CS 332 Homework Assignment 7

Fall, 2002

This assignment is not a programming assignment. You can either write up your solutions to these problems using pen and paper or you can produce an electronic document using programs like MS Word, LaTeX, or Maple (for example, this document was created using Maple). Whatever you turn in should be neat, readable, and well organized. This assignment is due on Friday, December 6.

Problem 1. Suppose that in a 0-1 knapsack problem, the order of the items when sorted by increasing weight is the same as their order when sorted by decreasing value. Give an efficient algorithm to find an optimal solution to this variant of the knapsack problem, and argue that your algorithm is correct.

Problem 2. Suppose that in a 0-1 knapsack problem, all of the items have the same value to weight ratio, and when the items are sorted by increasing weight, the weights are superincreasing, that is, each weight is larger than the sum of the preceding weights. Give an efficient algorithm to find an optimal solution to this variant of the knapsack problem, and argue that your algorithm is correct.

Problem 3. For the 0-1, integer knapsack problem, suppose that we have n items, the weights of the items are in the array $w[1 .. n]$, the values of the items are in the array $v[1..n]$, and the capacity of the knapsack is W. Define the two dimensional array $KV[0..n, 0..W]$ so that the entry $KV[i,j]$ represents the optimal value of the knapsack of capacity j using only items with index less than or equal to i. The entries of KV satisfy the following dynamic programming recurrence relation.

$$KV[i,j] = \begin{cases} KV[i-1,j] & j < w[i] \\ \max(KV[i-1,j],\ v[i] + KV[i-1,j-w[i]]) & w[i] \le j \end{cases}$$

The boundary conditions for the array KV are $KV[0,j] = 0$ for j from 0 to W, and $KV[i,0] = 0$ for i from 0 to n. The optimal solution of the knapsack problem is array entry $KV[n,W]$.

Here is an algorithm that computes the entries in the array *KV*.

```
for (i=0; i<=n; i++)
   KV[i,0] = 0;
for(j=0; j<=W; j++)
   KV[0,j] = 0;
for(i=1; i<=n; i++)
   for(j=1; j<=W; j++)
      if (w[i] > j)
         KV[i,j] = KV[i-1,j];
      else
         KV[i,j] = max( KV[i-1,j],  v[i] + KV[i-1, j-w[i]] );
```

Modify this algorithm so that the resulting pseudocode not only computes the entries in the array *KV*, and hence the optimal value of the knapsack, but so that the pseudocode also computes the list of items that make up the optimal load in the knapsack.

Problem 4. Do Exercise 6-2, page 160, from the course textbook.

Problem 5. Do Exercise 6-5 Part (a), page 160, from the course textbook.