Knapsack problem

Hyrum D. Carroll

(Based on slides prepared by Suk Jin Lee)

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 ($\Sigma w_i \leq W$). So we must consider weights of items as well as their value.

Item # (i)	Weight (w_i)	Benefit value (b_i)
1	1	8
2	3	6
3	5	5

Knapsack problem

- There are two versions of the problem:
  1) "0-1 knapsack problem"
  2) "Fractional knapsack problem"

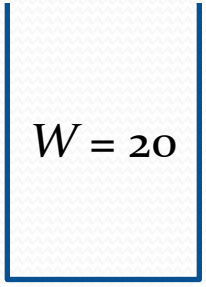
  1) Items are indivisible; you either take an item or not. Solved with ***dynamic programming***
  2) Items are divisible: you can take any fraction of an item. Solved with a ***greedy algorithm (covered later)***

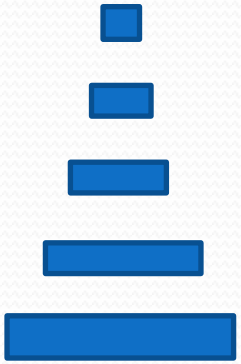
0-1 Knapsack problem

- 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

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





Item # (i)	Weight (w_i)	Benefit value (b_i)
1	2	3
2	3	4
3	4	5
4	5	8
5	9	10

0-1 Knapsack problem

- Problem, in other words, is to find

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

- 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.

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 the most total value and with total weight less or equal to W
- Running time
  - $O(2^n)$

Brute-force approach

- 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 = {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 can't do that.

Defining a Subproblem

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

For S_4 (w_1, w_2, w_3, w_4)
Total weight: 14; Total benefit: 20

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

For S_5 (w_1, w_3, w_4, w_5)
Total weight: 20; Total benefit: 26

Max weight: $W = 20$

Item # (i)	Weight (w_i)	Benefit value (b_i)
1	2	3
2	3	4
3	4	5
4	5	8
5	9	10

S_4

S_5

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

Item # to include

Recursive Formula for subproblems

- Recursive formula for subproblems:

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

- 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

Recursive Formula

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

- Second case: $w_k \le w$. Then the item k can be in the solution, and we choose the case with greater value
- It means, that the best subset of S_k with the total weight w is one of the two:
  - the best subset of $S_{k\text{-}1}$ that has total weight w, **or**
  - the best subset of $S_{k\text{-}1}$ that has total weight $w - w_k$ plus the item k
- The best subset of S_k that has the total weight w, either contains item k or not.

0-1 Knapsack Algorithm

for $w = 0$ to W // O(W)
    B[0, w] = 0
for $i = 0$ **to** n // Repeat n times
    B[i, 0] = 0
    for $w = 0$ **to** W // O(W)
        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
            $B[i, w] = B[i-1, w]$ // $w_i > w$

What is the running time of this algorithm? **O($n \times W$)**

Example

- Let's run the 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

$B(i, W)$

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

for $w = 0$ to W

$B[0, w] = 0$

Items:

(w_i, b_i)

1: (2, 3)

2: (3, 4)

3: (4, 5)

4: (5, 6)

Example

$B(i, W)$

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

$B[i, 0] = 0$

Items:

(w_i, b_i)

1: (2, 3)

2: (3, 4)

3: (4, 5)

4: (5, 6)

Example

Items:

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

$B(i, W)$

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

$i = 1$

$w = 1$

$w_1 = 2$

$b_1 = 3$

$w - w_1 = -1$

if $w_i \le w$ // 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

$\mathbf{B[i, w] = B[i-1, w]}$ // $w_i > w$

Example

Items:

(w_i, b_i)

1: (2, 3)

2: (3, 4)

3: (4, 5)

4: (5, 6)

$B(i, W)$

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				

$i = 1$

$w = 2$

$w_1 = 2$

$b_1 = 3$

$w - w_1 = 0$

if $w_i \le w$ // 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

Items:

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

$B(i, W)$

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				

$i = 1$

$w = 3$

$w_1 = 2$

$b_1 = 3$

$w - w_1 = 1$

if $w_i \le w$ // 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

Items:

(w_i, b_i)

1: (2, 3)

2: (3, 4)

3: (4, 5)

4: (5, 6)

$B(i, W)$

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				

$i = 1$

$w = 4$

$w_1 = 2$

$b_1 = 3$

$w - w_1 = 2$

if $w_i \le w$ // 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

Items:

