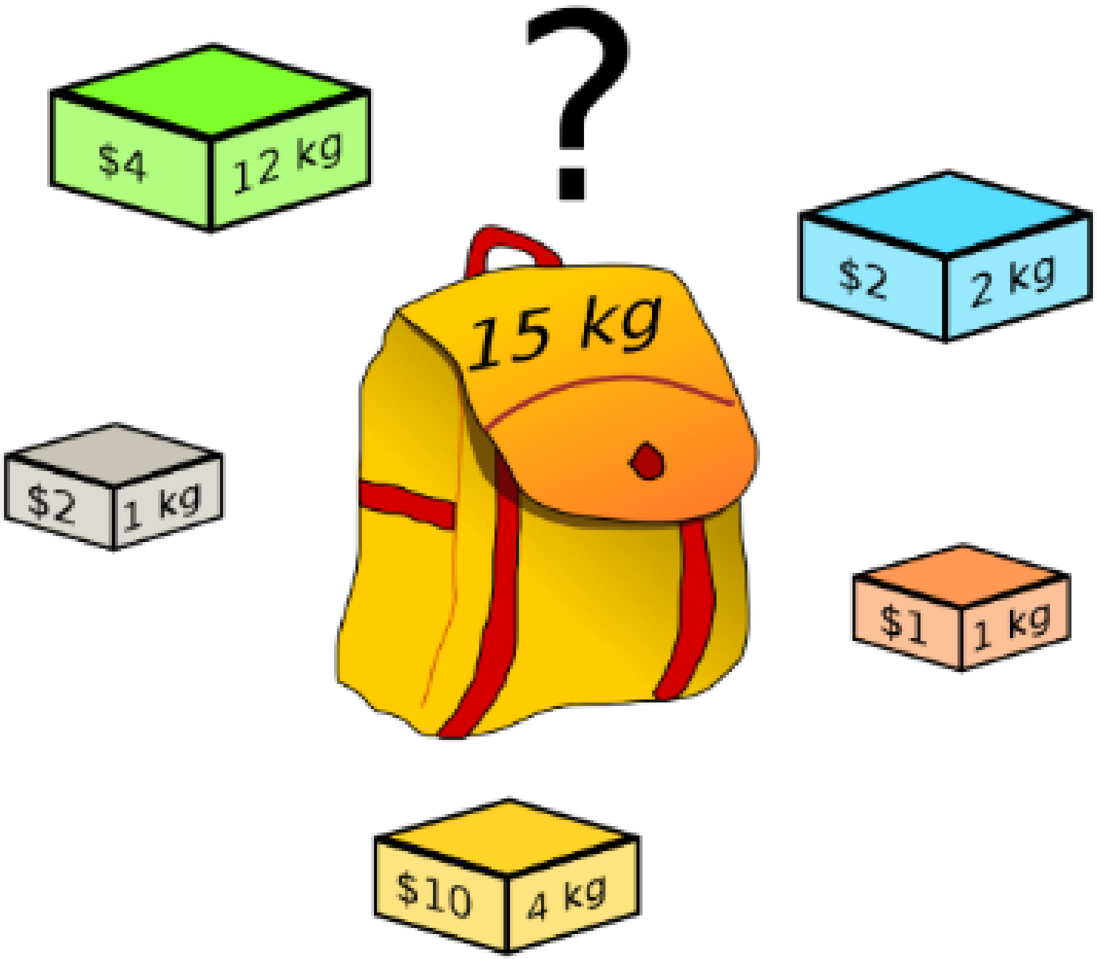
UNIT III DYNAMIC PROGRAMMING:

Dynamic Programming – Change-making Problem – Computing a Binomial Coefficient – All-pairs Shortest-paths Problem – Warshall's and Floyd's Algorithms – 0/1 Knapsack Problem



Knapsack Problem

The Knapsack Problem

The classic Knapsack problem is:

A thief breaks into a store and wants to fill his knapsack of capacity K with goods of as much value as possible.

Decision version: Does there exist a collection of items that fits into his knapsack and whose total value is >= W?

0-1 Knapsack: An item can either be picked or left. It cannot be picked partially. For example gold coins, diamond rings, TV etc.

0-1 Knapsack Problem

value[] = {60, 100, 120};
weight[] = {10, 20, 30};
W = 50;

Solution: 220

Weight = 10; Value = 60;
Weight = 20; Value = 100;
Weight = 30; Value = 120;
Weight = (20+10); Value = (100+60);
Weight = (30+10); Value = (120+60);
Weight = (30+20); Value = (120+100);
Weight = (30+20+10) > 50

Knapsack Problem by DP

Given n items of

integer weights: $w_1 \quad w_2 \quad \ldots \quad w_n$

values: $v_1 \quad v_2 \quad \ldots \quad v_n$

a knapsack of integer capacity W

find most valuable subset of the items that fit into the knapsack

Consider instance defined by first i items and capacity j ($j \leq W$).

