CS F364
Design & Analysis of Algorithms

Algorithm Design Techniques

Dynamic Programming and Optimal Sub-Structure Property

- Example: 0/1 Knapsack: Dynamic Programming Algorithm

Dynamic Programming

- Steps:
  - Characterize and define the solution in terms of a recurrence relation.
    - Verify whether optimal substructure property holds.
  - Write down the steps for bottom-up computation of the recurrence relation.
  - Inspect the list of intermediate results required and prune unnecessary items

EXAMPLE – 0/1 KNAPSACK - DEFINITION

- Given:
  - A sack with max. capacity by weight: Wmax
  - Set S of items j (in store) labeled with
    Weight w_i (<= Wmax) and Price p_j
- Assumption:
  - An item is either taken (in full) or not
  - All values (w_j, p_j, and Wmax) are ***positive integers***
- Goal:
  - Fill the sack with maximum value (by price)
    - i.e. Find T subset of S, such that
    - $\Sigma_{i \text{ in } T}\, p_i$ is maximum and $\Sigma_{i \text{ in } T}\, w_i$ <= Wmax

Example – 0/1 Knapsack - Optimal Substructure

- The problem structure of 0/1 Knapsack can be defined as follows:
  - Let P(k,w) be the maximum profit obtainable from a subset { 1, 2, … k} weighing no more than w in total.
  - Then P(k,w) =
    - P(k-1, w) if $w_k > w$
    - max { P(k-1, w), P(k-1, $w-w_k$) + p_k } otherwise

EXAMPLE – 0/1 KNAPSACK – BOTTOM UP

- P(k,w) is
  - P(k-1, w) if $w_k > w$
  - max { P(k-1, w), P(k-1, w-w_k) + p_k } otherwise
- P(0, w) = 0 for all w
- P(k, 0) = 0 for all k

k ↓ w →	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0							
2	0							
3	0							
4	0							
5	0							

EXAMPLE – 0/1 KNAPSACK – BOTTOM-UP

- P(k,w) is
  - P(k-1, w) if $w_k > w$
  - max { P(k-1, w), P(k-1, $w-w_k$) + p_k } otherwise

Given weights { 3, 1, 5, … } and prices { 14, 7, 10, … }

- **Consider the subset { 3 }**

↓ k → w	0	1	2	3	4	5	6	7 …
0	0	0	0	0	0	0	0	0
1	0	0	0	14	14	14	14	14
2	0							
3	0							

EXAMPLE – 0/1 KNAPSACK – BOTTOM-UP

- P(k,w) is
  - P(k-1, w) if $w_k > w$
  - max { P(k-1, w), P(k-1, w-w_k) + p_k } otherwise

Given weights {3, 1, 5,… } and prices {14,7,10,… }

- **Consider the subset { 3, 1 }**

max(14, 0+7)

max(14, 14+7)

↓k w→	0	1	2	3	4	5	6	7 …
0	0	0	0	0	0	0	0	0
1	0	0	0	14	14	14	14	14
2	0	7	**7**	**14**	**21**	**21**	**21**	**21**
3	0							

EXAMPLE – 0/1 KNAPSACK – BOTTOM-UP

- P(k,w) is
  - P(k-1, w) if $w_k > w$
  - max { P(k-1, w), P(k-1, w-w_k) + p_k } otherwise

Given weights { 3, 1, 5, … } and prices { 14, 7, 10, … }

- **Consider the subset { 3, 1, 5}**

max(21, 14+10) max(21, 21+10)

↓ k w →	0	1	2	3	4	5	6	7	8	9	…	Wmax
0	0	0	0	0	0	0	0	0	0	0		
1	0	0	0	14	14	14	14	14	14	14		
2	0	7	7	14	21	21	21	21	21	21		
3	0	7	7	14	21	21	21	21	24	31		
…												
N												

EXAMPLE – 0/1 KNAPSACK – DP SOLUTION

Known (Atomic) Solutions: $P(0, w)=0$ forall w and $P(k, 0)=0$ forall k

Recursive structure: $P(k,w) =$

$$\begin{cases} P(k-1, w) & \text{if } w_k > w \\ \max\{ P(k-1, w), P(k-1, w-w_k) + p_k \} & \text{otherwise} \end{cases}$$

```
Profit(k,w)
 // assume output array Pf[0..N][0..Wmax]
//  assume array wt[1..N] of weights and p[1..N] of prices
 {   for (k=0; k<=N; k++)  Pf[k,0] = 0;
     for (w=0; w<=Wmax; w++) Pf[0,w] = 0;
     for (k = 1; k<=N; k++)
         for (w=1; w<=Wmax; w++)
            Pf[k,w] = (wt[k] > w) ? Pf[k-1,w] :
                              max(Pf[k-1,w], Pf[k-1,w-wt[k]]+p[k]);
}
```