Dynamic Programming

0/1 Knapsack

0/1 knapsack

We've some objects/items that we want to sell in the market. Each item has some weight and a profit/value.

We've a bag/knapsack of some capacity that we'll use to carry the objects to market for selling.

Lets suppose:

number of objects = n = 4

Profit/value = P = {1, 2, 5, 6}

Weight = W = {2, 3, 4, 5}   total weight = 14

Knapsack capacity = m = 8

0/1 knapsack

We've some objects/items that we want to sell in the market. Each

number of objects = n = 4

Profit/value = P = {1, 2, 5, 6}

Weight = W = {2, 3, 4, 5} total weight = 14

Knapsack capacity = m = 8

Solution:

A set x = {1,0,1,,,,}

Each entry corresponds to an object.

$x_i = 1$ if object i is added to the knapsack

$x_i = 0$ if object i is not added to the knapsack

0/1 knapsack

Goal:

1: maximize ($\sum p_i$)

2: $\sum w_i$ <= m

0/1 knapsack – Brute Force Algorithm

One solution is to consider all possible solutions and pick the best one

Possible solutions: (for each item we've two option: add it or don't add it)

$x = \{0, 0, 0, 0\}$ no item added to knapsack

$x = \{1, 1, 1, 1\}$ all items added to knapsack

$x = \{1, 0, 0, 0\}$

$x = \{0, 1, 0, 0\}$

.

.

.

There are 2^4 possible solutions for 4 objects

There are 2^n possible solutions for n objects

Complexity = O(2^n)

0/1 knapsack
Identify sub-problems (optimal substructure)

- If items are labeled *1..n*, then a sub problem would be to find an optimal solution for S_i *= {items labeled 1, 2, .. i}*

- can we describe the final solution (S_n) in terms of sub-problems (S_i)?

Sub-problems (optimal substructure)

- Let capacity = 20
- Let S_5 = original problem with 5 items
- Lets now define a sub-problem
- Let S_4 = sub-problem with first 4 items

S_5 S_4

Item #	weight	value
1	2	3
2	3	4
3	4	5
4	5	8
5	9	10

Sub-problems (optimal substructure)

- Let capacity = 20
- Let S_5 = original problem with 5 items
- Lets now define a sub-problem
- Let S_4 = sub-problem with first 4 items

Optimal solution for $S_{5:}$

Add Items 1, 3, 4, 5 to the bag.

Max weight = 20, max value = 26

	Item #	**weight**	**value**
	1	2	3
S_4	2	3	4
S_5	3	4	5
	4	5	8
	5	9	10

Sub-problems (optimal substructure)

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

Sub-problems (optimal substructure)

- Let capacity = 20
- Let S_5 = original problem with 5 items
- Lets now define a sub-problem
- Let S_4 = sub-problem with first 4 items

Optimal solution for $S_{4:}$

Add Items 1, 2, 3, 4 to the bag.

Max weight = 14, max value = 20

		Item #	weight	value
		1	2	3
	S_4	2	3	4
S_5		3	4	5
		4	5	8
		5	9	10

Sub-problems (optimal substructure)

- Optimal solution for S_5 does not contain optimal solution for sub problem $S_{4.}$

Re-define Sub-problems

- We have two parameters. Number of items and capacity of knapsack. We were not considering capacity of knapsack earlier. Lets incorporate this as well.

Sub-problems (optimal substructure)

- Let capacity = 20
- Let $S_{5,20}$ = original problem with 5 items
- We can define a sub-problem in two ways:
  - Let $S_{4,11}$ = sub-problem when 5th item is a part of optimal solution
  - Let $S_{4,20}$ = sub-problem when 5th item is not a part of optimal solution.

