Homework Assignment #3

Note

This assignment is due 2:10PM Monday, October 15, 2018. Please write or type your answers on A4 (or similar size) paper. Drop your homework by the due time in the Algorithms box placed in the IM office on the 7-th floor of Management College Building 1. Late submission will be penalized by 20% for each working day overdue. You may discuss the problems with others, but copying answers is strictly forbidden.

Problems

There are five problems in this assignment, each accounting for 20 points. (Note: problems marked with "(X.XX)" are taken from [Manber 1989] with probable adaptation.)

1. (5.3) Consider algorithm *Mapping* (see slides). Is it possible that the set S will become empty at the end of the algorithm? Show an example, or prove that it cannot happen.

2. (5.7) Write a program (or modify the code discussed in class) to recover the solution (i.e., enumerate the elements in the solution) to a knapsack problem using the *belong* flag. You should make your algorithm as efficient as possible.

3. (5.12) Let x_1, x_2, ..., x_n be a sequence of real numbers (not necessarily positive). Design an $O(n)$ algorithm to find the subsequence x_i, x_{i+1}, ..., x_j (of consecutive elements) such that the product of the numbers in it is maximum over all consecutive subsequences. The product of the empty subsequence is defined as 1.

4. (5.17) The Knapsack Problem that we discussed in class is defined as follows: Given a set S of n items, where the ith item has an integer size $S[i]$, and an integer K, find a subset of the items whose sizes sum to exactly K or determine that no such subset exists.

   We have described in class an algorithm to solve the problem. Modify the algorithm to solve a variation of the knapsack problem where each item has an *unlimited* supply. In your algorithm, please change the type of $P[i,k].belong$ into integer and use it to record the number of copies of item i needed.

5. (5.20) Let x_1, x_2, ..., x_n be a set of integers, and let $S{=}\sum_{i=1}^{n} x_i$. Design an algorithm to partition the set into two subsets of equal sum, or determine that it is impossible to do so. The algorithm should run in time $O(nS)$.