0-1 Knapsack

- Given a knapsack with maximum capacity W, and a set S consisting of n items
- Each item i has some weight w_i and benefit value b_i (all w_i and W are integer values)

0-1 Knapsack problem

- Pack the knapsack to achieve maximum total value of packed items

$$\max \sum_{i \in T} b_i \quad \text{subject to} \sum_{i \in T} w_i \leq W$$

- “0-1”: Because items are indivisible; each item must be entirely accepted or rejected.

0-1 Knapsack Example

n = 4 (Number of elements)

W = 5 (Max capacity)

Elements (Weight, Benefit):

(2,3), (3,4), (4,5), (5,6)

Example (2)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1						
2						
3						
4						

for w = 0 to W

$$V[0,w] = 0$$

Example (3)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0					
2	0					
3	0					
4	0					

for i = 1 to n

    V[i,0] = 0

Example (4)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0				
2	0					
3	0					
4	0					

Items:

1: (2,3)

2: (3,4)

3: (4,5)

4: (5,6)

Example (5)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	**3**			
2	0					
3	0					
4	0					

Items:

1: (2,3)

2: (3,4)

3: (4,5)

4: (5,6)

Example (6)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	**3**		
2	0					
3	0					
4	0					

Items:

1: (2,3)

2: (3,4)

3: (4,5)

4: (5,6)

Example (7)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	**3**	
2	0					
3	0					
4	0					

Items:

1: (2,3)

2: (3,4)

3: (4,5)

4: (5,6)

Example (8)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	**3**
2	0					
3	0					
4	0					

Items:

1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

Example (9)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	**0**				
3	0					
4	0					

Items:

1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

Example (10)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	**3**			
3	0					
4	0					

Items:

1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

Example (11)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	**4**		
3	0					
4	0					

Items:

1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

Example (12)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	
3	0					
4	0					

Items:

1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

Example (13)

i\W	0	1	2	3	4	5
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					

Items:

1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

Example (14)

i\W	0	1	2	3	4	5
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					

Items:

1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

Example (15)

i\W	0	1	2	3	4	5
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					

Items:

1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

if w_i <= w // item i can be part of the solution
    if $b_i + V[i-1,w-w_i] > V[i-1,w]$
        $V[i,w] = b_i + V[i-1,w- w_i]$
    else
        $V[i,w] = V[i-1,w]$
else $V[i,w] = V[i-1,w]$ // $w_i > w$

Example (16)

i\W	0	1	2	3	4	5
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					

Items:

1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

if $w_i <= w$ // item i can be part of the solution
    if $b_i + V[i-1,w-w_i] > V[i-1,w]$
        $V[i,w] = b_i + V[i-1,w- w_i]$
    else
        V[i,w] = V[i-1,w]
else $V[i,w] = V[i-1,w]$ // $w_i > w$

Example (17)

i\W	0	1	2	3	4	5
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**	

Items:

1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

if $w_i <= w$ // item i can be part of the solution
    if $b_i + V[i-1,w-w_i] > V[i-1,w]$
        $V[i,w] = b_i + V[i-1,w- w_i]$
    else
        $V[i,w] = V[i-1,w]$
else **V[i,w] = V[i-1,w]** // $w_i > w$

Example (18)

i\W	0	1	2	3	4	5
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

Items:

1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

if $w_i <= w$ // item i can be part of the solution

if $b_i + V[i-1,w-w_i] > V[i-1,w]$

$V[i,w] = b_i + V[i-1,w- w_i]$

else

V[i,w] = V[i-1,w]

else $V[i,w] = V[i-1,w]$ // $w_i > w$

Recursive Formula

$$V[k,w] = \begin{cases} V[k-1,w] & \text{if } w_k > w \\ \max\{V[k-1,w], V[k-1,w-w_k]+b_k\} & \text{else} \end{cases}$$

0-1 Knapsack Algorithm

```
for w = 0 to W
   V[0,w] = 0
for i = 1 to n
   V[i,0] = 0
for i = 1 to n
   for w = 0 to W
         if w_i <= w // item i can be part of the solution
                  if b_i + V[i-1,w-w_i] > V[i-1,w]
                           V[i,w] = b_i + V[i-1,w- w_i]
                  else
                           V[i,w] = V[i-1,w]
         else V[i,w] = V[i-1,w]  // w_i > w
```

Finding the Items

i\W	0	1	2	3	4	5
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

Items:

1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

Finding the Items (2)

i\W	0	1	2	3	4	5
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

Items:

1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

Finding the Items (3)

i\W	0	1	2	3	4	5
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

Items:

1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

Finding the Items (4)

i\W	0	1	2	3	4	5
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

Items:

1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

Finding the Items (5)

i\W	0	1	2	3	4	5
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

Items:

1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

Finding the Items (6)

i\W	0	1	2	3	4	5
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

Items:

1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

The optimal knapsack should contain {1, 2}

Algorithm: Finding the Items

- All of the information we need is in the table.
- $V[n,W]$ is the maximal value of items that can be placed in the Knapsack.

- Let i=n and k=W

  if $V[i,k] \neq V[i-1,k]$ then

    mark the i^{th} item as in the knapsack

    $i = i-1$, $k = k-w_i$

  else

    $i = i-1$

Thank You

Stay Safe