Optimal solution for $S_{5,20}$:

Add Items 1, 3, 4, 5 to the bag. Total weight = 20, value = 26

Optimal solution for $S_{4,20}$:

Add Items 1, 2, 3, 4 to the bag. Total weight = 14, value = 20

Optimal solution for $S_{4,11}$:

Add Items 1, 3, 4 to the bag. Total weight = 11, value = 16

S_4 S_5

Item #	weight	value
1	2	3
2	3	4
3	4	5
4	5	8
5	9	10

0/1 Knapsack – DP recursive formula

- P is set of profits/prices/values/benefit.
- W is set of weights of the items.

$$K[i, j] = \begin{cases} K[i-1, j] & \text{if } w_i > j \\ \max\{(p_i + K[i-1][j-w_i],\ K[i-1][j])\} & \end{cases}$$

It means, that the best subset of S_i that has total weight *j* is one of these two:
1) the best subset of S_{i-1} that has total weight *j*, **or**
2) the best subset of S_{i-1} that has total weight $j - w_i$ plus the item *i*

0/1 Knapsack – DP Algortihm

```
for j = 0 to m
        K[0,j] = 0
for i = 0 to n
        K[i,0] = 0
for i = 0 to n
        for j = 0 to m
                if w_i <= j                                     // item i can be part of the solution
                        if p_i + K[i-1,j- w_i] > K[i-1,j]
                                K[i,j] = p_i + K[i-1,j- w_i]
                        else
                                K[i,j] = K[i-1,j]
                else K[i,j] = K[i-1,j]                          // w_i > j
```

0/1 Knapsack - DP Solution

Time complexity = $O(n*m)$

Dry Run

if $w_i <= j$

  if $p_i + K[i-1, j-w_i] > K[i-1, j]$

    $K[i,j] = p_i + K[i-1, j-w_i]$

  else

    $K[i,j] = K[i-1,j]$

else $K[i,j] = K[i-1,j]$

number of objects = n = 4

Weight= W = {2, 3, 4, 5}

Profit/value = P = {1, 2, 5, 6}

Knapsack capacity = m = 8

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0								
2	0								
3	0								
4	0								

1st row and 2nd column means we are considering 1st item only and the capacity of sack is 2.

Dry Run

if $w_i <= j$

if $p_i + K[i-1,j-w_i] > K[i-1,j]$

$K[i,j] = p_i + K[i-1,j-w_i]$

else

$K[i,j] = K[i-1,j]$

else $K[i,j] = K[i-1,j]$

number of objects = n = 4

Weight= W = {2, 3, 4, 5}

Profit/value = P = {1, 2, 5, 6}

Knapsack capacity = m = 8

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0							
2	0								
3	0								
4	0								

This entry is zero because capacity is 1. We are considering 1st item only and its weight is 2, so we cant add it to the sack

Dry Run

if $w_i <= j$

  if $p_i + K[i-1, j- w_i] > K[i-1,j]$

    $K[i,j] = p_i + K[i-1, j- w_i]$

  else

    $K[i,j] = K[i-1,j]$

else $K[i,j] = K[i-1,j]$

number of objects = n = 4

Weight= W = {2, 3, 4, 5}

Profit/value = P = {1, 2, 5, 6}

Knapsack capacity = m = 8

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	1						
2	0								
3	0								
4	0								

Capacity is now 2. We are considering 1st item only and its weight is 2, so we can add it to the sack. The profit value of first item is 1, so we write 1 here.

Dry Run

if $w_i <= j$

    if $p_i + K[i-1, j - w_i] > K[i-1, j]$

        $K[i,j] = p_i + K[i-1, j - w_i]$

    else

        $K[i,j] = K[i-1,j]$

else $K[i,j] = K[i-1,j]$

number of objects = n = 4

Weight= W = {2, 3, 4, 5}

Profit/value = P = {1, 2, 5, 6}

Knapsack capacity = m = 8

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	1	1					
2	0								
3	0								
4	0								

Dry Run

if $w_i <= j$

if $p_i + K[i-1,j- w_i] > K[i-1,j]$

$K[i,j] = p_i + K[i-1,j- w_i]$

else

$K[i,j] = K[i-1,j]$

else $K[i,j] = K[i-1,j]$

number of objects = n = 4

Weight= W = {2, 3, 4, 5}

Profit/value = P = {1, 2, 5, 6}

Knapsack capacity = m = 8

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	1	1	1				
2	0								
3	0								
4	0								

