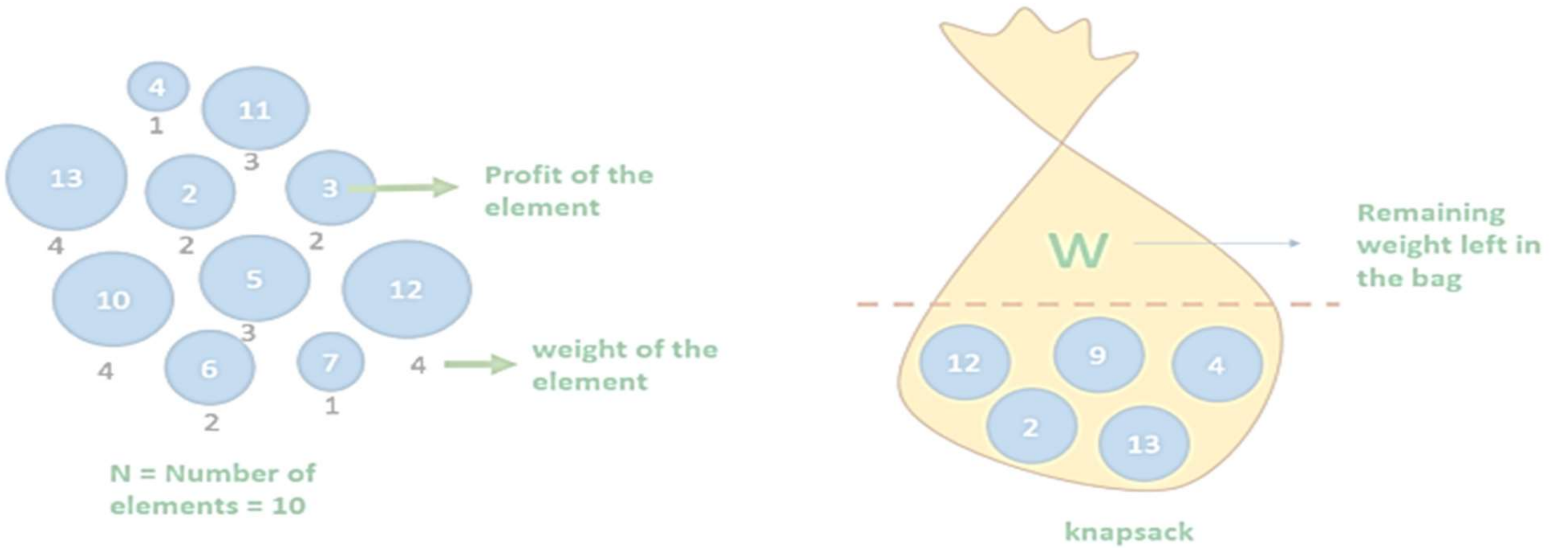

Lecture 22

Dynamic Programming (0/1 Knapsack Problem): Problem Analysis, Notations, Designing DP Algorithm for 0/1 Knapsack Problem & its Time Complexity, and Applications of 0/1 Knapsack Problem

Knapsack problem

Given some items, pack the knapsack to get the maximum total value. Each item has some weight and some value. Total weight that we can carry is no more than some fixed number W. So, we must consider weights of items as well as their values.

Item #	Weight	Value
1	1	8
2	3	6
3	5	5

Knapsack problem

There are two versions of the problem:

1. ***"0-1 knapsack problem"***
   - *Items are indivisible; you either take an item or not. Some special instances can be solved with* ***dynamic programming***

2. ***"Fractional knapsack problem"*** *– Self Learning*
   - *Items are divisible: you can take any fraction of an item*

0-1 Knapsack problem

- 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)
- Problem: How to pack the knapsack to achieve maximum total value of packed items?

Dynamic Programming: 0/1 Knapsack Problem

- Given: A set S of n items, with each item i having
  - w_i – a positive weight
  - b_i – a positive benefit
- Goal: Choose items with maximum total benefit but with weight at most W.
- If fractional amounts are not allowed, then this is the 0/1 knapsack problem
  - In this case, we let T denote the set of items we take

$$\textbf{\textit{Objective}}\text{: maximize } \sum_{i \in T} b_i$$

$$\textit{Constraint}\text{:} \quad \sum_{i \in T} w_i \leq W$$

0-1 Knapsack problem

- *Problem, in other words, is to find*

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

The problem is called a "0-1" problem, because each item must be entirely accepted or rejected.

