Knapsack(0-1)

Given two n-tuples

$\{v_1, v_2, v_3 \dots v_n\}$ and $\{w_1, w_2, w_3 \dots w_n\}$,

and, W>0. We wish to determine a subset T such that

Maximizes - ,

Subject to - ≤ W.

Meaning, given n items of weight w_i and value v_i, find the items that should be taken such that the weight is less than the maximum weight W and the corresponding total value is maximum. We can either take the complete item (1) or not (0).

Let A(i,j) represents maximum value that can be attained if the maximum weight is W and items are chosen from 1...i. We have the following recursive definition

$$A(i,j) = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ A(i-1,j) & \text{if } w_i > j \\ \max\{A(i-1,j), v_i + A(i-1,j-w_i)\} & \text{if } w_i \leq j \end{cases}$$

This problem exhibits both overlapping subproblems and optimal substructure and is therefore a good candidate for dynamic programming.

Algorithm:

Firstly, input the total number of items, the weight and value of each item. Then input the maximum weight (maxWeight). Lastly calculate the maximum value that can be attained using Knapsack function.

Knapsack function – This function takes total number of items (items), weight of all the items (weight), value of all the items (value) and the maximum weight (maxWeight) as arguments. It returns the maximum value that can be attained.

Declare dp[items+1][maxWeight+1]. Where, dp[i][w] represents maximum value that can be attained if the maximum weight is w and items are chosen from 1...i.
dp[0][w] = 0 for all w because we have chosen 0 items. And, dp[i][0] = 0 for all w because maximum weight we can take is 0.
Recurrence: for i=1 to items
for w=0 to maxWeight
dp[i][w] = dp[i-1][w], if we do not tale item i. if w-weight[i] >=0, suppose we take this item then, dp[i][w] = max(dp[i][w] , dp[i-1][w-weight[i]]+value[i]). Where, max is a function that returns the maximum of the two arguments it takes.
Return dp[items][maxWeight]

Property:

Time complexity is O(n*W), where n is the total number of items and W is the maximum weight.

Example:

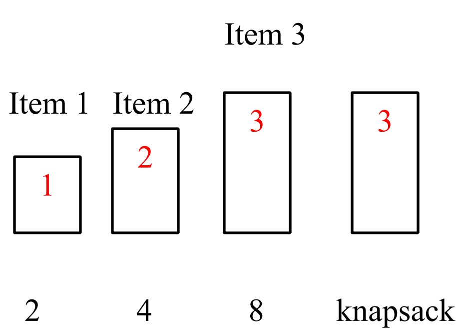
Number of items = 3
Item 1 has weight = 1 and value = 2
Item 2 has weight = 2 and value = 4
Item 3 has weight = 3 and value = 8
Maximum weight = 3
dp[0][0..5] = 0
dp[0..2][0] = 0
Recurrence:
i = 1
w = 0
dp[1][0] = dp[0][0] = 0
0-1 < 0
w = 1
dp[1][1] = dp[0][1] = 0
1-1= 0
dp[1][1] = max (dp[1][1], dp[0][0] + 2) = 2
w = 2
dp[1][2] = dp[0][2] = 0
2- 1 > 0
dp[1][2] = max (dp[1][2], dp[0][1] + 2) = 2
w = 3
dp[1][3] = dp[0][3] = 0
3- 1 > 0
dp[1][3] = max (dp[1][3], dp[0][2] + 2) = 2
i = 2
w = 0
dp[2][0] = dp[1][0] = 0
0-2 < 0
w = 1
dp[2][1] = dp[1][1] = 2
1-2< 0
w = 2
dp[2][2] = dp[1][2] = 2
2- 2 = 0
dp[2][2] = max (dp[2][2], dp[1][2] + 4) = 6
w = 3
dp[2][3] = dp[1][3] = 0

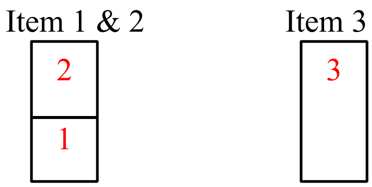
$3- 2 > 0$
dp[2][3] = max (dp[2][3], dp[1][1] + 4) = 6
i = 3
w = 0
dp[3][0] = dp[2][0] = 0
$0-3 < 0$
w = 1
dp[3][1] = dp[2][1] = 2
$1-3 < 0$
w = 2
dp[3][2] = dp[2][2] = 6

$2- 3 < 0$
w = 3
dp[3][3] = dp[2][3] = 0
$3- 3 = 0$
dp[3][3] = max (dp[3][3], dp[2][0] + 8) = 8

dp[3[3] = 8
Output: maximum value that can be attained = 8.

i/w	0	1	2	3
0	0	0	0	0
1	0	0	0	0
2	0	2	2	2
3	0	2	2	6
4	0	2	6	8





$$2 + 4 = 6 \quad < \quad 8$$

So item 3 is the optimal solution.