Design and Analysis of Algorithms

Memory Function Knapsack

Reetinder Sidhu

Department of Computer Science and Engineering

Design and Analysis of Algorithms

Memory Function Knapsack

Reetinder Sidhu

Department of Computer Science and Engineering

- Dynamic Programming
  - Computing a Binomial Coefficient
  - **The Knapsack Problem**
  - Memory Functions
  - Warshall's and Floyd's Algorithms
- Limitations of Algorithmic Power
  - Lower-Bound Arguments
  - Decision Trees
  - P, NP, and NP-Complete, NP-Hard Problems
- Coping with the Limitations
  - Backtracking
  - Branch-and-Bound. Architecture (microprocessor instruction set)

Concepts covered

- Memory Function Knapsack
  - Motivation
  - Algorithm
  - Example

Memory Function Knapsack
Bottom Up Approach

Memory Function Knapsack
Bottom Up Approach

- Advantage of bottom up approach: each value computed only once

Memory Function Knapsack
Bottom Up Approach

- Advantage of bottom up approach: each value computed only once
- Example computed bottom up:

	capacity *j*				
i	1	2	3	4	5
1	0	12	12	12	12
2	10	12	22	22	22
3	10	12	22	30	32
4	10	15	25	30	37

Memory Function Knapsack
Bottom Up Approach

- Advantage of bottom up approach: each value computed only once
- Example computed bottom up:

	capacity j				
i	1	2	3	4	5
1	0	12	12	12	12
2	10	12	22	22	22
3	10	12	22	30	32
4	10	15	25	30	37

- Disadvantage of bottom up approach: values not required also computed

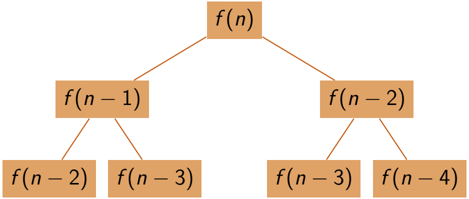
Memory Function Knapsack
Top Down Approach

Memory Function Knapsack
Top Down Approach

- Disadvantage of top down approach: same problem solved multiple times

Memory Function Knapsack
Top Down Approach

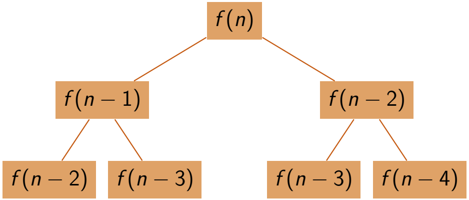
- Disadvantage of top down approach: same problem solved multiple times
- Example computed top down:



Memory Function Knapsack

Top Down Approach

- Disadvantage of top down approach: same problem solved multiple times
- Example computed top down:



- Advantage of top down approach: only the required subproblems solved

Memory Function Knapsack

Memory Function Dynamic Programming

- Combine the advantages of bottom up and top down approaches:
  - compute each subproblem only once
  - compute only the required subproblems

MF-DP Algorithm

Algorithm for Memory Function Dynamic Programming

procedure MFKnapsack(i, j)
    ▷ **Inputs:** i indicating the number items, and
    ▷ j, indicating the knapsack capacity
    ▷ **Output:** The value of an optimal feasible subset of the first i items
    ▷ **Note:** Uses global variables input arrays $Weights[1 \dots n]$, $Values[1 \dots n]$,
    ▷ and table $F[0 \dots n, 0 \dots W]$ whose entries are initialized with −1's except
    ▷ row 0 and column 0 is initialized with 0
    if $F[i, j] < 0$ **then**
        if $j < Weights[i]$ **then**
            $value \leftarrow MFKnapsack(i-1, j)$
        else $value \leftarrow max(MFKnapsack(i-1, j),\ \ Values[i] + MFKnapsack(i-1, j - Weights[i]))$
            $F[i, j] \leftarrow value$
    return $F[i, j]$

Memory Function Knapsack
Example

$$F(i,j) = \begin{cases} \max(F(i-1,j),\ \ v_i + F(i-1,j-w_i)) & \text{if } j - w_i \geq 0 \\ F(i-1,j) & \text{if } j - w_i < 0 \end{cases}$$

Dynamic Programming Example

item i	weight w_i	value v_i
1	2	12
2	1	10
3	3	20
4	2	15

What is the maximum value that can be stored in a knapsack of capacity 5?