0-1 Knapsack problem: brute-force approach

Let's first solve this problem with a straightforward algorithm

- *Since there are n items, there are 2^n possible combinations of items.*
- *We go through all combinations and find the one with maximum value and with total weight less or equal to W*
- *Running time will be $O(2^n)$*

0-1 Knapsack problem: Dynamic programming approach

- *We can do better with an algorithm based on dynamic programming*
- *We need to carefully identify the subproblems*

Defining a Subproblem

- *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)*
- *Problem: How to pack the knapsack to achieve maximum total value of packed items?*

Defining a Subproblem

- *We can do better with an algorithm based on dynamic programming*
- *We need to carefully identify the subproblems*

Let's try this:

If items are labeled 1..n, then a subproblem would be to find an optimal solution for S_k = {items labeled 1, 2, .. k}

Defining a Subproblem

If items are labeled $1..n$, then a subproblem would be to find an optimal solution for S_k = {items labeled 1, 2, .. k}

- *This is a reasonable subproblem definition.*
- *The question is: can we describe the final solution (S_n) in terms of subproblems (S_k)?*
- *Unfortunately, we <u>can't</u> do that.*

Defining a Subproblem

$w_1=2$ $b_1=3$	$w_2=4$ $b_2=5$	$w_3=5$ $b_3=8$	$w_4=3$ $b_4=4$	

Max weight: $W = 20$

For S_4:

Total weight: 14

Maximum benefit: 20

$w_1=2$ $b_1=3$	$w_2=4$ $b_2=5$	$w_3=5$ $b_3=8$	$w_5=9$ $b_5=10$	

For S_5:

Total weight: 20

Maximum benefit: 26

?

S_5 S_4

Item #	Weight w_i	Benefit b_i
1	2	3
2	4	5
3	5	8
4	3	4
5	9	10

Solution for S_4 is not part of the solution for S_5!!!

Defining a Subproblem

- *As we have seen, the solution for S_4 is not part of the solution for S_5*
- *So, our definition of a subproblem is flawed and we need another one!*

Defining a Subproblem

- *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)*
- *<u>Problem</u>: How to pack the knapsack to achieve maximum total value of packed items?*

Defining a Subproblem

- *Let's add another parameter: w, which will represent the maximum weight for each subset of items*
- *The subproblem then will be to compute V[k,w], i.e.,* to *find an optimal solution for S_k = {items labeled 1, 2, .. k} in a knapsack of size w*

Recursive Formula for subproblems

- *The subproblem will then be to compute $V[k,w]$, i.e.,* *to find an optimal solution for S_k = {items labeled 1, 2, .. k} in a knapsack of size w*
- *Assuming knowing $V[i, j]$, where i=0,1, 2, … k-1, j=0,1,2, …w, how to derive $V[k,w]$?*

Recursive Formula for subproblems (continued)

Recursive formula for subproblems:

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

It means, that the best subset of S_k that has total weight w is:

1) the best subset of S_{k-1} that has total weight $\leq w$, ***or***

2) the best subset of S_{k-1} that has total weight $\leq w\text{-}w_k$ plus the item k

Recursive Formula

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

- *The best subset of S_k that has the total weight $\leq w$, either contains item k or not.*
- *First case: $w_k > w$. Item k can't be part of the solution, since if it was, the total weight would be $> w$, which is unacceptable.*
- *Second case: $w_k \leq w$. Then the item k can be in the solution, and we choose the case with greater value.*

