0-1 Knapsack problem

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?

0-1 Knapsack problem

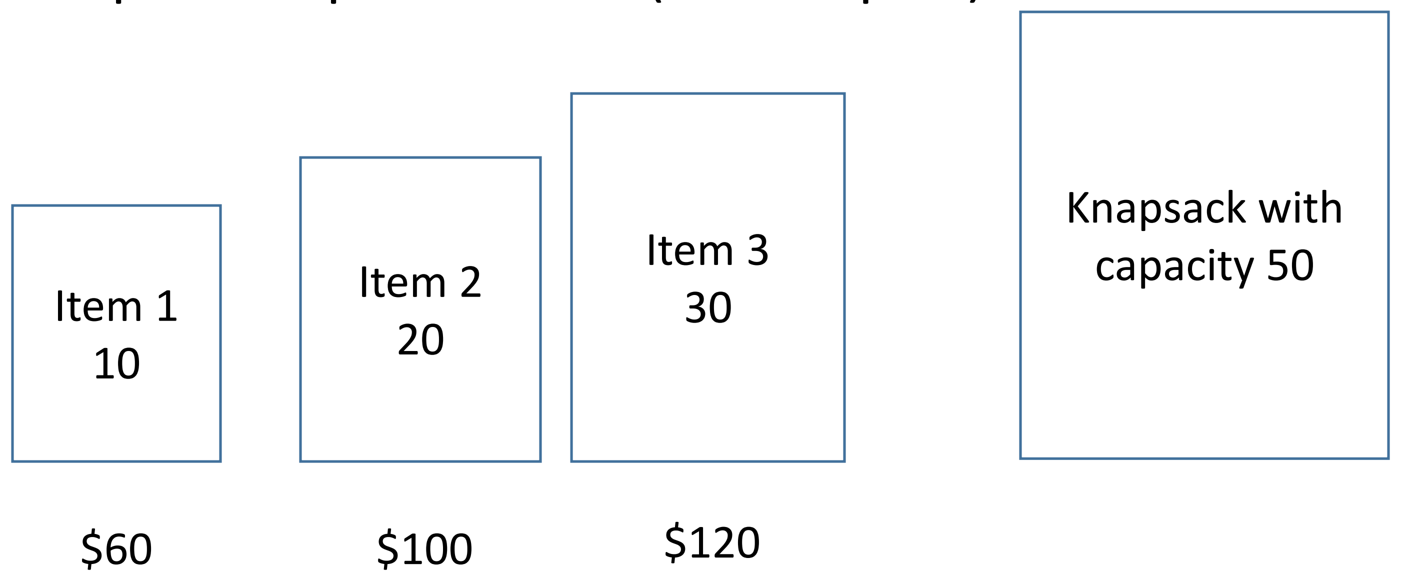
Problem, in other words, is to find

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

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

Just another version of this problem is the “*Fractional Knapsack Problem*”, where we can take fractions of items.

0-1 Knapsack problem (Example)



0-1 Knapsack problem: brute- force approach

Let’s first solve this problem with a straightforward algorithm

Since there are *n* items, how many possible combination of items are possible?

It is the same problem as determining all possible subsets of a set.

0-1 Knapsack problem: brute- force approach

Since there are n items, there are 2^n possible combinations of items.