Dry Run

if w_i <= j

if $p_i + K[i-1,j-w_i] > K[i-1,j]$

$K[i,j] = p_i + K[i-1,j-w_i]$

else

$K[i,j] = K[i-1,j]$

else $K[i,j] = K[i-1,j]$

number of objects = n = 4

Weight= W = {2, 3, 4, 5}

Profit/value = P = {1, 2, 5, 6}

Knapsack capacity = m = 8

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	1	1	1	1			
2	0								
3	0								
4	0								

Dry Run

if $w_i <= j$

if $p_i + K[i-1, j-w_i] > K[i-1,j]$

$K[i,j] = p_i + K[i-1, j-w_i]$

else

$K[i,j] = K[i-1,j]$

else $K[i,j] = K[i-1,j]$

number of objects = n = 4

Weight= W = {2, 3, 4, 5}

Profit/value = P = {1, 2, 5, 6}

Knapsack capacity = m = 8

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	1	1	1	1	1	1	1
2	0								
3	0								
4	0								

Capacity is increasing but we are considering 1st item only so this whole row remains 1, which is the profit of item 1.

Dry Run

if $w_i <= j$

if $p_i + K[i-1, j-w_i] > K[i-1,j]$

$K[i,j] = p_i + K[i-1, j-w_i]$

else

$K[i,j] = K[i-1,j]$

else $K[i,j] = K[i-1,j]$

number of objects = n = 4

Weight= W = {2, 3, 4, 5}

Profit/value = P = {1, 2, 5, 6}

Knapsack capacity = m = 8

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	1	1	1	1	1	1	1
2	0	0	1						
3	0								
4	0								

Lets fill the second row. Here we'll consider two items now.

Only First item added to the sack

Dry Run

if w_i <= j
    if $p_i + K[i-1,j-w_i] > K[i-1,j]$
        $K[i,j] = p_i + K[i-1,j-w_i]$
    else
        $K[i,j] = K[i-1,j]$
else $K[i,j] = K[i-1,j]$

number of objects = n = 4

Weight= W = {2, 3, 4, 5}

Profit/value = P = {1, 2, 5, 6}

Knapsack capacity = m = 8

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	1	1	1	1	1	1	1
2	0	0	1	2					
3	0								
4	0								

Lets fill the second row. Here we'll consider two items now.

Only 2nd item added to the sack as capacity of sack is 3. This 2 is the profit value of 2nd item

Dry Run

if $w_i \le j$
    if $p_i + K[i-1, j-w_i] > K[i-1, j]$
        $K[i,j] = p_i + K[i-1, j-w_i]$
    else
        $K[i,j] = K[i-1,j]$
else $K[i,j] = K[i-1,j]$

number of objects = n = 4

Weight= W = {2, 3, 4, 5}

Profit/value = P = {1, 2, 5, 6}

Knapsack capacity = m = 8

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	1	1	1	1	1	1	1
2	0	0	1	2	2				
3	0								
4	0								

Lets fill the second row. Here we'll consider two items now.

Only 2nd item added to the sack. As capacity of sack is 4, we can't add both items. The profit value of 2nd item is more so we add it instead of 1st item.

Dry Run

if w_i <= j
    if $p_i + K[i-1, j-w_i] > K[i-1,j]$
        $K[i,j] = p_i + K[i-1, j-w_i]$
    else
        $K[i,j] = K[i-1,j]$
else $K[i,j] = K[i-1,j]$

number of objects = n = 4

Profit/value = P = {1, 2, 5, 6}

Weight= W = {2, 3, 4, 5}

Knapsack capacity = m = 8

Lets fill the second row. Here we'll consider two items now.

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	1	1	1	1	1	1	1
2	0	0	1	2	2	3			
3	0								
4	0								

As capacity of sack is 5, we can add both items. So, total profit becomes 3.