Memory Function Knapsack

Example

$$F(i,j) = \begin{cases} \max(F(i-1,j), \;\; v_i + F(i-1, j-w_i)) & \text{if } j - w_i \geq 0 \\ F(i-1,j) & \text{if } j - w_i < 0 \end{cases}$$

Dynamic Programming Example

item i	weight w_i	value v_i
1	2	12
2	1	10
3	3	20
4	2	15

What is the maximum value that can be stored in a knapsack of capacity 5?

	capacity j					
i	0	1	2	3	4	5
0	0					
1						
2						
3						
4						$\square$

Memory Function Knapsack
Example

$$F(i,j) = \begin{cases} \max(F(i-1,j),\ \ v_i + F(i-1,j-w_i)) & \text{if } j - w_i \geq 0 \\ F(i-1,j) & \text{if } j - w_i < 0 \end{cases}$$

Dynamic Programming Example

item i	weight w_i	value v_i
1	2	12
2	1	10
3	3	20
4	2	15

What is the maximum value that can be stored in a knapsack of capacity 5?

	capacity j					
i	0	1	2	3	4	5
0	0					
1						
2						
3						$\square$
4						$\square$

Memory Function Knapsack

Example

$$F(i,j) = \begin{cases} \max(F(i-1,j),\ \ v_i + F(i-1,j-w_i)) & \text{if } j - w_i \geq 0 \\ F(i-1,j) & \text{if } j - w_i < 0 \end{cases}$$

Dynamic Programming Example

item i	weight w_i	value v_i
1	2	12
2	1	10
3	3	20
4	2	15

What is the maximum value that can be stored in a knapsack of capacity 5?

	capacity j					
i	0	1	2	3	4	5
0	0					
1						
2						
3				□		□
4						□

Memory Function Knapsack

Example

$$F(i,j) = \begin{cases} \max(F(i-1,j), \;\; v_i + F(i-1,j-w_i)) & \text{if } j - w_i \geq 0 \\ F(i-1,j) & \text{if } j - w_i < 0 \end{cases}$$

Dynamic Programming Example

item i	weight w_i	value v_i
1	2	12
2	1	10
3	3	20
4	2	15

What is the maximum value that can be stored in a knapsack of capacity 5?

	capacity j					
i	0	1	2	3	4	5
0	0	□	□	□	□	□
1	□	□	□	□	□	□
2	□		□	□		□
3	□			□		□
4	□					□

Memory Function Knapsack
Example

$$F(i,j) = \begin{cases} \max(F(i-1,j),\ \ v_i + F(i-1, j-w_i)) & \text{if } j - w_i \geq 0 \\ F(i-1,j) & \text{if } j - w_i < 0 \end{cases}$$

Dynamic Programming Example

item i	weight w_i	value v_i
1	2	12
2	1	10
3	3	20
4	2	15

What is the maximum value that can be stored in a knapsack of capacity 5?

	capacity j					
i	0	1	2	3	4	5
0	0	□	□	□	□	□
1	□	□	□	□	□	□
2	□	–	□	□	–	□
3	□	–	–	□	–	□
4	□	–	–	–	–	□

Memory Function Knapsack

Example

$$F(i,j) = \begin{cases} \max(F(i-1,j),\ \ v_i + F(i-1,j-w_i)) & \text{if } j - w_i \geq 0 \\ F(i-1,j) & \text{if } j - w_i < 0 \end{cases}$$

Dynamic Programming Example

item i	weight w_i	value v_i
1	2	12
2	1	10
3	3	20
4	2	15

What is the maximum value that can be stored in a knapsack of capacity 5?

	capacity j					
i	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	12	12	12	12
2	0	–	12	22	–	22
3	0	–	–	22	–	32
4	0	–	–	–	–	37

Memory Function Knapsack
Example

$$F(i,j) = \begin{cases} \max(F(i-1,j),\ \ v_i + F(i-1,j-w_i)) & \text{if } j - w_i \geq 0 \\ F(i-1,j) & \text{if } j - w_i < 0 \end{cases}$$

Dynamic Programming Example

item i	weight w_i	value v_i
1	2	12
2	1	10
3	3	20
4	2	15

What is the maximum value that can be stored in a knapsack of capacity 5?

	capacity j					
i	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	12	12	12	12
2	0	–	12	22	–	22
3	0	–	–	22	–	32
4	0	–	–	–	–	37

Knapsack problem solved by

- computing 21 out 30 possible subproblems
- reusing subproblem entry $(1,2)$

Memory Function Knapsack
Complexity

- Constant factor imrpovement in efficiency
  - Space complexity: $\Theta(nW)$
  - Time complexity: $\Theta(nW)$
  - Time to compose optimal solution: $O(n)$
- Bigger gains possible where computation of a subproblem takes more than constant time

Memory Function Knapsack

Think About It

- Consider the use of the MF technique to compute binomial coefficient using the recurrence

$$C(n,k) = C(n-1,k-1) + C(n-1,k)$$

  - How many table entires are filled?
  - How many are reused?