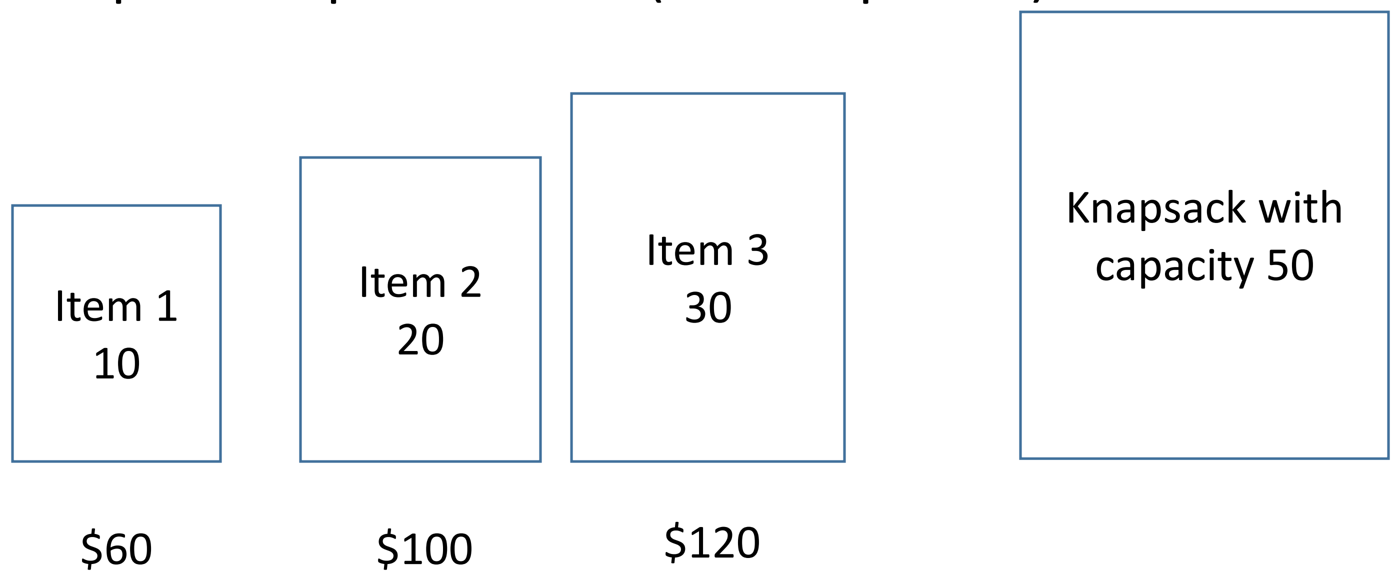
We go through all combinations and find the one with the most total value and with total weight less or equal to W

Running time will be $O(2^n)$

Does any greedy strategy work for 0-1 knapsack problem?

Lets find out!

0-1 Knapsack problem (Example 1)



Value per weight: Item 1 (6) Item 2 (5) Item 3 (4)

Lets try value per weight greedy strategy that we used for fractional knapsack

0-1 Knapsack (Counter Example for Greedy Algorithm)

Item 1
10

Item 2
20

Item 3
30

Knapsack with capacity 50

$60 $100 $120

Value per weight: Item 1 (6) Item 2 (5) Item 3 (4)

Value per weight Greedy Solution

Item 1 (10)

Item 2 (20)
Total weight =30

Total Value = $160

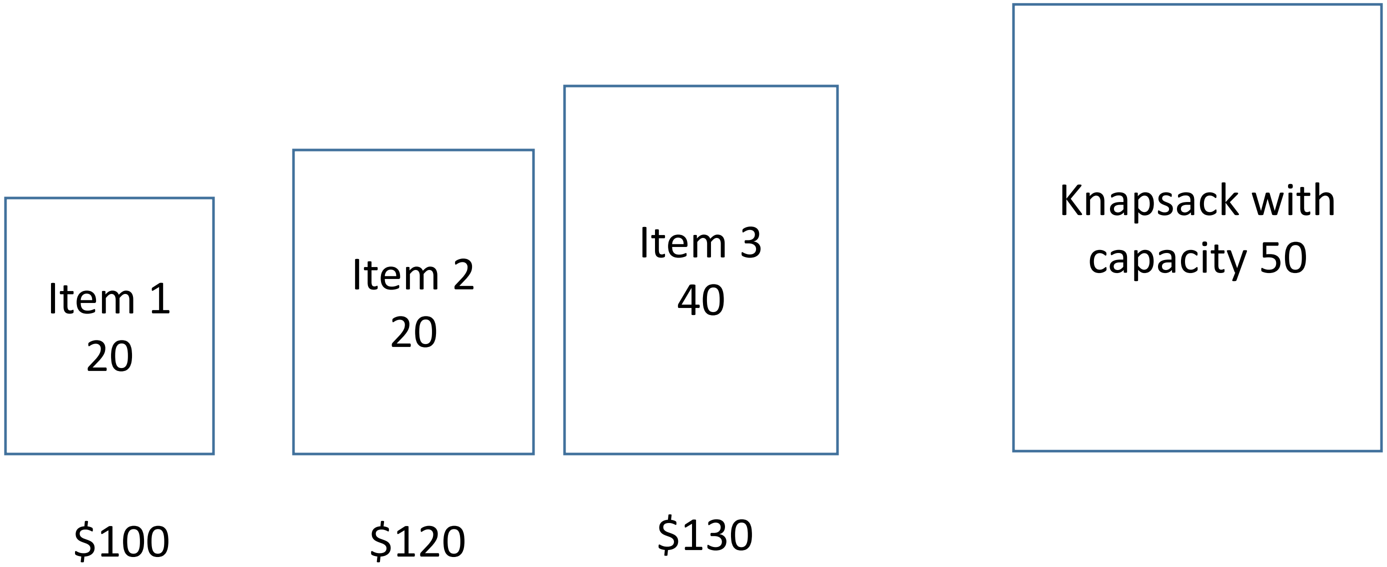
Optimal Solution

Item 2 (20)

