Approximation Algorithms for NP-Complete Problems

Dynamic Programming for Knapsack, Revisited

Algorithms: Design and Analysis, Part II

Two Dynamic Programming Algorithms

Dynamic programming algorithm #1 (see earlier video)

1. Assume sizes w_i and capacity W are integers
2. Running time $= O(nW)$

Dynamic programming algorithm #2 (this video)

1. Assume values v_i are integers
2. Running time $= O(n^2 v_{max})$, where $v_{max} = \max_i v_i$

The Subproblems and Recurrence

Subproblems: For $i = 0, 1, 2, \ldots, n$ and
$x = 0, 1, 2, 3, \ldots, n \cdot v_{max}$

define $S_{i,x}$ = minimum total size needed to achieve value $\geq x$ while using only the first i items. (or $+\infty$ if impossible)

Recurrence: $(i \geq 1)$

$$S_{i,x} = \min \begin{cases} S_{(i-1),x} & \text{Case 1, item } i \text{ not used in optimal solution} \\ w_i + S_{(i-1),(x-v_i)} & \text{Case 2, item } i \text{ used in optimal solution} \end{cases}$$

interpret as 0 if $v_i \geq x$

The Algorithm

Let A = 2-D array [indexed by $i = 0, 1, 2, \ldots, n$ and $x = 0, 1, 2, \ldots, n v_{max}$]

Base case: $A[0, x] = \begin{cases} 0 & \text{if } x = 0 \\ +\infty & \text{otherwise} \end{cases}$

For $i = 1, 2, 3, \ldots, n$:
  For $x = 0, 1, 2, \ldots, n \cdot v_{max}$:
    $A[i, x] = \min\{A[i-1, x], w_i + \underbrace{A[i-1, x-v_i]}_{\text{interpret as 0 if } v_i \geq x}\}$

$n^2 v_{max}$ iterations

$O(1)$ work per iteration

Return the largest x such that $A[n, x] \leq W$. ← $O(n v_{max})$

Running time: $O(n^2 v_{max})$.