Let $V[i,j]$ be optimal value of such instance. Then

$$V[i,j] = \begin{cases} \max \{V[i-1,j],\ v_i + V[i-1,j-w_i]\} & \text{if } j-w_i \geq 0 \\ V[i-1,j] & \text{if } j-w_i < 0 \end{cases}$$

Initial conditions: $V[0,j] = 0$ and $V[i,0] = 0$

Knapsack Problem by DP (pseudocode)

```
Algorithm DPKnapsack(w[1..n], v[1..n], W)
    var V[0..n,0..W], P[1..n,1..W]: int
    for j := 0 to W do
        V[0,j] := 0
    for i := 0 to n do
        V[i,0] := 0
    for i := 1 to n do
        for j := 1 to W do
            if w[i] ≤ j and v[i] + V[i-1,j-w[i]] > V[i-1,j] then
                V[i,j] := v[i] + V[i-1,j-w[i]]; P[i,j] := j-w[i]
            else
                V[i,j] := V[i-1,j]; P[i,j] := j
    return V[n,W] and the optimal subset by backtracing
```

Running time and space: O(nW).

Example: Knapsack of capacity $W = 5$

item	weight	value
1	2	$12
2	1	$10
3	3	$20
4	2	$15

capacity j

	i	0	1	2	3	4	5
	0	0	0	0			
$w_1 = 2, v_1 = 12$	1	0	0	12			
$w_2 = 1, v_2 = 10$	2	0	10	12	22	22	22
$w_3 = 3, v_3 = 20$	3	0	10	12	22	30	32
$w_4 = 2, v_4 = 15$	4	0	10	15	25	30	37

Backtracking finds the actual optimal subset, i.e. solution

Item #	Weight (Kg)	Value (Rs.)
1	2	3
2	3	4
3	4	5
4	5	6

<table>
<tr><th>Step</th><th>Calculation</th><th>Table</th></tr>
<tr><td>1</td><td>Initial conditions:
V[0, j] = 0 for j ≥ 0
V[i, 0] = 0 for i ≥ 0</td><td>
<table>
<tr><th>V[i,j]</th><th>j=0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th></tr>
<tr><td>i=0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>1</td><td>0</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>2</td><td>0</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>3</td><td>0</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>4</td><td>0</td><td></td><td></td><td></td><td></td><td></td></tr>
</table></td></tr>
<tr><td>2</td><td>W1 = 2,
Available knapsack capacity = 1
W1 > WA, CASE 1 holds:
V[i, j] = V[i-1, j]
V[1,1] = V[0, 1] = 0</td><td>
<table>
<tr><th>V[i,j]</th><th>j=0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th></tr>
<tr><td>i=0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td></td><td></td><td></td><td></td></tr>
<tr><td>2</td><td>0</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>3</td><td>0</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>4</td><td>0</td><td></td><td></td><td></td><td></td><td></td></tr>
</table></td></tr>
<tr><td>3</td><td>W1 = 2,
Available knapsack capacity = 2
W1 = WA, CASE 2 holds:
V[i, j] = max { V[i-1, j],
vi +V[i-1, j - wi] }
V[1,2] = max { V[0, 2],
3 +V[0, 0] }
= max { 0, 3 + 0 } = 3</td><td>
<table>
<tr><th>V[i,j]</th><th>j=0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th></tr>
<tr><td>i=0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>3</td><td></td><td></td><td></td></tr>
<tr><td>2</td><td>0</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>3</td><td>0</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>4</td><td>0</td><td></td><td></td><td></td><td></td><td></td></tr>
</table></td></tr>
<tr><td>4</td><td>W1 = 2,
Available knapsack capacity =
3,4,5
W1 < WA, CASE 2 holds:
V[i, j] = max { V[i-1, j],
vi +V[i-1, j - wi] }
V[1,3] = max { V[0, 3],
3 +V[0, 1] }
= max { 0, 3 + 0 } = 3</td><td>
<table>
<tr><th>V[i,j]</th><th>j=0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th></tr>
<tr><td>i=0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>3</td><td>3</td><td>3</td><td>3</td></tr>
<tr><td>2</td><td>0</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>3</td><td>0</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>4</td><td>0</td><td></td><td></td><td></td><td></td><td></td></tr>
</table></td></tr>
<tr><td>5</td><td>W2 = 3,
Available knapsack capacity = 1
W2 >WA, CASE 1 holds:
V[i, j] = V[i-1, j]
V[2,1] = V[1, 1] = 0</td><td>
<table>
<tr><th>V[i,j]</th><th>j=0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th></tr>
<tr><td>i=0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>3</td><td>3</td><td>3</td><td>3</td></tr>
<tr><td>2</td><td>0</td><td>0</td><td></td><td></td><td></td><td></td></tr>
<tr><td>3</td><td>0</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>4</td><td>0</td><td></td><td></td><td></td><td></td><td></td></tr>
</table></td></tr>
</table>

