Homework Assignment #4

Note

This assignment is due 2:10PM Thursday, April 7, 2011. Please write or type your answers on A4 (or similar size) paper. Drop your homework by the due time in Yih-Kuen Tsay's mail box on the first floor of Management College Building II. 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.

There are five problems in this assignment, each accounting for 20 points. There is also a bonus problem, which is worth 20 points.

Problems

(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. (5.8) In algorithm *Knapsack*, we first checked whether the ith item is unnecessary (by checking $P[i-1,j]$). If there is a solution with the $i-1$ items, we take this solution. We can also make the opposite choice, which is to take the solution with the ith item if it exists (i.e., check $P[i, j-k_i]$ first). Which version do you think will have a better performance? Redraw Fig. 5.11 (see slides) to reflect this choice.

3. (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.

4. (5.20) Let x_1, x_2, $\ldots$, 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 $O(nS)$.

5. (5.22) In the **towers of Hanoi** puzzle, there are three pegs A, B, and C, with n (generalizing the original eight) disks of different sizes stacked in decreasing order on peg A. The

objective is to transfer all the disks on peg A to peg B, moving one disk at a time (from one peg to one of the other two) and never having a larger disk stacked upon a smaller one.

(a) Give an algorithm to solve the puzzle. Explain how induction works here.

(b) Compute the total number of moves in the algorithm. Show the details of your calculation.

6. (5.23; bonus) Write a non-recursive program (in suitable pseudo code) that prints the moves of the solution to the towers of Hanoi puzzle.