(w_i, b_i)

1: (2, 3)

2: (3, 4)

3: (4, 5)

4: (5, 6)

$B(i, W)$

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			

$i = 1$

$w = 5$

$w_1 = 2$

$b_1 = 3$

$w - w_1 = 3$

if $w_i \le w$ // 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

Items:

(w_i, b_i)

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

$B(i, W)$

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			

$i = 2$

$w = 1$

$w_2 = 3$

$b_2 = 4$

$w - w_2 = -2$

if $w_i \le w$ // 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

Items:

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

$B(i, W)$

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			

$i = 2$

$w = 2$

$w_2 = 3$

$b_2 = 4$

$w - w_2 = -1$

if $w_i \le w$ // 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

$\boldsymbol{B[i, w] = B[i-1, w]}$ // $w_i > w$

Example

Items:

(w_i, b_i)

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

$B(i, W)$

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			

$i = 2$

$w = 3$

$w_2 = 3$

$b_2 = 4$

$w - w_2 = 0$

if $w_i \le w$ // 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

Items:

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

$B(i, W)$

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			

$i = 2$

$w = 4$

$w_2 = 3$

$b_2 = 4$

$w - w_2 = 1$

if $w_i \le w$ // 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

Items:

(w_i, b_i)

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

$B(i, W)$

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**		

$i = 2$

$w = 5$

$w_2 = 3$

$b_2 = 4$

$w - w_2 = 2$

if $w_i \le w$ // 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

Items:

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

$B(i, W)$

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		

$i = 3$

$w = 1, 2, 3$

$w_3 = 4$

$b_3 = 5$

$w - w_3 < 0$

if $w_i \le w$ // 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

Items:

(w_i, b_i)

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

$B(i, W)$

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		

$i = 3$

$w = 4$

$w_3 = 4$

$b_3 = 5$

$w - w_3 = 0$

if $w_i \le w$ // 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

Items:

(w_i, b_i)

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

$B(i, W)$

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**		

$i = 3$

$w = 5$

$w_3 = 4$

$b_3 = 5$

$w - w_3 = 1$

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

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

Example

Items:

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

$B(i, W)$

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	

$i = 4$

$w = 1, 2, 3, 4$

$w_4 = 5$

$b_4 = 6$

$w - w_4 < 0$

if $w_i \le w$ // 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

  $\boldsymbol{B[i, w] = B[i-1, w]}$ // $w_i > w$

Example

Items:

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

$B(i, W)$

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**

$i = 4$

$w = 5$

$w_4 = 5$

$b_4 = 6$

$w - w_4 = 0$

if $w_i \le w$ // 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

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

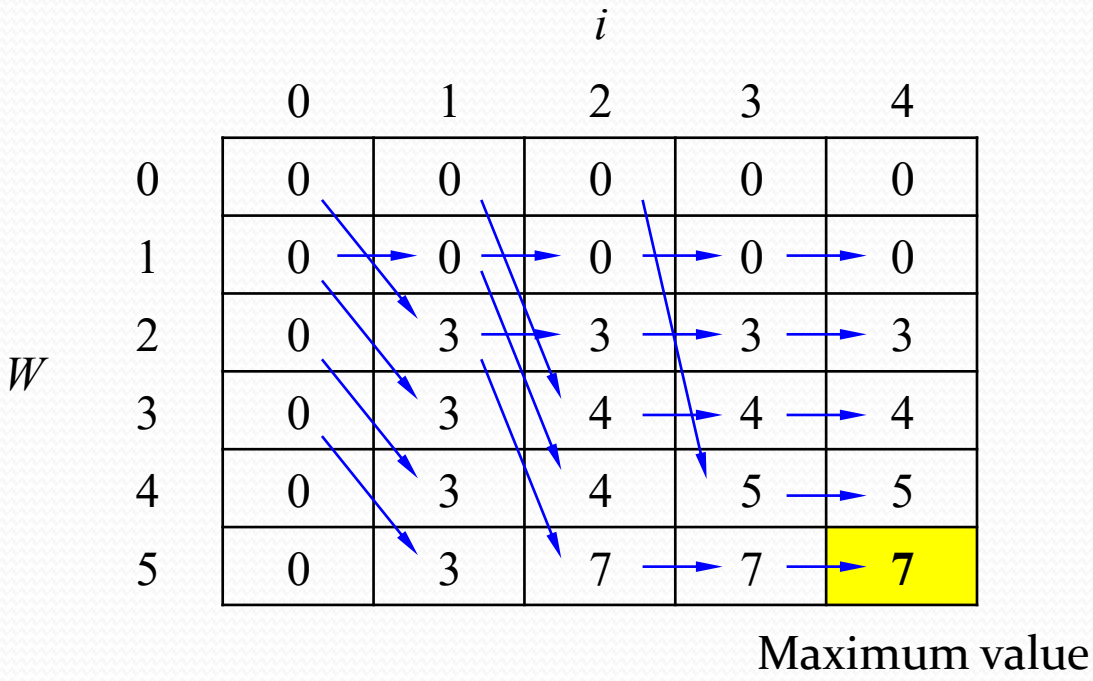
else

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

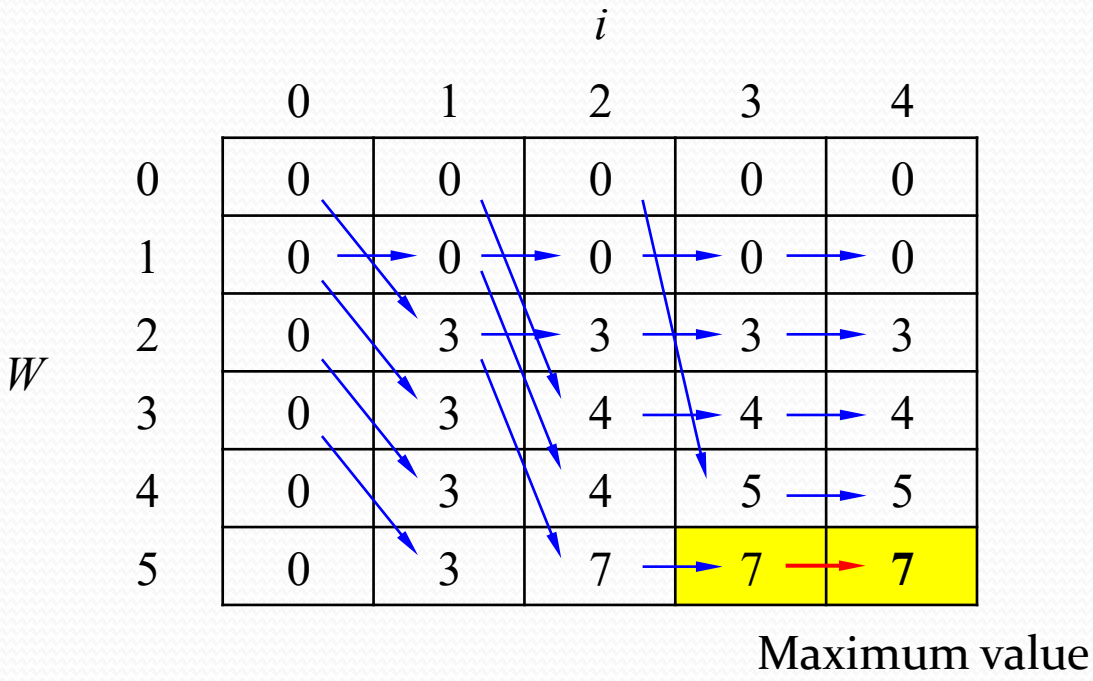
Comments

- This algorithm only finds the max possible value that can be carried in the knapsack.
- To know the items that make this maximum value, an addition to this algorithm is necessary.
- How to extract this data from the table we built?

