

Algorithm Design and Analysis

CSE
565

LECTURE 21
Dynamic Programming
• Knapsack Problem

Sofya Raskhodnikova

10/15/2007

S. Raskhodnikova; based on slides by K. Wayne

CSE
565

Review questions

- Do Shortest Path and Longest Path have optimal substructure that can be used for a polynomial time dynamic programming algorithm?

10/15/2007

S. Raskhodnikova; based on slides by K. Wayne

CSE
565

Knapsack Problem

- Given n objects and a "knapsack."
- Item i weighs $w_i > 0$ kilograms and has value $v_i > 0$.
- Knapsack has capacity of W kilograms.
- Goal: fill knapsack so as to maximize total value.

- Ex: { 3, 4 } has value 40.

W = 11

#	value	weight
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

- Greedy: repeatedly add item with maximum ratio v_i / w_i .
- Ex: { 5, 2, 1 } achieves only value = 35 \Rightarrow greedy not optimal.

10/15/2007

S. Raskhodnikova; based on slides by K. Wayne

CSE
565

Dynamic programming: attempt 1

- Definition: $OPT(i) =$
maximum profit subset of items 1, ..., i .
- Case 1: OPT does not select item i .
 - OPT selects best of { 1, 2, ..., $i-1$ }
- Case 2: OPT selects item i .
 - without knowing what other items were selected before i , we don't even know if we have enough room for i
- Conclusion. Need more sub-problems!

10/15/2007

S. Raskhodnikova; based on slides by K. Wayne

CSE
565

Adding a new variable

- Definition: $OPT(i, w) =$ max profit subset of items 1, ..., i with weight limit w .
- Case 1: OPT does not select item i .
 - OPT selects best of { 1, 2, ..., $i-1$ } with weight limit w
- Case 2: OPT selects item i .
 - new weight limit = $w - w_i$
 - OPT selects best of { 1, 2, ..., $i-1$ } with new weight limit

$$OPT(i, w) = \begin{cases} 0 & \text{if } i = 0 \\ OPT(i-1, w) & \text{if } w_i > w \\ \max\{OPT(i-1, w), v_i + OPT(i-1, w-w_i)\} & \text{otherwise} \end{cases}$$

10/15/2007

S. Raskhodnikova; based on slides by K. Wayne

CSE
565

Bottom-up algorithm

- Fill up an n -by- W array.

```

Input: n, W, w1, ..., wn, v1, ..., vn
for w = 0 to W
  M[0, w] = 0
for i = 1 to n
  for w = 1 to W
    if (wi > w)
      M[i, w] = M[i-1, w]
    else
      M[i, w] = max {M[i-1, w], vi + M[i-1, w-wi]}
return M[n, W]

```

10/15/2007

S. Raskhodnikova; based on slides by K. Wayne

Knapsack table

		W + 1											
		0	1	2	3	4	5	6	7	8	9	10	11
n + 1	ϕ	0	0	0	0	0	0	0	0	0	0	0	0
	{1}	0	1	1	1	1	1	1	1	1	1	1	1
	{1, 2}	0	1	6	7	7	7	7	7	7	7	7	7
	{1, 2, 3}	0	1	6	7	7	18	19	24	25	25	25	25
	{1, 2, 3, 4}	0	1	6	7	7	18	22	24	28	29	29	40
	{1, 2, 3, 4, 5}	0	1	6	7	7	18	22	28	29	34	34	40

OPT: { 4, 3 }
value = 22 + 18 = 40

W - 11		
Item	Value	Weight
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

Time and space complexity

- **Time and space:** $\Theta(nW)$.
 - Not polynomial in input size!
 - "Pseudo-polynomial."
 - Decision version of Knapsack is NP-complete. [KT, chapter 8]
- **Knapsack approximation algorithm.** There is a poly-time algorithm that produces a solution with value within 0.01% of optimum. [KT, section 11.8]