Dry Run

if $w_i <= j$

    if $p_i + K[i-1, j-w_i] > K[i-1,j]$

        $K[i,j] = p_i + K[i-1, j-w_i]$

    else

        $K[i,j] = K[i-1,j]$

else $K[i,j] = K[i-1,j]$

number of objects = n = 4

Weight= W = {2, 3, 4, 5}

Profit/value = P = {1, 2, 5, 6}

Knapsack capacity = m = 8

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	1	1	1	1	1	1	1
2	0	0	1	2	2	3	3	3	3
3	0								
4	0								

Dry Run

if $w_i <= j$

    if $p_i + K[i-1, j-w_i] > K[i-1,j]$

        $K[i,j] = p_i + K[i-1, j-w_i]$

    else

        $K[i,j] = K[i-1,j]$

else $K[i,j] = K[i-1,j]$

number of objects = n = 4

Weight= W = {2, 3, 4, 5}

Profit/value = P = {1, 2, 5, 6}

Knapsack capacity = m = 8

Lets fill the third row. Here we'll consider three items now.

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	1	1	1	1	1	1	1
2	0	0	1	2	2	3	3	3	3
3	0	0	1						
4	0								

Only 1st item added to the sack.

Dry Run

if $w_i <= j$

  if $p_i + K[i-1, j-w_i] > K[i-1, j]$

    $K[i,j] = p_i + K[i-1, j-w_i]$

  else

    $K[i,j] = K[i-1,j]$

else $K[i,j] = K[i-1,j]$

number of objects = n = 4

Weight= W = {2, 3, 4, 5}

Profit/value = P = {1, 2, 5, 6}

Knapsack capacity = m = 8

Lets fill the third row. Here we'll consider three items now.

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	1	1	1	1	1	1	1
2	0	0	1	2	2	3	3	3	3
3	0	0	1	2					
4	0								

Only 2nd item added to the sack.

Dry Run

if $w_i <= j$
    if $p_i + K[i-1, j-w_i] > K[i-1, j]$
        $K[i,j] = p_i + K[i-1, j-w_i]$
    else
        $K[i,j] = K[i-1, j]$
else $K[i,j] = K[i-1, j]$

number of objects = n = 4

Weight= W = {2, 3, 4, 5}

Profit/value = P = {1, 2, 5, 6}

Knapsack capacity = m = 8

Lets fill the third row. Here we'll consider three items now.

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	1	1	1	1	1	1	1
2	0	0	1	2	2	3	3	3	3
3	0	0	1	2	5				
4	0								

Only 3rd item added to the sack.

Dry Run

if $w_i <= j$

if $p_i + K[i-1, j-w_i] > K[i-1, j]$

$K[i,j] = p_i + K[i-1, j-w_i]$

else

$K[i,j] = K[i-1,j]$

else $K[i,j] = K[i-1,j]$

number of objects = n = 4

Weight= W = {2, 3, 4, 5}

Profit/value = P = {1, 2, 5, 6}

Knapsack capacity = m = 8

Lets fill the third row. Here we'll consider three items now.

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	1	1	1	1	1	1	1
2	0	0	1	2	2	3	3	3	3
3	0	0	1	2	5	5			
4	0								

Only 3rd item added to the sack.

Dry Run

if $w_i \le j$

if $p_i + K[i-1, j-w_i] > K[i-1, j]$

$K[i,j] = p_i + K[i-1, j-w_i]$

else

$K[i,j] = K[i-1,j]$

else $K[i,j] = K[i-1,j]$

number of objects = n = 4

Weight= W = {2, 3, 4, 5}

Profit/value = P = {1, 2, 5, 6}

Knapsack capacity = m = 8

Lets fill the third row. Here we'll consider three items now.

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	1	1	1	1	1	1	1
2	0	0	1	2	2	3	3	3	3
3	0	0	1	2	5	5	6		
4	0								

1st & 3rd items added to the sack.

