The Subset Sum Problem

**Input** A set of $n$ items $\{1, \ldots, n\}$, where item $i$ has nonnegative integer weight $w(i)$, and a bound $W$.

**Output** A subset $S$ of the items with maximum total weight which is $\leq W$. That is, $S \subseteq \{1, \ldots, n\}$ so that

$$\sum_{i \in S} w(i) \leq W$$

and $\sum_{i \in S} w(i)$ is maximum.

Notice that a subproblem is defined by two parameters: the set of items and the bound on the total weight. Here are the four steps of designing an algorithm using the Dynamic Programming technique for this problem:

1. Let $A[i, m]$: total weight of an optimal solution for the subproblem defined by the set of items $\{1, \ldots, i\}$ and bound $m$. (Here $0 \leq i \leq n$, $0 \leq m \leq W$.)
2. Initial values and recurrence:

   $$A[0, m] = 0 \text{ for } m \leq W,$$

   $$A[i, m] = \begin{cases} 
   A[i - 1, m] & \text{if } w(i) > m \\
   \max\{A[i - 1, m], w(i) + A[i - 1, m - w(i)]\} & \text{otherwise}
   \end{cases}$$

3. Program:

   1. For $m = 0$ to $W$ do $A[0, m] \leftarrow 0$
   2. For $i = 1$ to $n$ do
   3. For $m = 0$ to $W$ do
   4. If $w(i) > m$ then $A[i, m] \leftarrow A[i - 1, m]$
   5. Else $A[i, m] \leftarrow \max\{A[i - 1, m], w(i) + A[i - 1, m - w(i)]\}$
   6. End If
   7. End For
   8. End For

4. Computing an optimal solution using $A$:

   1. $S \leftarrow \emptyset$ % solution
   2. $i \leftarrow n$, $m \leftarrow W$
   3. While $i \leq 0$ do
   4. If $A[i, m] = A[i - 1, m]$ then $i \leftarrow i - 1$
   5. Else
   6. $S \leftarrow S \cup \{i\}$
   7. $i \leftarrow i - 1$
8. \( m \leftarrow m - w(i) \)
9. End If
10. End While
11. Return \( S \).

**Knapsack Problem**

**Input** A set of \( n \) items \( \{1, \ldots, n\} \), where item \( i \) has nonnegative integer weight \( w(i) \) and a value \( v(i) \); and a bound \( W \).

**Output** A subset \( S \) of the items with maximum total value, whose total weight is \( \leq W \). That is, \( S \subseteq \{1, \ldots, n\} \) so that

\[
\sum_{i \in S} w(i) \leq W
\]

and \( \sum_{i \in S} v(i) \) is maximum.

**Observation** The Subset Sum Problem is a special case of the Knapsack Problem: We obtain the former from the latter by assigning \( v(i) = w(i) \) for each item \( i \).

The algorithm for this more general problem turns out to be quite similar to the previous algorithm. In fact, only a few slight changes are needed.

First, an element \( A[i, m] \) of the array \( A \) should now contain the total value (as opposed to the total weight) of an optimal solution for the problem defined by items \( \{1, \ldots, i\} \) and bound \( m \).

**Exercise** Complete the next three steps of designing an algorithm for the Knapsack Problem using the Dynamic Programming technique.

**Scheduling Jobs with Deadlines, Durations and Profits**

**Input** \( n \) jobs \( \{J_1, \ldots, J_n\} \), where job \( J_i \) has deadline \( d_i > 0 \), duration \( t_i \) and profit \( w_i \) (here \( d_i \) and \( t_i \) are positive integers, and \( w_i > 0 \)).

**Output** A schedule with maximum total profit.

Here a schedule is just a sequence of jobs that meet their deadlines if they are processed in that order. In other words, a schedule is a sequence

\[
J_{s_1}, \ldots, J_{s_k}
\]

so that

\[
t_{s_1} + \ldots + t_{s_i} \leq d_i
\]

for all \( 1 \leq i \leq k \). (Condition (1) says that job \( J_{s_i} \) meets its deadline.)

**Claim** This problem is more general than the Knapsack Problem: We obtain the Knapsack Problem by letting (i) the jobs have common deadline: \( d_i = W \) for all \( 1 \leq i \leq n \); (ii) the durations \( t_i = w(i) \); and (iii) the profits \( w_i = v(i) \).

**Exercise** Verify the Claim.

**Designing a Recursive Algorithm**
Consider an optimal solution $OPT$. For each job $J_i$, there are two cases: either $J_i \in OPT$ or $J_i \not\in OPT$. For the case where $J_i \in OPT$, the other jobs in $OPT$ cannot overlap with $J_i$. In particular, all other jobs in $OPT$ must either finish before $J_i$ starts, or start after $J_i$ finishes. This suggests that in defining the (sub)problems, we might need to specify the period of time where the jobs can be scheduled. (For the original problem, the period of time is from 0 to $max\{d_1, \ldots, d_n\}$.)

Note that the time periods may not start from 0 (e.g. after some jobs $J_i$ has finished). However, if we sort the jobs in increasing order of deadlines, then all time periods start from 0. The reason is as follows. Suppose that the jobs have been sorted so that

$$d_1 \leq d_2 \leq \ldots \leq d_n$$

Consider the job $J_n$: if $J_n$ belongs to OPT, then we can assume that $J_n$ is the last job in OPT and that $J_n$ finishes at exactly time $d_n$. In this case, the subproblem that we need to solve is that of scheduling the jobs $J_1, \ldots, J_{n-1}$, with the (possibly new) deadlines $min\{d_n - t_n, d_i\}$ for each job $J_i$ $(1 \leq i \leq n-1)$.

With the above ordering of the jobs, the array $A[i,t]$ is defined so that $A[i,m]$ is the total profit of an optimal schedule of the jobs $\{1, \ldots, i\}$ in the time period from 0 to $t$. Here $0 \leq i \leq n$, $0 \leq t \leq d_n$.

Initial values and recurrence:

$$A[0,t] = 0 \text{ for } t \leq d_n,$$

$$A[i,t] = \begin{cases} A[i-1,t] & \text{if } t < t_i \\ \max\{A[i-1,t], w_i + A[i-1,\min\{t, d_i\} - t_i]\} & \text{otherwise} \end{cases}$$

The next two steps (write down the program to compute $A$, and compute an optimal schedule using $A$) are left as an Exercise.

**Exercise** Carry out the next two steps to obtain a complete algorithm for solving the problem.

**Matrix Chain Multiplication Problem**

**Input** Given a sequence of matrices $A_1, \ldots, A_n$, where $A_i$ has dimension $d_i \times d_{i+1}$, for $1 \leq i \leq n$.

**Output** An order of multiplying the matrices with smallest number of operations.

(Here multiplying $A_i$ and $A_{i+1}$ requires $\Theta(d_i \times d_{i+1} \times d_{i+2}$ operations.)

(To be continued next week.)