Dynamic Programming

Algorithms: Design and Analysis, Part II

An Algorithm for the Knapsack Problem

Recurrence from Last Time

Notation: Let $V_{i,x}$ = value of the best solution that:

(1) uses only the first i items

(2) has total size $\leq x$

Upshot from last video: For $i \in \{1, 2, \ldots, n\}$ and only x,

$$V_{i,x} = \max\{V_{(i-1),x} \text{ (case 1, item } i \text{ excluded)},$$
$$v_i + V_{(i-1),x-w_i} \text{ (case 2, item } i \text{ included)}\}$$

Edge case: If $w_i > x$, must have $V_{i,x} = V_{(i-1),x}$

The Subproblems

Step 2: Identify the subproblems.

- All possible prefixes of items $\{1, 2, \ldots, i\}$
- All possible (integral) residual capacities $x \in \{0, 1, 2, \ldots, W\}$

Recall W and the w_i's are integral

Step 3: Use recurrence from Step 1 to systematically solve all problems.

Let A = 2-D array
Initialize $A[0, x] = 0$ for $x = 0, 1, \ldots, W$
For $i = 1, 2, \ldots, n$
    For $x = 0, 1, \ldots, W$
        $A[i, x] := \max\{ A[i-1, x] , A[i-1, x - w_i] + v_i \}$
Return $A[n, W]$

Previously computed, available for $O(1)$-time lookup. Ignore second case if $w_i > x$.

Running Time

Question: What is the running time of this algorithm?

A) $\Theta(n^2)$

B) $\boxed{\Theta(nW)}$ ($\Theta(nW)$ subproblems, solve each in $\Theta(1)$ time)

C) $\Theta(n^2 W)$

D) $\Theta(2^n)$

Correctness: Straightforward induction [use step 1 argument to justify inductive step]