CS 161: Homework 7

Due by December 2, 2016 at noon

Instructions: Please answer the following questions to the best of your ability. Provide full and rigorous proofs and include all relevant calculations. When writing proofs, please strive for clarity and brevity (in that order). Please see the course website for submission instructions and collaboration policy. Cite any sources that you use.

Remember that when you design an algorithm, in addition to an appropriate description of your algorithm, you must provide sufficient explanation for its correctness and analyze its running time. See the Homework Advice document on the course website for details.

In your submission, please start each question on a new page.

Question 1 (25 points)

In this problem we revisit the knapsack problem. Recall that while there exists a pseudo-polynomial time solution to the knapsack problem, the problem is NP-hard. Due to its hardness it is worth studying algorithms to efficiently find an approximation of the optimal solution. Here we will work through deriving a greedy approximate solution to the knapsack problem and then prove an upper bound on its error.

We are given a knapsack of capacity W (where W is an integer) and n items. Item i has value v_i and weight $w_i \leq W$. v_i and w_i are integers. We consider two variants of the knapsack problem: 0-1 knapsack and fractional knapsack.

In 0-1 knapsack, we pick a subset S of the items such that $\sum_{i \in S} w_i \leq W$. Our goal is to maximize the value of the items we picked: $\sum_{i \in S} v_i$. In fractional knapsack, for each item i, we can take a fraction $0 \leq \alpha_i \leq 1$ of that item, as long as $\sum_{i=1}^{n} \alpha_i w_i \leq W$. Our goal is to maximize $\sum_{i=1}^{n} \alpha_i v_i$. Note that 0-1 knapsack is the same as fractional knapsack when the fractions α_i are either 0 or 1.

(a) (7 points) Design an efficient greedy algorithm for solving fractional knapsack optimally.

(b) (6 points) For 0-1 knapsack, consider the greedy algorithm that iterates through the items in the same order as your algorithm from part (a), and adds to the knapsack any item that the knapsack has enough free capacity for. Provide an example that shows that this algorithm is not optimal. In your example, use no more than 3 items.

(c) (2 points) Show that the optimal solution to fractional knapsack has at least the same value as the optimal solution to 0-1 knapsack.

(d) (10 points) Consider algorithms for 0-1 knapsack that return the better of (i) the output of the algorithm from part (b), and (ii) one item. Show that there is an algorithm of that form that returns a solution with value at least 1/2 of the optimal solution to 0-1 knapsack.

Question 2 (20 points)

Consider a distribution over n possible outcomes, with probabilities $p_1, p_2, \ldots, p_n$.

(a) (10 points) Just for this part of the problem, assume that each p_i is a power of 2 (that is, $p_i = 2^{-k}$). Suppose a long sequence of m samples is drawn from the distribution and that for all $1 \leq i \leq n$, the i^{th} outcome occurs exactly mp_i times in the sequence. Show that if Huffman encoding is applied to this sequence, the resulting encoding will have length

$$\sum_{i=1}^{n} mp_i \log \frac{1}{p_i}$$

Hint: Prove by induction.

(b) (10 points) Now consider arbitrary distributions—that is, the probabilities p_i are not restricted to powers of 2. The most commonly used measure of the *amount of randomness* in the distribution is the *entropy* which has the following form:

$$\sum_{i=1}^{n} p_i \log \frac{1}{p_i}$$

(If $p_i = 0$, we replace the term $p_i \log \frac{1}{p_i}$ with 0.) For what distribution over n outcomes (that is, probabilities $p_1, \ldots, p_n$) is the expression the largest possible? The smallest possible?

Note that part (a) can be interpreted as follows: The length of the Huffman encoding of a random string depends on its entropy (that is, on how predictable the string is).

Hint: For $p, q > 0$, $p \log \frac{1}{p} + q \log \frac{1}{q} \leq (p+q) \log \frac{2}{p+q}$, and this inequality is tight if and only if $p = q$.

Question 3 (25 points)

We have n identical machines and m factories. We would like to assign machines to factories in order to maximize the output produced, however the amount of output produced at each factory is different and depends on the number of machines at that factory. For example, if factory i has j machines, then the output produced at factory i is O_i^j. It is possible to have a factory with 0 machines, and $O_i^0 = 0$ for all i. The output produced by factory i is non-decreasing in the number of machines in the factory, namely, for every i, j, $O_i^{j+1} \geq O_i^j$.

(a) (10 points) Design an efficient algorithm for determining the number of machines we should place in each factory.

(b) (15 points) Now we are also guaranteed that $O_i^{j+1} - O_i^j \leq O_i^j - O_i^{j-1}$, or in other words, the added value of each machine we add to a factory is no more than that of the previous machine. Given this constraint, provide an efficient algorithm for determining how many machines we should place in each factory. Note that you should be able to find a more efficient algorithm than in part (a).

Hint: The algorithm may use a priority queue.

Question 4 (10 points)

Given a connected, undirected graph $G = (V, E)$ in which each edge e has positive edge weight $w(e)$, we wish to compute a spanning tree $T = (V, E_T)$ such that the *product* of all its edge weights is *maximized.* In other words, the spanning tree T maximizing $\prod_{e \in E_T} w(e)$.

Suppose we have a black box algorithm $\mathcal{A}$ that solves the MST problem. Use $\mathcal{A}$ to solve this new problem, using only $O(|E|)$ additional time (that is, not including the call to $\mathcal{A}$).

Question 5 (20 points)

Imagine a set of n points $X = \{x_1, \ldots, x_n\}$ in some unknown space. The only thing we know about them is the distance $d(x, y)$ between each pair of points x and y.

We would like to compute a partition of X into disjoint sets of points $C_1, \ldots, C_k$ so as to maximize the minimum distance between any two points in different clusters. Each cluster must contain at least one point, and their union is the set X. More formally, we want a partition $\{C_1, \ldots, C_k\}$ of X that achieves

$$\max_{C_1,\ldots,C_k} \min_{\substack{x \in C_i \\ y \in C_j \\ i \neq j}} d(x, y)$$

Give an efficient algorithm for this task.

Hint: Show that you can obtain such a partition from the MST.