Dynamic Programming

The Knapsack Problem

Algorithms: Design and Analysis, Part II

Problem Definition

Input: n items. Each has a value:

- Value v_i (nonnegative)
- Size w_i (nonnegative and <u>integral</u>)
- Capacity W (a nonnegative integer)

Output: A subset $S \subseteq \{1, 2, \ldots, n\}$ that maximizes $\sum_{i \in S} v_i$ subject to $\sum_{i \in S} w_i \leq W$.

Developing a Dynamic Programming Algorithm

Step 1: Formulate recurrence [optimal solution as function of solutions to "smaller subproblems"] based on a structure of an optimal solution.

Let S = a max-value solution to an instance of knapsack.

Case 1: Supose item $n \notin S$.
$\Rightarrow S$ must be optimal with the first $n-1$ items (same capacity W)
[If S^* were better than S with respect to 1st $n-1$ items, then this equally true w.r.t. all n items - contradiction]

Optimal Substructure

Case 2: Suppose item $n \in S$. Then $S - \{n\}$ ...

A) is an optimal solution with respect to the 1st $n - 1$ items and capacity W.

B) is an optimal solution with respect to the 1st $n - 1$ items and capacity $W - v_n$.

C) is an optimal solution with respect to the 1st $n - 1$ items and capacity $W - w_n$.

D) might not be feasible for capacity $W - w_n$.

Proof: If S^* has higher value than $S - \{n\}$ + total size $\leq W - w_n$, then $S^* \cup \{n\}$ has size $\leq W$ and value more than S [contradiction]

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 $x = 0, 1, \ldots, W$
        $A[i, x] := \max\{\boxed{A[i-1, x]}, \boxed{A[i-1, x - w_i] + v_i}\}$
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) $\Theta(n^2W)$

D) $\Theta(2^n)$

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

The Knapsack Problem

An Example

Algorithms: Design and Analysis, Part II

Example ($n = 4$, $W = 6$)

Initialization: $A[0, x] = 0$ for all x

Main loop:

For $i = 1, \ldots, n$

  For $x = 0, \ldots, W$

    $A[i, x] := \max\{A[i-1, x], A[i-1, x - w_i] + v_i\}$

Example:

$W = 6$

$v_1 = 3, w_1 = 4$

$v_2 = 2, w_2 = 3$

$v_3 = 4, w_3 = 2$

$v_4 = 4, w_3 = 3$

6	0	3	3	7	8
5	0	3	3	6	8
4	0	3	3	4	4
3	0	0	2	4	4
2	0	0	0	4	4
1	0	0	0	0	0
$x =$ 0	0	0	0	0	0
	$i =$ 0	1	2	3	4

Optimal value = 8

Optimal solution = {item 3, item 4}