MEMORY FUNCTIONS

- The classic dynamic programming approach to solve a Knapsack Problem, works in bottom-up manner: it fills a table with solutions to *all* smaller subproblems, but each of them is solved only once.
- Solutions to some of these smaller subproblems are often not necessary for getting a solution to the problem given.
- The method that solves only subproblems that are necessary and does it only once is by using Memory Functions.
- Initially, all the table's entries are initialized with a special "null" symbol to indicate that they have not yet been calculated.
- Thereafter, whenever a new value needs to be calculated, the method checks the corresponding entry in the table first: if this entry is not "null," it is simply retrieved from the table; otherwise, it is computed by the recursive call whose result is then recorded in the table.
- The following algorithm implements this idea for the knapsack problem.
- After initializing the table, the recursive function needs to be called with i = *n* (the number of items) and *j* = *W* (the capacity of the knapsack).

```
ALGORITHM MFKnapsack(i, j)
//Implements the memory function method for the knapsack problem
//Input: A nonnegative integer i indicating the number of items being considered and a
//         nonnegative integer j indicating  the knapsack's capacity
//Output: The value of an optimal feasible subset of the first i items
//Note: Uses as global variables input arrays Weights[l .. n ], Values[l .. n ],
// and table V[0 .. n, 0 .. W] whose entries are initialized with -l's except for
//row 0 and column 0 initialized with 0's
if  V[i,j] < 0
    if  j < Weights[i]
        value ← MFKnapsack(i- 1, j)
    else
        value ← max(MFKnapsack(i-1,j), Values[i] + MFKnapsack(i - 1, j - Weights[i]))
    V[i, j] ← value
return V[i, j]
```

EXAMPLE

Solve the given Knapsack problem using memory functions

item	weight	value
1	2	$12
2	1	$10
3	3	$20
4	2	$15

capacity $W = 5$

Solution

i		0	1	2	3	4	5
	0	0	0	0	0	0	0
$w_1 = 2, v_1 = 12$	1	0	-	-	-	-	-
$w_2 = 1, v_2 = 10$	2	0	-	-	-	-	-
$w_3 = 3, v_3 = 20$	3	0	-	-	-	-	-
$w_4 = 2, v_4 = 15$	4	0	-	-	-	-	-

$V[4,5] = \max\{V[3,5], v_4 + V[3,3]\} = \max\{\ ,\ 15 +\ \} =$ {here, $j - w_i \geq 0$}

$V[3,5] = \max\{V[2,5], v_3 + V[2,2]\} = \max\{\ ,\ 20 +\ \} =$ {here, $j - w_i \geq 0$}

$V[2,5] = \max\{V[1,5], v_2 + V[1,4]\} = \max\{\ ,\ 10 +\ \} =$ {here, $j - w_i \geq 0$}

$V[1,5] = \max\{V[0,5], v_1 + V[0,3]\} = \max\{0, 12 + 0\} = 12$ {here, $j - w_i \geq 0$}

i		0	1	2	3	4	5
	0	0	0	0	0	0	0
$w_1 = 2, v_1 = 12$	1	0	-	-	-	-	12
$w_2 = 1, v_2 = 10$	2	0	-	-	-	-	-
$w_3 = 3, v_3 = 20$	3	0	-	-	-	-	-
$w_4 = 2, v_4 = 15$	4	0	-	-	-	-	-

$V[1,4] = \max\{V[0,4], v_1 + V[0,2]\} = \max\{0, 12 + 0\} = 12$ {here, $j - w_i \geq 0$}

i		0	1	2	3	4	5
	0	0	0	0	0	0	0
$w_1 = 2, v_1 = 12$	1	0	-	-	-	12	12
$w_2 = 1, v_2 = 10$	2	0	-	-	-	-	-
$w_3 = 3, v_3 = 20$	3	0	-	-	-	-	-
$w_4 = 2, v_4 = 15$	4	0	-	-	-	-	-

$V[2,5] = \max\{V[1,5], v_2 + V[1,4]\} = \max\{ 12 , 10 + 12 \} = 22$ {here, $j - w_i \geq 0$}

i		0	1	2	3	4	5
	0	0	0	0	0	0	0
$w_1 = 2, v_1 = 12$	1	0	-	-	-	12	12
$w_2 = 1, v_2 = 10$	2	0	-	-	-	-	22
$w_3 = 3, v_3 = 20$	3	0	-	-	-	-	-
$w_4 = 2, v_4 = 15$	4	0	-	-	-	-	-

