Algorithms – CS-27200/CS-37000
Dynamic programming: the knapsack problem

The input of the "Knapsack Problem" is a list of weights $w_1, \ldots, w_n$, a list of values $v_1, \ldots, v_n$, and a weight limit W. All these are positive reals.

The problem is to find a subset $S \subseteq \{1, \ldots, n\}$ such that the following *constraint* is observed:

$$\sum_{k \in S} w_k \leq W. \tag{1}$$

The *objective* is to maximize the total value under this constraint:

$$\max \leftarrow \sum_{k \in S} v_k. \tag{2}$$

Theorem. Under the assumption that the weights are *integers* (but the values are real), one can find the optimum in $O(nW)$ operations (arithmetic, comparison, bookkeeping).

The solution illustrates the method of "dynamic programming." The idea is that rather than attempting to solve the problem directly, we embed the problem in an $n \times W$ array of problems, and solve those problems successively.

For $0 \leq i \leq n$ and $0 \leq j \leq W$, let $m[i, j]$ denote the maximum value of the knapsack problem restricted to $S \subseteq \{1, \ldots, i\}$, under weight limit j.

The heart of the solution is the following recurrence.

$$m[i, j] = \max\{m[i-1, j], \quad v_i + m[i-1, j-w_i]\}. \qquad \heartsuit$$

Explanation: if in the optimal solution $i \notin S$ then $m[i, j] = m[i-1, j]$; otherwise we gain value v_i and have to maximize from the remaining objects under the remaining weight limit $j - w_i$ (assuming $j \geq w_i$). The optimum will be the bigger of these two values.

It should also be clear that $m[0, k] = m[k, 0] = 0$ for all $k \geq 0$. With this initialization, a double **for**-loop fills in the array of values $m[i, j]$:

Initialize (lines 1–6):
1 **for** $i = 0$ to n
2 $\quad\quad m[i, 0] := 0$
3 **end**
4 **for** $j = 1$ to W
5 $\quad\quad m[0, j] := 0$
6 **end**
Main loops:
7 **for** $i = 1$ to n
8 $\quad$ **for** $j = 1$ to W
9 $\quad\quad$ **if** $j < w_i$ **then** $m[i, j] := m[i-1, j]$ (* item i cannot be selected *)
10 $\quad\quad$ **else** $m[i, j] :=$ as in equation $\heartsuit$ (* heart of solution *)
11 $\quad$ **end**
12 **end**

The statement inside the inner loop expresses the value of the next $m[i, j]$ in terms of values already known so the program can be executed.

The required optimum is the value $m[n, W]$. Evaluating equation $\heartsuit$ requires a constant number of operations per entry, justifying the $O(nW)$ claim.