Dry Run

if $w_i \le j$
    if $p_i + K[i-1, j-w_i] > K[i-1,j]$
        $K[i,j] = p_i + K[i-1, j-w_i]$
    else
        $K[i,j] = K[i-1,j]$
else $K[i,j] = K[i-1,j]$

number of objects = n = 4

Weight= W = {2, 3, 4, 5}

Profit/value = P = {1, 2, 5, 6}

Knapsack capacity = m = 8

Lets fill the third row. Here we'll consider three items now.

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	1	1	1	1	1	1	1
2	0	0	1	2	2	3	3	3	3
3	0	0	1	2	5	5	6	7	
4	0								

2^{nd} & 3^{rd} items added to the sack.

Dry Run

if $w_i <= j$

  if $p_i + K[i-1, j-w_i] > K[i-1, j]$

    $K[i,j] = p_i + K[i-1, j-w_i]$

  else

    $K[i,j] = K[i-1,j]$

else $K[i,j] = K[i-1,j]$

number of objects = n = 4

Weight= W = {2, 3, 4, 5}

Profit/value = P = {1, 2, 5, 6}

Knapsack capacity = m = 8

Lets fill the third row. Here we'll consider three items now.

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	1	1	1	1	1	1	1
2	0	0	1	2	2	3	3	3	3
3	0	0	1	2	5	5	6	7	7
4	0								

2nd & 3rd items added to the sack.

Dry Run

if $w_i <= j$

if $p_i + K[i-1, j-w_i] > K[i-1, j]$

$K[i,j] = p_i + K[i-1, j-w_i]$

else

$K[i,j] = K[i-1, j]$

else $K[i,j] = K[i-1, j]$

number of objects = n = 4

Weight= W = {2, 3, 4, 5}

Profit/value = P = {1, 2, 5, 6}

Knapsack capacity = m = 8

Lets fill the fourth row. Here we'll consider four items now.

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	1	1	1	1	1	1	1
2	0	0	1	2	2	3	3	3	3
3	0	0	1	2	5	5	6	7	7
4	0	0							

Dry Run

if $w_i \le j$
    if $p_i + K[i-1, j-w_i] > K[i-1, j]$
        $K[i,j] = p_i + K[i-1, j-w_i]$
    else
        $K[i,j] = K[i-1, j]$
else $K[i,j] = K[i-1, j]$

number of objects = n = 4

Weight= W = {2, 3, 4, 5}

Profit/value = P = {1, 2, 5, 6}

Knapsack capacity = m = 8

Lets fill the fourth row. Here we'll consider four items now.

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	1	1	1	1	1	1	1
2	0	0	1	2	2	3	3	3	3
3	0	0	1	2	5	5	6	7	7
4	0	0	1	2	5	6			

Only 4th item added to the sack.

Dry Run

if $w_i <= j$

  if $p_i + K[i-1, j-w_i] > K[i-1, j]$

    $K[i,j] = p_i + K[i-1, j-w_i]$

  else

    $K[i,j] = K[i-1,j]$

else $K[i,j] = K[i-1,j]$

number of objects = n = 4

Weight= W = {2, 3, 4, 5}

Profit/value = P = {1, 2, 5, 6}

Knapsack capacity = m = 8

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	1	1	1	1	1	1	1
2	0	0	1	2	2	3	3	3	3
3	0	0	1	2	5	5	6	7	7
4	0	0	1	2	5	6	6		

Lets fill the fourth row. Here we'll consider four items now.

Only 4th item added to the sack.

Dry Run

if $w_i <= j$

if $p_i + K[i-1, j-w_i] > K[i-1, j]$

$K[i,j] = p_i + K[i-1, j-w_i]$

else

$K[i,j] = K[i-1, j]$

else $K[i,j] = K[i-1, j]$

number of objects = n = 4

Weight= W = {2, 3, 4, 5}

Profit/value = P = {1, 2, 5, 6}

Knapsack capacity = m = 8

Lets fill the fourth row. Here we'll consider four items now.

