasible set, and objective function. State whether the task involves maximization or minimization. Your answers should be specific to the particular computational task in Algorithm 1.

   (b) Using your anwers from 1a, provide detailed pseudocode for a greedy computational procedure for Algorithm 1 that solves the stated computational task.

   (c) Show the details of the computation performed by your algorithm from 1b on the input $n = 395$. What is the optimal value of the objective function for this particular input instance? Explain.

   (d) Give a careful argument that shows that your algorithm in 1b returns a solution that is optimal in the sense of optimizing the objective function from 1a.

2. (From exercise 5.22 in the book by Dasgupta, Papadimitriou, and Vazirani)

   (a) Use the Cut Property as discussed in class to prove the property of undirected weighted graphs stated below.

   > Given any cycle of the graph, there is a minimal spanning tree that does *not* contain the heaviest edge of that cycle.

   Hint: given a cycle, start by picking any MST T of the graph. There are two cases depending on whether the heaviest edge of the cycle forms part of T or not.

   (b) Prove carefully that Algorithm 2 works correctly, that is, that it returns a MST of the input graph.

Algorithm 2: MST By Pruning
Input: Weighted undirected graph G.
Output: A MST of G.
PRUNEMST(G)
(1) sort the edges of G in decreasing order of weight
(2) **foreach** edge e (starting with the heaviest)
(3) **if** e is part of a cycle of G
(4) remove e from G
(5) **return** G

3. Consider the computational task of finding prefix codes of shortest possible expected length per symbol. Refer to my notes on coding and Huffman's algorithm. An obvious greedy strategy for this task is to assign the shortest code, say 0, to the most probable symbol of the alphabet, and longer codes of the form $1X$ to all other symbols, proceeding recursively to assign specific suffixes X to the various remaining symbols as in Algorithm 3.

Algorithm 3: Greedy Coding
Input: Symbol alphabet sequence $\Sigma = (\sigma_1...\sigma_n)$ in decreasing order of occurrence probability: $P(\sigma_1) \geq P(\sigma_2) \geq ... \geq P(\sigma_n)$.
Output: A prefix code $c(\sigma_1)...c(\sigma_n)$ of minimum expected length.
GREEDYCODE($\sigma_1...\sigma_n$)
(1) **if** $n = 0$
(2) **return** empty sequence
(3) $c(\sigma_1) = 0$
(4) $(c(\sigma_2)...c(\sigma_n)) = 1 \cdot$ GREEDYCODE($\sigma_2...\sigma_n$)
(5) **return** $(c(\sigma_1)...c(\sigma_n))$

(a) Show that Algorithm 3 always returns an optimal prefix code (with lowest expected codelength per symbol) for input alphabets with symbol probabilities of the specific form $2^{-1}, 2^{-2}, ...2^{-(n-1)}, 2^{-(n-1)}$. Note that the last two probabilities are equal. This is so that the sum of all of the probabilities is 1.

(b) Does Algorithm 3 work for general input alphabets? If yes, provide a proof. If no, provide a specific counterexample. Explain carefully in either case.