Item 3 (30)
Total weight =50

Total Value = $220

0-1 Knapsack problem (Example 2)



Lets try greedy algorithm: Select item with maximum value first.

0-1 Knapsack problem (Example 2)

Item 1
20

$100

Item 2
20

$120

Item 3
40

$130

Knapsack with
capacity 50

Maximum value first
Greedy Solution

Item 3 (40)

Total Value = $130

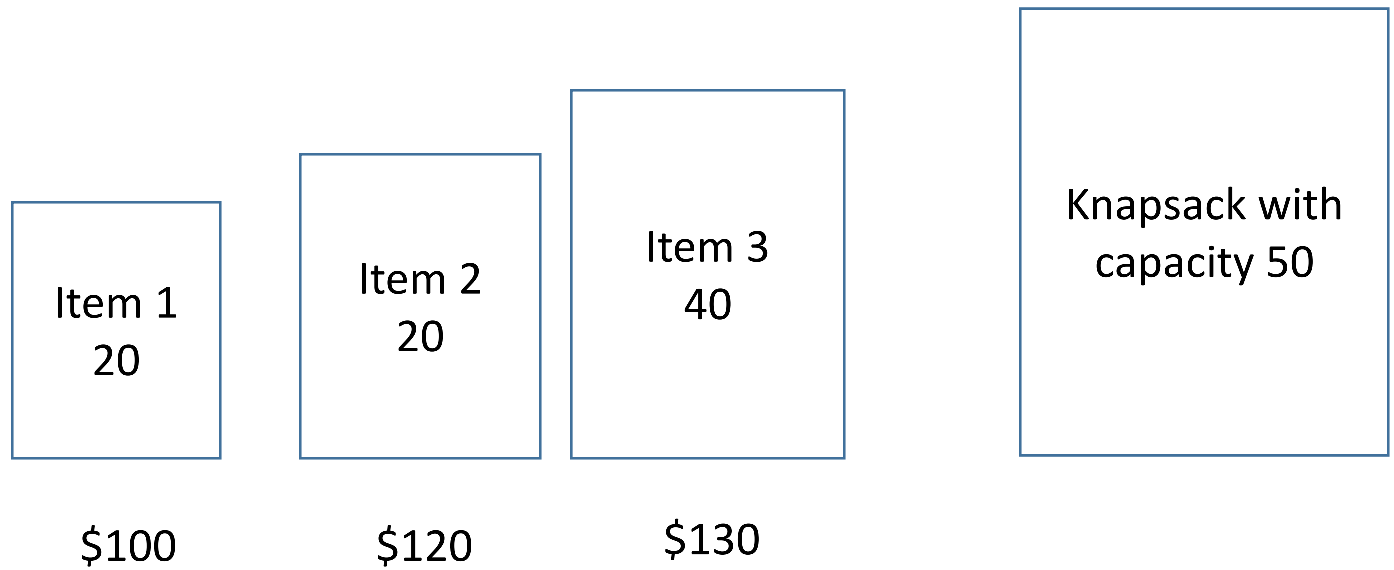
Optimal
Solution

Item 1 (20)