6

W2 = 3,
Available knapsack capacity = 2
W2 >WA, CASE 1 holds:
V[i, j] = V[i-1, j]
V[2,2] = V[1, 2] = 3

V[i,j]	j=0	1	2	3	4	5
i=0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3			
3	0					
4	0					

7

W2 = 3,
Available knapsack capacity = 3
W2 = WA, CASE 2 holds:
V[i, j] = max { V[i-1, j],
vi +V[i-1, j - wi] }
V[2,3] = max { V[1, 3],
4 +V[1, 0] }
= max { 3, 4 + 0 } = 4

V[i,j]	j=0	1	2	3	4	5
i=0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4		
3	0					
4	0					

8

W2 = 3,
Available knapsack capacity = 4
W2 < WA, CASE 2 holds:
V[i, j] = max { V[i-1, j],
vi +V[i-1, j - wi] }
V[2,4] = max { V[1, 4],
4 +V[1, 1] }
= max { 3, 4 + 0 } = 4

V[i,j]	j=0	1	2	3	4	5
i=0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	
3	0					
4	0					

9

W2 = 3,
Available knapsack capacity = 5
W2 < WA, CASE 2 holds:
V[i, j] = max { V[i-1, j],
vi +V[i-1, j - wi] }
V[2,5] = max { V[1, 5],
4 +V[1, 2] }
= max { 3, 4 + 3 } = 7

V[i,j]	j=0	1	2	3	4	5
i=0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0					
4	0					

10

W3 = 4,
Available knapsack capacity =
1,2,3
W3 > WA, CASE 1 holds:
V[i, j] = V[i-1, j]

V[i,j]	j=0	1	2	3	4	5
i=0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0	0	3	4		
4	0					

11

W3 = 4,
Available knapsack capacity = 4
W3 = WA, CASE 2 holds:
V[i, j] = max { V[i-1, j],
vi +V[i-1, j - wi] }
V[3,4] = max { V[2, 4],
5 +V[2, 0] }
= max { 4, 5 + 0 } = 5

V[i,j]	j=0	1	2	3	4	5
i=0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0	0	3	4	5	
4	0					

12

W3 = 4,
Available knapsack capacity = 5
W3 < WA, CASE 2 holds:
V[i, j] = max { V[i-1, j],
vi +V[i-1, j - wi] }
V[3,5] = max { V[2, 5],
5 +V[2, 1] }
= max { 7, 5 + 0 } = 7

V[i,j]	j=0	1	2	3	4	5
i=0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0	0	3	4	5	7
4	0					

13

W4 = 5,
Available knapsack capacity = 1,2,3,4
W4 < WA, CASE 1 holds:
V[i, j] = V[i-1, j]

V[i,j]	j=0	1	2	3	4	5
i=0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0	0	3	4	5	7
4	0	0	3	4	5	

14

W4 = 5,
Available knapsack capacity = 5
W4 = WA, CASE 2 holds:
V[i, j] = max { V[i-1, j],
vi +V[i-1, j - wi] }
V[4,5] = max { V[3, 5],
6 +V[3, 0] }
= max { 7, 6 + 0 } = 7

V[i,j]	j=0	1	2	3	4	5
i=0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0	0	3	4	5	7
4	0	0	3	4	5	7

Maximal value is V [4, 5] = 7/-

Step	Table	Remarks

Step 1

V[i,j]	j=0	1	2	3	4	5
i=0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0	0	3	4	5	7
4	0	0	3	4	5	7

Remarks: V[4, 5] = V[3, 5]

→ ITEM 4 NOT included in the subset

Step 2

V[i,j]	j=0	1	2	3	4	5
i=0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0	0	3	4	5	7
4	0	0	3	4	5	7

Remarks: V[3, 5] = V[2, 5]

→ ITEM 3 NOT included in the subset

Step 3

V[i,j]	j=0	1	2	3	4	5
i=0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0	0	3	4	5	7
4	0	0	3	4	5	7

Remarks: V[2, 5] ≠ V[1, 5]

→ ITEM 2 included in the subset

Step 4

Since item 2 is included in the knapsack:
Weight of item 2 is 3kg, therefore, remaining capacity of the knapsack is (5 - 3 =) 2kg

V[i,j]	j=0	1	2	3	4	5
i=0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0	0	3	4	5	7
4	0	0	3	4	5	7

Remarks: V[1, 2] ≠ V[0, 2]

→ ITEM 1 included in the subset

Step 5

Since item 1 is included in the knapsack:
Weight of item 1 is 2kg, therefore, remaining capacity of the knapsack is (2 - 2 =) 0 kg.

Remarks: **Optimal subset: { item 1, item 2 }**

Total weight is: 5kg (2kg + 3kg)
Total profit is: 7/- (3/- + 4/-)

Thank You