**CMPSC 623 Problem Set 7. Solution.**
**by Prof. Honggang Zhang**

**Out: November 30, 2006**
**Due: December 7, 2006, before class.**

**Problem.** In this problem, we consider some generalizations of the knapsack problem.

1. Describe how to change the knapsack algorithm described in class to deal with the case where the value of an item can be different from its weight.
   **Solution:**
   If the value of an item can be different from its weight, our dynamic programming algorithm runs as follows.

$$i \geq 1 : knap(i,j) = \begin{cases} max\{knap(i-1,j), knap(i-1,j-w_i)+v_i\} & w_i \leq j \\ knap(i-1,j) & w_i > j \end{cases}$$

$$i = 0 : knap(i,j) = 0$$

   $knap(i,j)$ denotes the optimal value selected from the first $i$ items with weight capacity $j$. Compared with the algorithm from the lecture, here we only gain a value of $v_i$ instead of $w_i$ when we select item $i$.
2. Consider a knapsack problem where each item, in addition to having a weight $w_i$ and a value $v_i$, also has a size $s_i$. The knapsack has a weight capacity $W$ and also a size capacity $S$. Give an algorithm that runs in time $O(nWS)$ for determining the maximum value subsets of the items that can be placed in the knapsack without violating the weight capacity or the size capacity.
   **Solution:**
   To take item size into account, we need to construct and look up a 3-dimensional $n \times W \times S$ table $knap(i,j,k)$. It follows that the running time is $O(nWS)$. Our dynamic programming algorithm runs as follows.

$$i \geq 1 : knap(i,j,k) = \begin{cases} max\{knap(i-1,j,k), knap(i-1,j-w_i,k-s_i)+v_i\} & w_i \leq j \text{ and } s_j \leq k \\ knap(i-1,j,k) & w_i > j \text{ or } s_i > k \end{cases}$$

$$i = 0 : knap(i,j,k) = 0$$

   Compared with the algorithm from the lecture, here we can select item $i$ only if $w_i$ does not exceed currently available weight capacity $j$ and $s_i$ does not exceed currently available size capacity $k$. If we select item $i$, the maximum value will be $v_i$ plus the maximum value selected from the first $i-1$ items with weight capacity $j-w_i$ and size capacity $k-s_i$; if we do not select item $i$, the optimal solution will be selected from the first $i-1$ items with weight capacity $j$ and size capacity $k$. The optimal solution of the whole problem is the better one of these two.

3. Consider a scenario where the items to be stolen by the thief are partitioned into types (i.e., stereos, computers, necklaces... ). Every item belongs to exactly one type. In addition to the weight capacity of the knapsack, the thief is further constrained to be able to steal at most one item of each type. Give an algorithm, that runs in time $O(nW)$, that determines the optimal subset of items that the thief is able to steal. You can assume that each item comes with an indication of which type it belongs to, and that the items start off sorted by item type. For this subproblem, there is no longer the size consideration introduced in part (b).

**Solution:**

Suppose the items are partitioned into $m$ types and each type is assigned an integral *type ID* between 1 and $m$. Define function $f : Z_n \rightarrow Z_m$ such that $f(i)$ is the type ID of item $i$. This takes $O(n)$ time. Items are sorted by their type IDs. We define an array $first[1..m]$ such that $first[k]$ records the index of the first item of type $k$ in the sorted list. This takes $O(n)$ time. To enforce that at most one item of each type can be selected, our dynamic programming algorithm runs as follows.

$$i \geq 1 : knap(i,j) = \begin{cases} max\{knap(i-1,j), knap(first[f(i)]-1, j-w_i)+v_i\} & w_i \leq j \\ knap(i-1,j) & w_i > j \end{cases}$$

$$i = 0 : knap(i,j) = 0$$

Here we are filling the same $n \times W$ table. So the running time is still $O(nW)$.