Item 2 (20)
Total weight =40

Total Value = $220

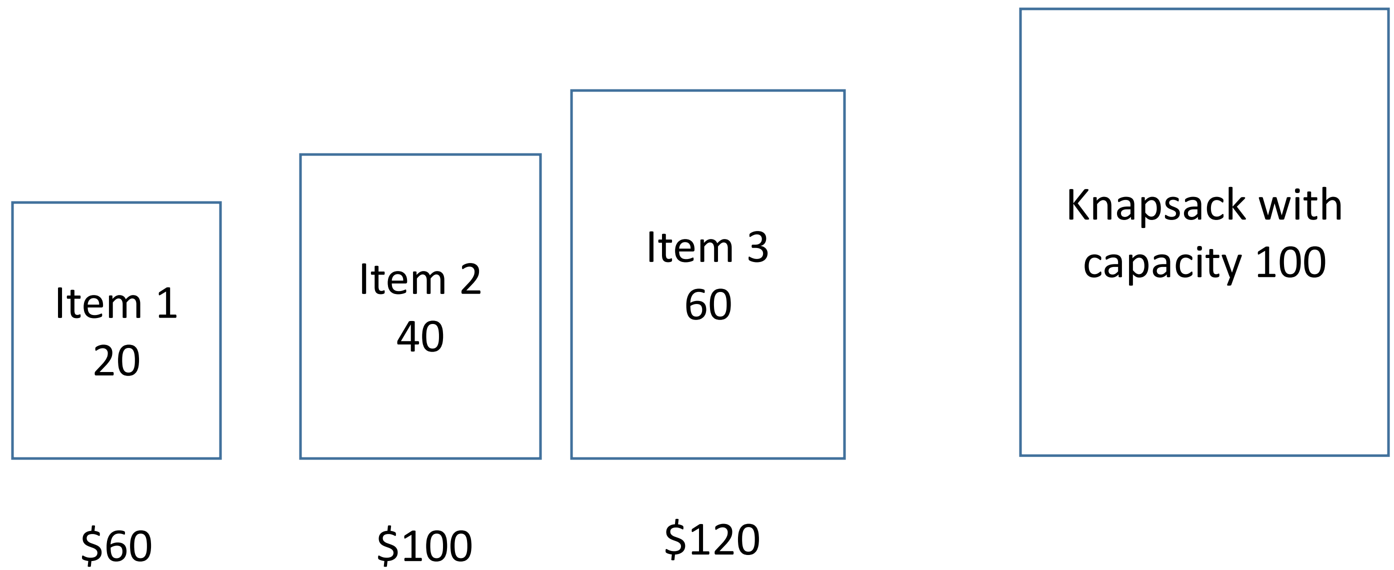
0-1 Knapsack problem (Example 2)



Lets try another greedy algorithm: Select item with minimum weight first.

It will give optimal solution in this example with value $220

0-1 Knapsack problem (Example 3)



Lets try greedy algorithm: Select item with minimum weight first.

0-1 Knapsack problem (Example 3)

Item 1 20	Item 2 40	Item 3 60	Knapsack with capacity 100
$60	$100	$120	

Minimum weight first
Greedy Solution

Item 1 (20)

Item 2 (40)

Total weight = 60

Total Value = $160

Optimal Solution

Item 2 (40)

Item 3 (60)
Total weight =100

Total Value = $220

No greedy strategy guarantees optimal solution for 0-1 Knapsack problem

- We have seen counter examples for three greedy algorithms in previous slides

0-1 Knapsack problem

Can we do better than exponential time brute force?

Yes, with an algorithm based on dynamic programming

We need to carefully identify the sub problems

Let's try this:

If items are labeled *1..n*, then a sub problem 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 valid 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.

Explanation follows….

Defining a Subproblem

Max weight: W = 20

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

S_4

S_5

Suppose original problem has 5 items S_5 and we define a sub problem with first 4 items S_4

Defining a Subproblem

Max weight: W = 20

Optimal solution For S_5 :

$w_1=2$ $b_1=3$	$w_3=4$ $b_3=5$	$w_4=5$ $b_4=8$	$w_5=9$ $b_5=10$

Total weight: 20
total benefit: 26

S_5 S_4

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

Does the optimal solution for S_5 contain optimal solution for sub problem S_4?

Defining a Subproblem

Max weight: W = 20

Optimal solution For S_5 :

$w_1=2$ $b_1=3$	$w_3=4$ $b_3=5$	$w_4=5$ $b_4=8$	$w_5=9$ $b_5=10$

Total weight: 20
total benefit: 26

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

S_4

S_5

Optimal solution For S_4 :

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

Total weight: 14;
total benefit: 20

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

Defining a Subproblem (continued)

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 <u>exact</u> weight for each subset of items

Defining a Subproblem

Max weight: W = 20

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

S_4 S_5

Suppose original problem has 5 items and total weight 20, $S_{5,20}$ and we define a sub problem with first 4 items and how much weight?