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
```

Running time

for w = 0 to W

V[0,w] = 0 — *O(W)*

for i = 1 to n

V[i,0] = 0

for i = 1 to n — *Repeat n times*

for w = 0 to W — *O(W)*

< the rest of the code >

What is the running time of this algorithm?

*O(n*W)*

Remember that the brute-force algorithm takes O(2n)

Example

Let's run our algorithm on the following data:

n = 4 (# of elements)

W = 5 (max weight)

Elements (weight, benefit):

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

Example (Cont !!!)

	i\W	0	1	2	3	4	5
wi, bi	0	0	0	0	0	0	0
(2,3),	1						
(3,4),	2						
(4,5),	3						
(5,6)	4						

$$\textit{for } w = 0 \textit{ to } W$$
$$V[0,w] = 0$$

n = 4 (# of elements)

W = 5 (max weight)

Elements (weight, benefit):

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

Example (Cont !!!)

	i\W	0	1	2	3	4	5
wi, bi	0	0	0	0	0	0	0
(2,3),	1	0					
(3,4),	2	0					
(4,5),	3	0					
(5,6)	4	0					

for i = 1 to n

$$V[i,0] = 0$$

- n = 4 (# of elements)
- W = 5 (max weight)
- Elements (weight, benefit):
- (2,3), (3,4), (4,5), (5,6)

Example (Cont !!!)

Items: (w_i,b_i)

1: (2,3)

2: (3,4)

3: (4,5)

4: (5,6)

	i\W	0	1	2	3	4	5
wi, bi	0	0	0	0	0	0	0
(2,3),	1	0	0				
(3,4),	2	0					
(4,5),	3	0					
(5,6)	4	0					

$i=1$

$b_i=3$

$w_i=2$

$w=1$

$w-w_i=-1$

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

$n = 4$ *(# of elements)*

$W = 5$ *(max weight)*

Elements (weight, benefit):

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

Example (Cont !!!)

Items:

1: (2,3)

2: (3,4)

3: (4,5)

4: (5,6)

	i\W	0	1	2	3	4	5
wi, bi	0	0	0	0	0	0	0
(2,3),	1	0	0	3			
(3,4),	2	0					
(4,5),	3	0					
(5,6)	4	0					

$i=1$

$b_i=3$

$w_i=2$

$w=2$

$w-w_i=0$

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

n = 4 (# of elements)

W = 5 (max weight)

Elements (weight, benefit):

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

Example (Cont !!!)

Items:

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

	i\W	0	1	2	3	4	5
wi, bi	0	0	0	0	0	0	0
(2,3),	1	0	0	3	3		
(3,4),	2	0					
(4,5),	3	0					
(5,6)	4	0					

$i=1$

$b_i=3$

$w_i=2$

$w=3$

$w-w_i=1$

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

n = 4 (# of elements)

W = 5 (max weight)

Elements (weight, benefit):

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

Example (Cont !!!)

	i\W	0	1	2	3	4	5
wi, bi	0	0	0	0	0	0	0
(2,3),	1	0	0	3	3	3	
(3,4),	2	0					
(4,5),	3	0					
(5,6)	4	0					

Items:

1: (2,3)

2: (3,4)

3: (4,5)

4: (5,6)

$i=1$

$b_i=3$

$w_i=2$

$w=4$

$w-w_i=2$

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

n = 4 (# of elements)

W = 5 (max weight)

Elements (weight, benefit):

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

Example (8)

Items:

1: (2,3)

2: (3,4)

3: (4,5)

4: (5,6)

	i\W	0	1	2	3	4	5
wi, bi	0	0	0	0	0	0	0
(2,3),	1	0	0	3	3	3	3
(3,4),	2	0					
(4,5),	3	0					
(5,6)	4	0					

$i=1$

$b_i=3$

$w_i=2$

$w=5$

$w-w_i=3$

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

n = 4 (# of elements)

W = 5 (max weight)

Elements (weight, benefit):

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

Example (9)

Items:

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

	i\W	0	1	2	3	4	5
wi, bi	0	0	0	0	0	0	0
(2,3),	1	0	0	3	3	3	3
(3,4),	2	0	0				
(4,5),	3	0					
(5,6)	4	0					

$i=2$

$b_i=4$

$w_i=3$

$w=1$

$w-w_i=-2$

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

- n = 4 (# of elements)
- W = 5 (max weight)
- Elements (weight, benefit):
- (2,3), (3,4), (4,5), (5,6)

Example (Cont !!!)

Items:

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

	i\W	0	1	2	3	4	5
wi, bi	0	0	0	0	0	0	0
(2,3),	1	0	0	3	3	3	3
(3,4),	2	0	0	3			
(4,5),	3	0					
(5,6)	4	0					

$i=2$

$b_i=4$

$w_i=3$

$w=2$

$w-w_i=-1$

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

n = 4 (# of elements)

W = 5 (max weight)

Elements (weight, benefit):

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

Example (Cont !!!)

Items:

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

	i\W	0	1	2	3	4	5
wi, bi	0	0	0	0	0	0	0
(2,3),	1	0	0	3	3	3	3
(3,4),	2	0	0	3	4		
(4,5),	3	0					
(5,6)	4	0					

$i=2$

$b_i=4$

$w_i=3$

$w=3$

$w-w_i=0$

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 (Cont !!!)

Items:

1: (2,3)

2: (3,4)

3: (4,5)

4: (5,6)

	i\W	0	1	2	3	4	5
wi, bi	0	0	0	0	0	0	0
(2,3),	1	0	0	3	3	3	3
(3,4),	2	0	0	3	4	4	
(4,5),	3	0					
(5,6)	4	0					

$i=2$

$b_i=4$

$w_i=3$

$w=4$

$w-w_i=1$

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

n = 4 (# of elements)

W = 5 (max weight)

Elements (weight, benefit):

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

Example (Cont !!!)

Items:

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

	i\W	0	1	2	3	4	5
wi, bi	0	0	0	0	0	0	0
(2,3),	1	0	0	3	3	3	3
(3,4),	2	0	0	3	4	4	7
(4,5),	3	0					
(5,6)	4	0					

$i=2$

$b_i=4$

$w_i=3$

$w=5$

$w-w_i=2$

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

- n = 4 (# of elements)
- W = 5 (max weight)
- Elements (weight, benefit):
- (2,3), (3,4), (4,5), (5,6)

Example (Cont !!!)

	i\W	0	1	2	3	4	5
wi, bi	0	0	0	0	0	0	0
(2,3),	1	0	0	3	3	3	3
(3,4),	2	0	0	3	4	4	7
(4,5),	3	0	0	3	4		
(5,6)	4	0					

Items:

1: (2,3)

2: (3,4)

3: (4,5)

4: (5,6)

$i=3$

$b_i=5$

$w_i=4$

$w= 1..3$

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

- n = 4 (# of elements)
- W = 5 (max weight)
- Elements (weight, benefit):
- (2,3), (3,4), (4,5), (5,6)

Example (Cont !!!)

Items:

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

4: (5,6)

	i\W	0	1	2	3	4	5
wi, bi	0	0	0	0	0	0	0
(2,3),	1	0	0	3	3	3	3
(3,4),	2	0	0	3	4	4	7
(4,5),	3	0	0	3	4	5	
(5,6)	4	0					

$i=3$

$b_i=5$

$w_i=4$

$w= 4$

$w- w_i=0$

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

- n = 4 (# of elements)
- W = 5 (max weight)
- Elements (weight, benefit):
- (2,3), (3,4), (4,5), (5,6)

Example (Cont !!!)

Items:

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

	i\W	0	1	2	3	4	5
wi, bi	0	0	0	0	0	0	0
(2,3),	1	0	0	3	3	3	3
(3,4),	2	0	0	3	4	4	7
(4,5),	3	0	0	3	4	5	7
(5,6)	4	0					

$i=3$

$b_i=5$

$w_i=4$

$w= 5$

$w- w_i=1$

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

n = 4 (# of elements)

W = 5 (max weight)

Elements (weight, benefit):

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

Example (Cont !!!)

Items:

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

	i\W	0	1	2	3	4	5
wi, bi	0	0	0	0	0	0	0
(2,3),	1	0	0	3	3	3	3
(3,4),	2	0	0	3	4	4	7
(4,5),	3	0	0	3	4	5	7
(5,6)	4	0	0	3	4	5	

i=4

$b_i=6$

$w_i=5$

$w= 1..4$

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

- n = 4 (# of elements)
- W = 5 (max weight)
- Elements (weight, benefit):
- (2,3), (3,4), (4,5), (5,6)

Example (Cont !!!)

	i\W	0	1	2	3	4	5
wi, bi	0	0	0	0	0	0	0
(2,3),	1	0	0	3	3	3	3
(3,4),	2	0	0	3	4	4	7
(4,5),	3	0	0	3	4	5	7
(5,6)	4	0	0	3	4	5	7

Items:

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

$i=4$
$b_i=6$
$w_i=5$
$w= 5$
$w- w_i=0$

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

n = 4 (# of elements)

W = 5 (max weight)

Elements (weight, benefit):

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

Comments

- *This algorithm only finds the max possible value that can be carried in the knapsack*
  - *i.e., the value in V[n,W]*
- *To know the items that make this maximum value, an addition to this algorithm is necessary*

How to find actual Knapsack 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

  $i = i-1$ *// Assume the* i^{th} *item is* *not* *in the knapsack*

  // Could it be in the optimally packed knapsack?

Finding the Items

	i\W	0	1	2	3	4	5
wi, bi	0	0	0	0	0	0	0
(2,3),	1	0	0	3	3	3	3
(3,4),	2	0	0	3	4	4	7
(4,5),	3	0	0	3	4	5	7
(5,6)	4	0	0	3	4	5	7

Items:

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

$i=4$
$k=5$
$b_i=6$
$w_i=5$
$V[i,k] = 7$
$V[i-1,k] = 7$

$i=n,\ k=W$

while i,k > 0

    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

        $i = i-1$

- n = 4 (# of elements)
- W = 5 (max weight)
- Elements (weight, benefit):