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	1	1	1	1	1	1	1
2	0	0	1	2	2	3	3	3	3
3	0	0	1	2	5	5	6	7	7
4	0	0	1	2	5	6	6	7	

Only 2nd & 3rd items added to the sack.

Dry Run

if $w_i <= j$
    if $p_i + K[i-1, j-w_i] > K[i-1,j]$
        $K[i,j] = p_i + K[i-1, j-w_i]$
    else
        $K[i,j] = K[i-1,j]$
else $K[i,j] = K[i-1,j]$

number of objects = n = 4

Weight= W = {2, 3, 4, 5}

Profit/value = P = {1, 2, 5, 6}

Knapsack capacity = m = 8

Lets fill the fourth row. Here we'll consider four items now.

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	1	1	1	1	1	1	1
2	0	0	1	2	2	3	3	3	3
3	0	0	1	2	5	5	6	7	7
4	0	0	1	2	5	6	6	7	8

2nd & 4th items added to the sack.

0/1 Knapsack - DP Solution

We want a solution in this form:

A set x = {1,0,1,,,,}

Each entry corresponds to an object.

$x_i = 1$ if object i is not added to the knapsack

$x_i = 0$ if object i is not added to the knapsack

0/1 Knapsack - DP Solution

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	1	1	1	1	1	1	1
2	0	0	1	2	2	3	3	3	3
3	0	0	1	2	5	5	6	7	7
4	0	0	1	2	5	6	6	7	8

We can use this table to find our solution set x.

0/1 Knapsack - DP Solution

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	1	1	1	1	1	1	1
2	0	0	1	2	2	3	3	3	3
3	0	0	1	2	5	5	6	7	7
4	0	0	1	2	5	6	6	7	8

Start from last element of the matrix. Profit value is 8. Now see if there's any 8 in previous row. Since there isn't so this profit value has been achieved by adding the 4^{th} item to the sack.

0/1 Knapsack - DP Solution

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	1	1	1	1	1	1	1
2	0	0	1	2	2	3	3	3	3
3	0	0	1	2	5	5	6	7	7
4	0	0	1	2	5	6	6	7	8

Start from last element of the matrix. Profit value is 8. Now see if there's any 8 in previous row. Since there isn't, so this profit value has been achieved by adding the 4^{th} item to the sack.

Profit of 4^{th} item is 6. So, 8-6 = 2 is the remaining value.

0/1 Knapsack - DP Solution

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	1	1	1	1	1	1	1
2	0	0	1	2	2	3	3	3	3
3	0	0	1	2	5	5	6	7	7
4	0	0	1	2	5	6	6	7	8

Look for value 2 in previous (i.e the 3rd) row. We've a 2 in the third row. Look for a 2 in 2nd row now. Since there's also a 2 in 2nd row, so the 2 in 3rd row isn't achieved by adding 3rd item. So, third item won't be added to the sack.

0/1 Knapsack - DP Solution

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	1	1	1	1	1	1	1
2	0	0	1	2	2	3	3	3	3
3	0	0	1	2	5	5	6	7	7
4	0	0	1	2	5	6	6	7	8

So, we still have a profit value of 2 and we want to see which item is added to the sack to get this profit.
Look for 2 in 2^{nd} row now. There is a 2 value there. And since there's no 2 in 1^{st} row so this 2 is coming from adding 2^{nd} item to the sack.

0/1 Knapsack - DP Solution

	0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0	0
1	0	0	1	1	1	1	1	1	1
2	0	0	1	2	2	3	3	3	3
3	0	0	1	2	5	5	6	7	7
4	0	0	1	2	5	6	6	7	8

Remaining profit is now 0. We can use this as a check to stop here. Or We can go back to previous rows and stop when there are no more rows to explore.
So look for 0 in row 1. There is a 0. But there's also a 0 in previous row. By using the same logic as earlier, we wont add 1st item to the sack.

0/1 Knapsack - DP Solution

- So the final solution is:

  x = {0, 1, 0, 1}

- Task: Modify the algorithm to find the optimal solution. (Save the sub-problem from where solution of next sub-problem is formed.)