If 5th item is part of optimal solution then the total weight for subproblem should be $20 - w_5 = 20 - 9 = 11$ subproblem = $S_{4,11}$,
if 5th item is not part of optimal solution then total capacity stays same so Subproblem = $S_{4,20}$

Recurrence for DP solution ?

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 one of the two:

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

Recursive Formula

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

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: $w_k <= w$. Then the item k can be in the solution, and we choose the case with great value

0-1 Knapsack Algorithm

for $w = 0$ to W

$B[0,w] = 0$

for $i = 0$ to n

$B[i,0] = 0$

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

Running time

for w = 0 to W

    B[0,w] = 0

O(W)

for i = 0 to n

    B[i,0] = 0

    for w = 0 to W

        < the rest of the code >

Repeat *n* times

O(W)

What is the running time of this algorithm O(Wn)

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 (2)

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

for w = 0 to W

$$B[0,w] = 0$$

Example (3)

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

for i = 0 to n

$B[i,0] = 0$

Example (4)

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

$i=1$

$b_i=3$

$w_i=2$

$w=1$

$w-w_i=-1$

Items:

(Weight, benefit)

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

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

  else

    $B[i,w] = B[i-1,w]$

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

Example (5)

Items:

1: (2,3)

2: (3,4)

3: (4,5)

4: (5,6)

W \ i	0	1	2	3	4
0	0	0	0	0	0
1	0	0			
2	0	**3**			
3	0				
4	0				
5	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 $b_i + B[i-1,w-w_i] > B[i-1,w]$

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

    else

        $B[i,w] = B[i-1,w]$

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

Example (6)

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

Items:

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

$i=1$

$b_i=3$

$w_i=2$

$w=3$

$w-w_i=1$

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

Example (7)

Items:

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

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

$i=1$

$b_i=3$

$w_i=2$

$w=4$

$w-w_i=2$

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

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

    $\mathbf{B[i,w] = b_i + B[i-1,w- w_i]}$

  else

    $B[i,w] = B[i-1,w]$

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

Example (8)

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

Items:

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

$i=1$

$b_i=3$

$w_i=2$

$w=5$

$w-w_i=2$

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

Example (9)

Items:

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

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

$i=2$

$b_i=4$

$w_i=3$

$w=1$

$w-w_i=-2$

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

  if b_i + 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

Example (10)

Items:

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

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

$i=2$

$b_i=4$

$w_i=3$

$w=2$

$w-w_i=-1$

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

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

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

    else

        $B[i,w] = B[i-1,w]$

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

Example (11)

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

Items:

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

$i=2$

$b_i=4$

$w_i=3$

$w=3$

$w-w_i=0$

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

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

    $\mathbf{B[i,w] = b_i + B[i-1,w- w_i]}$

  else

    $B[i,w] = B[i-1,w]$

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

Example (12)

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

Items:

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

$i=2$

$b_i=4$

$w_i=3$

$w=4$

$w-w_i=1$

```
if wi <= w // item i can be part of the solution
    if bi + B[i-1,w-wi] > B[i-1,w]
        B[i,w] = bi + B[i-1,w- wi]
    else
        B[i,w] = B[i-1,w]
else B[i,w] = B[i-1,w]  // wi > w
```

Example (13)

Items:

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

W \ i	0	1	2	3	4
0	0	0	0	0	0
1	0	0	0		
2	0	3	3		
3	0	3	4		
4	0	3	4		
5	0	3	7		

$i=2$

$b_i=4$

$w_i=3$

$w=5$

$w-w_i=2$

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

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

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

  else

    $B[i,w] = B[i-1,w]$

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

Example (14)

W \ i	0	1	2	3	4
0	0	0	0	0	0
1	0	0	0	**0**	
2	0	3	3	**3**	
3	0	3	4	**4**	
4	0	3	4		
5	0	3	7		

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

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

  else

    $B[i,w] = B[i-1,w]$

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

Example (15)

Items:

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

W \ i	0	1	2	3	4
0	0	0	0	0	0
1	0	0	0	0	
2	0	3	3	3	
3	0	3	4	4	
4	0	3	4	**5**	
5	0	3	7		

$i=3$

$b_i=5$

$w_i=4$

$w=4$

$w- w_i=0$

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

Example (15)

Items:

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

W \ i	0	1	2	3	4
0	0	0	0	0	0
1	0	0	0	0	
2	0	3	3	3	
3	0	3	4	4	
4	0	3	4	5	
5	0	3	7 →	7	

