

## Design & Analysis of Algorithms (Dynamic Programming)

### Binary (0/1)/Integer Knapsack Problem

Dr. Soharab Hossain Shaikh  
BML Munjal University

1

## 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 value.

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

2

## Variations of Knapsack Problems

There are two versions of the problem:

1. "0-1 knapsack problem" and
  2. "Fractional knapsack problem"
1. Items are indivisible; you either take an item or not. Solved with *dynamic programming*
  2. Items are divisible; you can take any fraction of an item. Solved with a *greedy algorithm*.
    - ❖ We have already seen this version

3

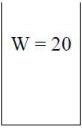
## 0/1 Knapsack Problem






- ◆ 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  $w_i$ ,  $b_i$  and  $W$  are integer values)
- ◆ Problem: How to pack the knapsack to achieve maximum total value of packed items?

4

## 0/1 Knapsack Problem

This is a knapsack  
Max weight:  $W = 20$



Items	Weight $w_i$	Benefit value $b_i$
	2	3
	3	4
	4	5
	5	8
	9	10

5

## 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.
- ◆ In the “*Fractional Knapsack Problem*,” we can take fractions of items.

6

## Brute Force / Naïve Solution

- 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  $O(2^n)$

7

## Better Solution?

- ◆ Can we do better?
- ◆ Yes, 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  $S_k = \{\text{items labeled } 1, 2, \dots, k\}$

8

## Defining a Subproblem

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

- ◆ 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 can't do that.

9

## Defining a Subproblem

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

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

Max weight:  $W = 20$   
**For  $S_4$ :**  
 Total weight: 14;  
 Maximum benefit: 20

**For  $S_5$ :**  
 Total weight: 20  
 Maximum benefit: 26

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

10

## 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!
- ◆ Let's add another parameter:  $w$ , which will represent the exact weight for each subset of items
- ◆ The subproblem then will be to compute  $B[k, w]$

11

## Recursive Formula for Subproblems

Recursive formula for subproblems:

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

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  $w$ . **or**
- 2) the best subset of  $S_{k-1}$  that has total weight  $w-w_k$  plus the item  $k$

12

## Recursive Formula for Subproblems

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

- ◆ The best subset of  $S_k$  that has the total weight  $w$ , 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.

13

## 0/1 Knapsack Algorithm

```

for w = 0 to W
  B[0,w] = 0
for i = 1 to n
  B[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 + B[i-1,w-w_i] > B[i-1,w]
        B[i,w] = b_i + B[i-1,w-w_i]
      else
        B[i,w] = B[i-1,w]
    else B[i,w] = B[i-1,w] // w_i > w

```

14

## Running Time

```

for w = 0 to W      O(W)
  B[0,w] = 0
for i = 1 to n
  B[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  $O(2^n)$

15

## 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)

16

## Example

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

for  $w = 0$  to  $W$   
 $B[0, w] = 0$

17

## Example

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  $n$   
 $B[i, 0] = 0$

18

## Example

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)

$i=1$   
 $b_i=3$   
 $w_i=2$   
 $w=1$   
 $w-w_i=-1$

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

19

## Example

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)

$i=1$   
 $b_i=3$   
 $w_i=2$   
 $w=2$   
 $w-w_i=0$

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

20

## Example

Items:

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

$i=1$   
 $b_i=3$   
 $w_i=2$   
 $w=3$   
 $w-w_i=1$

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					

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

21

## Example

Items:

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

$i=1$   
 $b_i=3$   
 $w_i=2$   
 $w=4$   
 $w-w_i=2$

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					

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

22

## Example

Items:

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

$i=1$   
 $b_i=3$   
 $w_i=2$   
 $w=5$   
 $w-w_i=3$

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

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

23

## Example

Items:

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

$i=2$   
 $b_i=4$   
 $w_i=3$   
 $w=1$   
 $w-w_i=-2$

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	0					
4	0					

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

24

## Example

Items:

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

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			
3	0					
4	0					

$i=2$   
 $b_i=4$   
 $w_i=3$   
 $w=2$   
 $w-w_i=-1$

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

25

## Example

Items:

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

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		
3	0					
4	0					

$i=2$   
 $b_i=4$   
 $w_i=3$   
 $w=3$   
 $w-w_i=0$

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

26

## Example

Items:

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

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		
3	0					
4	0					

$i=2$   
 $b_i=4$   
 $w_i=3$   
 $w=4$   
 $w-w_i=1$

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

27

## Example

Items:

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

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					
4	0					

$i=2$   
 $b_i=4$   
 $w_i=3$   
 $w=5$   
 $w-w_i=2$

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

28

## Example

Items:

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

$i=3$   
 $b_i=5$   
 $w_i=4$   
 $w=1..3$

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	4	7
4	0	0	3	4	4	7

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

29

## Example

Items:

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

$i=3$   
 $b_i=5$   
 $w_i=4$   
 $w=4$   
 $w-w_i=0$

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

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

30

## Example

Items:

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

$i=3$   
 $b_i=5$   
 $w_i=4$   
 $w=5$   
 $w-w_i=1$

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

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

31

## Example

Items:

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

$i=4$   
 $b_i=6$   
 $w_i=5$   
 $w=1..4$

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

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

32



## Example

Items:

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

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

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

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

33

## Example

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

34

## Finding the Knapsack Items

- ◆ All of the information we need is in the table.
- ◆  $B[n, W]$  is the maximal value of items that can be placed in the Knapsack.
- ◆ Let  $i=n$  and  $k=W$   
 if  $B[i, k] \neq B[i-1, k]$  then  
   mark the  $i^{\text{th}}$  item as in the knapsack  
    $i = i-1$ ,  $k = k-w_i$   
 else  
    $i = i-1$  // Assume the  $i^{\text{th}}$  item is not in the knapsack  
   // Could it be in the optimally packed knapsack?

35

## Finding the Knapsack Items

Items:

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

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

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

$i=n$ ,  $k=W$   
 while  $i, k > 0$   
   if  $B[i, k] \neq B[i-1, k]$  then  
     mark the  $i^{\text{th}}$  item as in the knapsack  
      $i = i-1$ ,  $k = k-w_i$   
   else  
      $i = i-1$

36

## Finding the Knapsack Items

Items:

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

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

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