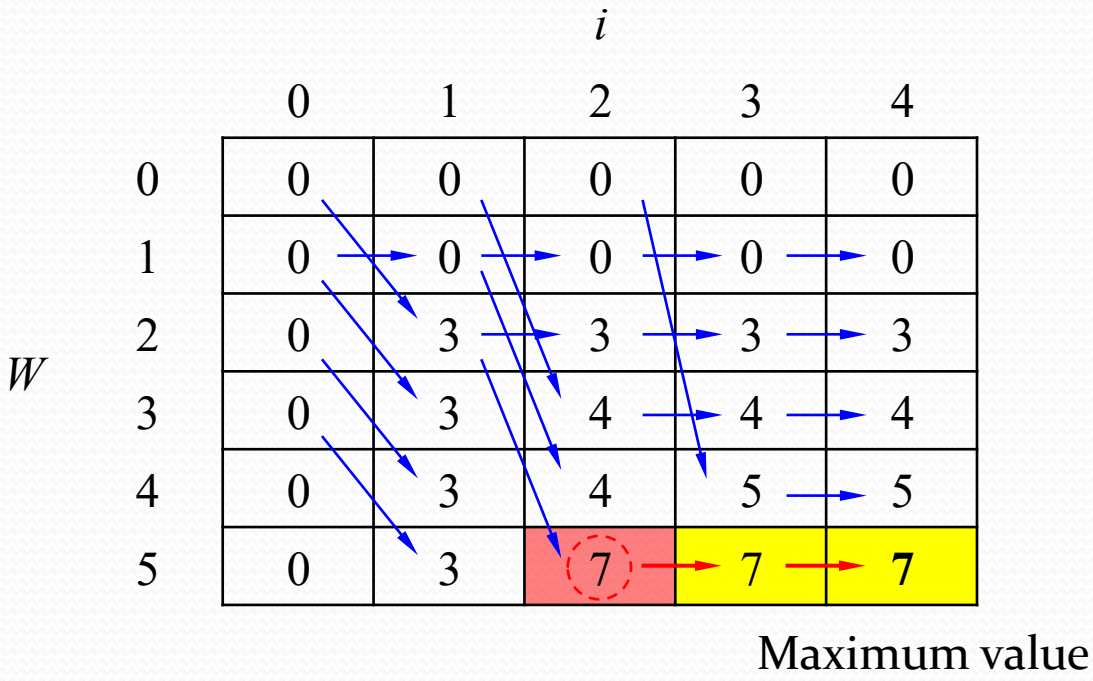
References


i
0 1 2 3 4
W
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
Maximum value

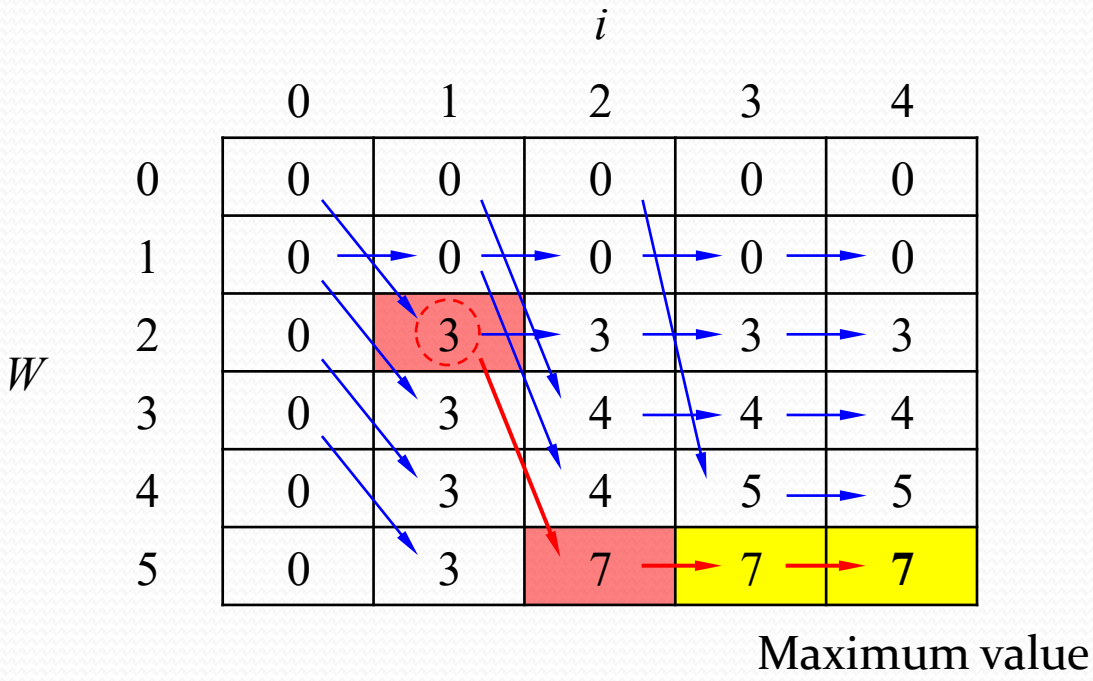
References


i
0 1 2 3 4
W
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
Maximum value

References


i
0 1 2 3 4
W
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
Maximum value

References


i
0 1 2 3 4
W
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
Maximum value