$i=3$

$b_i=5$

$w_i=4$

$w=5$

$w- w_i=1$

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

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

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

    else

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

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

Example (16)

Items:

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

W \ i	0	1	2	3	4
0	0	0	0	0	0
1	0	0	0	0 →	**0**
2	0	3	3	3 →	**3**
3	0	3	4	4 →	**4**
4	0	3	4	5 →	**5**
5	0	3	7	7	

$i=3$

$b_i=5$

$w_i=4$

$w=1..4$

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

Example (17)

Items:

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

W \ i	0	1	2	3	4
0	0	0	0	0	0
1	0	0	0	0	0
2	0	3	3	3	3
3	0	3	4	4	4
4	0	3	4	5	5
5	0	3	7	7 →	**7**

$i=3$

$b_i=5$

$w_i=4$

$w=5$

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] // $w_i > w$

Optimal Solution: Algorithm to Look Up the Table to Find the Selected Items

- When calculating the table, you are interested in B[n][W] which is the maximum value obtained when selecting all n items with the weight limit W.
- If **B[n][W] = B[n – 1][W]** then package n is not selected, you trace B[n – 1][W].
- If **B[n][W] ≠ B[n – 1][W]**, you notice that the optimal selection has the package n and trace B[n – 1][W – w_n].

Example

Items:

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

W \ i	0	1	2	3	4
0	0	0	0	0	0
1	0	0	0	0	0
2	0	3	3	3	3
3	0	3	4	4	4
4	0	3	4	5	5
5	0	3	7	7	**7**

Parameters to function call will be n and W.
PrintSelectedItems(n, W)

```
PrintSelectedItems(i,w)
 While i > 0
    if  B[i-1,w] ≠ B[i,w]
          Print "item i"
          w = w – w_i
    i = i-1
```

Example

W \ i	0	1	2	3	4
0	0	0	0	0	0
1	0	0	0	0	0
2	0	3	3	3	3
3	0	3	4	4	4
4	0	3	4	5	5
5	0	3	7	7	**7**

Items:

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

PrintSelectedItems(i,w)
  While i > 0
    if B[i-1,w] $\neq$ B[i,w]
      Print “item i”
      $w = w - w_i$
    i = i-1

Example

Items:

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

W \ i	0	1	2	3	4
0	0	0	0	0	0
1	0	0	0	0	0
2	0	3	3	3	3
3	0	3	4	4	4
4	0	3	4	5	5
5	0	3	7	7	**7**

PrintSelectedItems(i,w)
  While i > 0
    if B[i-1,w] $\neq$ B[i,w]
      Print "item i"
      $w = w - w_i$
    i = i-1

Example

W \ i	0	1	2	3	4
0	0	0	0	0	0
1	0	0	0	0	0
2	0	3	3	3	3
3	0	3	4	4	4
4	0	3	4	5	5
5	0	3	7	7	**7**

Items:

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

Item 2,

PrintSelectedItems(i,w)

While i > 0

if B[i-1,w] $\neq$ B[i,w]

Print "item i"

$w = w - w_i$

i = i-1

Example

Items:

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

W \ i	0	1	2	3	4
0	0	0	0	0	0
1	0	0	0	0	0
2	0	3	3	3	3
3	0	3	4	4	4
4	0	3	4	5	5
5	0	3	7	7	**7**

Item 2, Item 1

PrintSelectedItems(i,w)

While i > 0

if B[i-1,w] ≠ B[i,w]

Print “item i”

$w = w - w_i$

i = i-1

items i $W = 20$

w	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	0	0
2	0	3	3	3	3	3
3	0	3	4	4	4	4
4	0	3	4	5	5	5
5	0	3	7	7	8	8
6	0	3	7	8	8	8
7	0	3	7	9	9	9
8	0	3	7	9	9	9
9	0	3	7	12	13	13
10	0	3	7	12	13	13
11	0	3	7	12	16	16
⋮						
20	0	3	7	12	20	26

$(8 - 4 = 2)$
$w - w_3$

weight

$(11 - 5 = 6)$
$w - w_4$

$20 - 9 =$
$20 - w_5 =$

Items i	Weight w_i	Benefit b_i
1	2	3
2	3	4
3	4	5
4	5	8
5	9	10

Items 1, 3, 4, 5

You cannot print items by only looking at last row