- (2,3), (3,4), (4,5), (5,6)

Finding the Items (Cont !!!)

Items:

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

	i\W	0	1	2	3	4	5
wi, bi	0	0	0	0	0	0	0
(2,3),	1	0	0	3	3	3	3
(3,4),	2	0	0	3	4	4	7
(4,5),	3	0	0	3	4	5	7
(5,6)	4	0	0	3	4	5	7

i=4
k= 5
b_i=6
w_i=5
$V[i,k] = 7$
$V[i-1,k] = 7$

i=n, k=W
while i,k > 0
    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
        $i = i-1$

n = 4 (# of elements)

W = 5 (max weight)

Elements (weight, benefit):

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

Finding the Items (Cont !!!)

Items:

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

	i\W	0	1	2	3	4	5
wi, bi	0	0	0	0	0	0	0
(2,3),	1	0	0	3	3	3	3
(3,4),	2	0	0	3	4	4	7
(4,5),	3	0	0	3	4	5	7
(5,6)	4	0	0	3	4	5	7

$i=3$

$k=5$

$b_i=5$

$w_i=4$

$V[i,k] = 7$

$V[i-1,k] = 7$

i=n, k=W

while i,k > 0

  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

    $i = i-1$

- n = 4 (# of elements)
- W = 5 (max weight)
- Elements (weight, benefit):
- (2,3), (3,4), (4,5), (5,6)

Finding the Items (Cont !!!)

Items:

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

	i\W	0	1	2	3	4	5
wi, bi	0	0	0	0	0	0	0
(2,3),	1	0	0	3	3	3	3
(3,4),	2	0	0	3	4	4	7
(4,5),	3	0	0	3	4	5	7
(5,6)	4	0	0	3	4	5	7

$i=2$

$k=5$

$b_i=4$

$w_i=3$

$V[i,k] = 7$

$V[i-1,k] = 3$

$k - w_i = 2$

i=n, k=W

while i,k > 0

    if V[i,k] ≠ V[i−1,k] then

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

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

    else $i = i-1$

- n = 4 (# of elements)
- W = 5 (max weight)
- Elements (weight, benefit):
- (2,3), (3,4), (4,5), (5,6)

Finding the Items (Cont !!!)

Items:

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

	i\W	0	1	2	3	4	5
wi, bi	0	0	0	0	0	0	0
(2,3),	1	0	0	3	3	3	3
(3,4),	2	0	0	3	4	4	7
(4,5),	3	0	0	3	4	5	7
(5,6)	4	0	0	3	4	5	7

$i=1$

$k=2$

$b_i=3$

$w_i=2$

$V[i,k]=3$

$V[i-1,k]=0$

$k-w_i=0$

i=n, k=W

while i,k > 0

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

$i = i-1$

n = 4 (# of elements)

W = 5 (max weight)

Elements (weight, benefit):

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

Finding the Items (Cont !!!)

Items:

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

	i\W	0	1	2	3	4	5
wi, bi	0	0	0	0	0	0	0
(2,3),	1	0	0	3	3	3	3
(3,4),	2	0	0	3	4	4	7
(4,5),	3	0	0	3	4	5	7
(5,6)	4	0	0	3	4	5	7

i=0

k= 0

The optimal knapsack should contain {1, 2}

i=n, k=W

while i,k > 0

if V[i,k] ≠ V[i−1,k] then

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

$i = i−1$, $k = k-w_i$

else

$i = i−1$

n = 4 (# of elements)

W = 5 (max weight)

Elements (weight, benefit):

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

Finding the Items (Cont !!!)

	i\W	0	1	2	3	4	5
wi, bi	0	0	0	0	0	0	0
(2,3),	1	0	0	3	3	3	3
(3,4),	2	0	0	3	4	4	7
(4,5),	3	0	0	3	4	5	7
(5,6)	4	0	0	3	4	5	7

i=n, k=W

while i,k > 0

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

$i = i-1$

Items:

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

The optimal knapsack should contain {1, 2}

n = 4 (# of elements)

W = 5 (max weight)

Elements (weight, benefit):

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

Homework

1. a. Apply the bottom-up dynamic programming algorithm to the following instance of the knapsack problem:

item	weight	value
1	3	\$25
2	2	\$20
3	1	\$15
4	4	\$40
5	5	\$50

, capacity $W = 6$.

› *How to find out which items are in the optimal subset?*

Summary

- *Dynamic programming is a useful technique of solving certain kind of problems*
- *When the solution can be recursively described in terms of partial solutions, we can store these partial solutions and re-use them as necessary (memorization)*
- *Knapsack problem can be solve using dynamic programming approach where items are selected in such a optimal way which help to maximize the values*
- *Running time of dynamic programming algorithm vs. naïve algorithm:*
  - *0-1 Knapsack problem:* $\boldsymbol{O(W*n)}$ *vs.* $\boldsymbol{O(2^n)}$

Thank You!!!

Have a good day