$V[2,2] = \max\{V[1,2], v_2 + V[1,1]\} = \max\{ \ , 10 + \ \} =$ {here, $j - w_i \geq 0$}

$V[1,2] = \max\{V[0,2], v_1 + V[0,0]\} = \max\{0, 12 + 0\} = 12$ {here, $j - w_i \geq 0$}

$V[1,1] = V[0,1] = 0$ {here, $j - w_i < 0$}

Now

$V[2,2] = \max\{V[1,2], v_2 + V[1,1]\} = \max\{ 12 , 10 + 0 \} = 12$ {here, $j - w_i \geq 0$}

i		0	1	2	3	4	5
	0	0	0	0	0	0	0
$w_1 = 2, v_1 = 12$	1	0	0	12	-	12	12
$w_2 = 1, v_2 = 10$	2	0	-	12	-	-	22
$w_3 = 3, v_3 = 20$	3	0	-	-	-	-	-
$w_4 = 2, v_4 = 15$	4	0	-	-	-	-	-

Now,

$V[3,5] = \max\{V[2,5], v_3 + V[2,2]\} = \max\{22 , 20 + 12\} = 32$ {here, $j - w_i \geq 0$}

i		0	1	2	3	4	5
	0	0	0	0	0	0	0
$w_1 = 2, v_1 = 12$	1	0	0	12	-	12	12
$w_2 = 1, v_2 = 10$	2	0	-	12	-	-	22
$w_3 = 3, v_3 = 20$	3	0	-	-	-	-	32
$w_4 = 2, v_4 = 15$	4	0	-	-	-	-	-

$V[3,3] = \max\{V[2,3], v_3 + V[2,0]\} = \max\{ \ , 20 + 0\} =$ {here, $j - w_i \geq 0$}

$V[2,3] = \max\{V[1,3], v_2 + V[1,2]\} = \max\{ \ , 10 + \ \} =$ {here, $j - w_i \geq 0$}

$V[1,3] = \max\{V[0,3], v_1 + V[0,1]\} = \max\{0, 12 + 0\} = 12$ {here, $j - w_i \geq 0$}

$V[1,2] = \max\{V[0,2], v_1 + V[0,0]\} = \max\{0, 12 + 0\} = 12$ {here, $j - w_i \geq 0$}

Now,

$V[2,3] = \max\{V[1,3], v_2 + V[1,2]\} = \max\{12 , 10 + 12 \} = 22$ {here, $j - w_i \geq 0$}

$V[3,3] = \max\{V[2,3], v_3 + V[2,0]\} = \max\{ 22 , 20 + 0\} = 22$ {here, $j - w_i \geq 0$}

i		0	1	2	3	4	5
	0	0	0	0	0	0	0
$w_1 = 2, v_1 = 12$	1	0	0	12	12	12	12
$w_2 = 1, v_2 = 10$	2	0	-	12	22	-	22
$w_3 = 3, v_3 = 20$	3	0	-	-	22	-	32
$w_4 = 2, v_4 = 15$	4	0	-	-	-	-	-

$V[4,5] = \max\{V[3,5], v_4 + V[3,3]\} = \max\{ 32 , 15 + 22 \} = 37$ {here, $j - w_i \geq 0$}

The final table is

i		0	1	2	3	4	5
	0	0	0	0	0	0	0
$w_1 = 2, v_1 = 12$	1	0	0	12	12	12	12
$w_2 = 1, v_2 = 10$	2	0	-	12	22	-	22
$w_3 = 3, v_3 = 20$	3	0	-	-	22	-	32
$w_4 = 2, v_4 = 15$	4	0	-	-	-	-	37

The maximum value of the sack is **37**

To find Solution Set

The composition of the optimal subset is obtained by tracing back the computations of the entry in the table.

Since $V[4, 5] \neq V[3, 5]$, item 4 was included in the optimal solution

The remaining 3 units of the knapsack capacity is represented by element V[3, 3]. Since V[3,3] = V[2, 3], item 3 is not a part of the optimal subset.

Since $V[2, 3] \neq V[l, 3]$, item 2 is a part of an optimal selection.

Similarly, since $V[1, 2] \neq V[0, 2]$, item 1 is the final part of the optimal solution.

Therefore, solution set **$\{I_1, I_2, I_4\}$**