```

i=n, k=W
while i,k > 0
  if B[i,k] ≠ B[i-1,k] then
    mark the ith item as in the knapsack
    i = i-1, k = k-wi
  else
    i = i-1
  
```

37

## Finding the Knapsack Items

Items:

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

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

$i=3$   
 $k=5$   
 $b_i=6$   
 $w_i=4$   
 $B[i,k] = 7$   
 $B[i-1,k] = 7$

```

i=n, k=W
while i,k > 0
  if B[i,k] ≠ B[i-1,k] then
    mark the ith item as in the knapsack
    i = i-1, k = k-wi
  else
    i = i-1
  
```

38

## Finding the Knapsack Items

Items:

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

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

$i=2$   
 $k=5$   
 $b_i=4$   
 $w_i=3$   
 $B[i,k] = 7$   
 $B[i-1,k] = 3$   
 $k - w_i = 2$

```

i=n, k=W
while i,k > 0
  if B[i,k] ≠ B[i-1,k] then
    mark the ith item as in the knapsack
    i = i-1, k = k-wi
  else
    i = i-1
  
```

39

## Finding the Knapsack Items

Items:

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

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

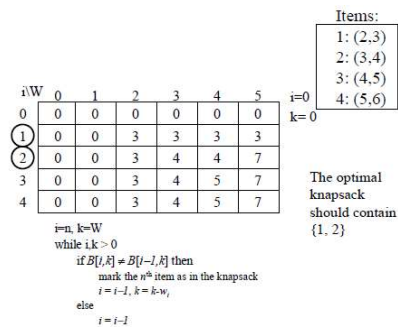
$i=1$   
 $k=2$   
 $b_i=3$   
 $w_i=2$   
 $B[i,k] = 3$   
 $B[i-1,k] = 0$   
 $k - w_i = 0$

```

i=n, k=W
while i,k > 0
  if B[i,k] ≠ B[i-1,k] then
    mark the ith item as in the knapsack
    i = i-1, k = k-wi
  else
    i = i-1
  
```

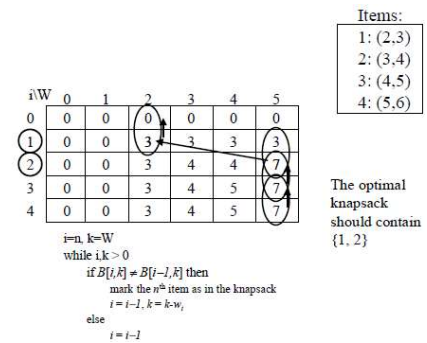
40

## Finding the Knapsack Items



41

## Finding the Knapsack Items



42

## Conclusion

- ◆ 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 (memoization)
- ◆ Running time of dynamic programming algorithm vs. naïve algorithm:
  - » 0-1 Knapsack problem:  $O(W \cdot n)$  vs.  $O(2^n)$

43

## End of Lecture

44