UE19CS251
Design and Analysis of Algorithms
Unit 5: Limitations of Algorithmic Power and Coping with the Limitations

Memory Function Knapsack

PES University

Concepts covered

- Memory Function Knapsack
  - Motivation
  - Algorithm
  - Example

1 Bottom Up Approach

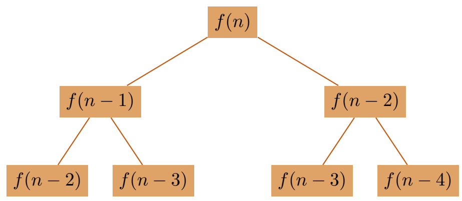
- Advantage of bottom up approach: each value computed only once
- Example computed bottom up:

	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

2 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

3 Memory Function Dynamic Programming

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

4 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$value \leftarrow max(MFKnapsack(i-1, j), \quad Values[i] + MFKnapsack(i-1, j-Weights[i]))$
      $F[i, j] \leftarrow value$
  return $F[i, j]$

5 Example

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

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 $(1, 2)$

6 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

7 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?