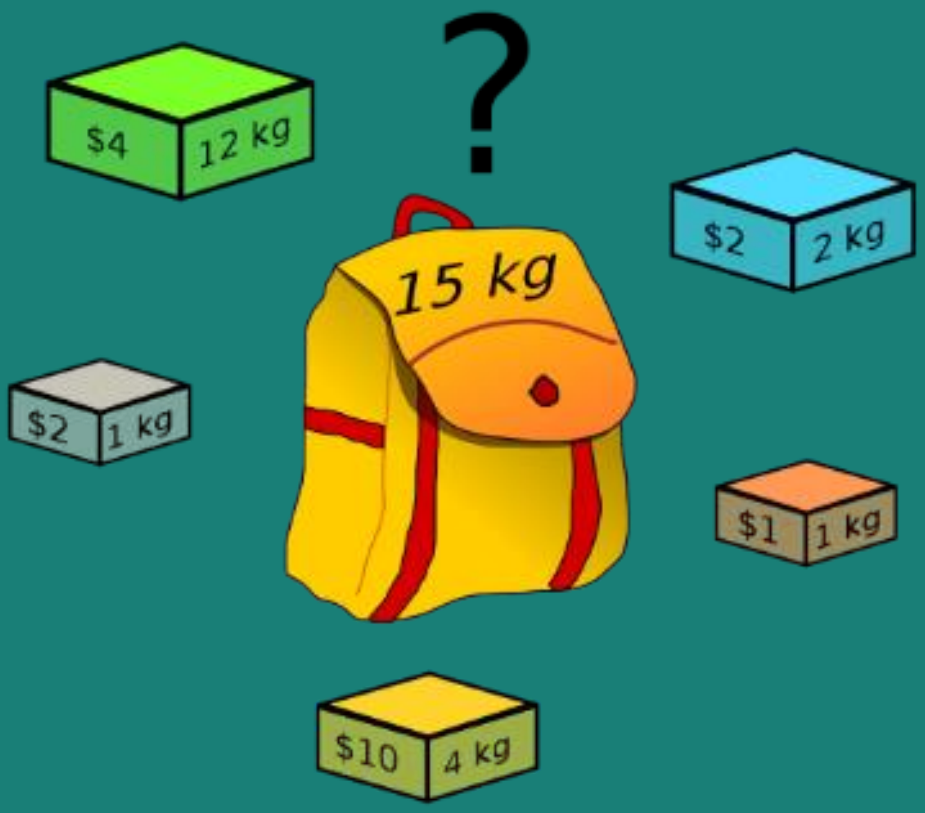


CTA PRACTICAL 10

20MCED08

Practical 6

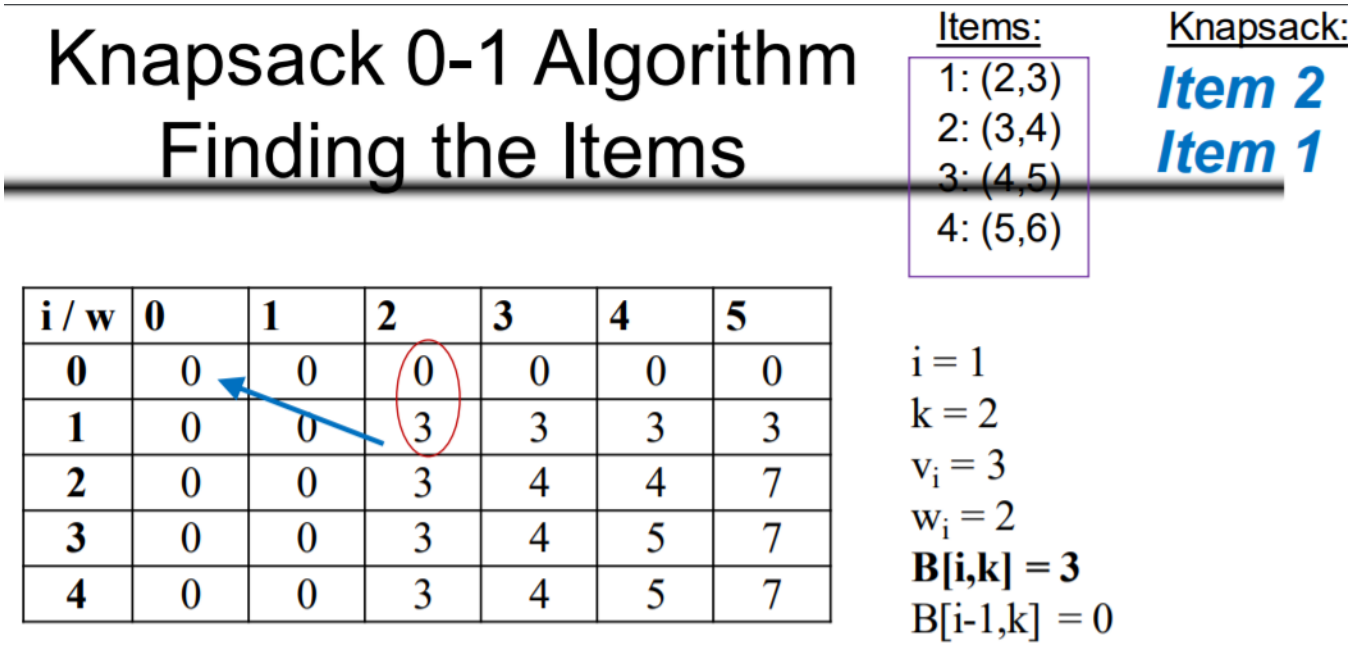
Implement 0-1 knapsack problem using dynamic programming.

Introduction

Given weights and values of n items, put these items in a knapsack of capacity W to get the maximum total value in the knapsack.

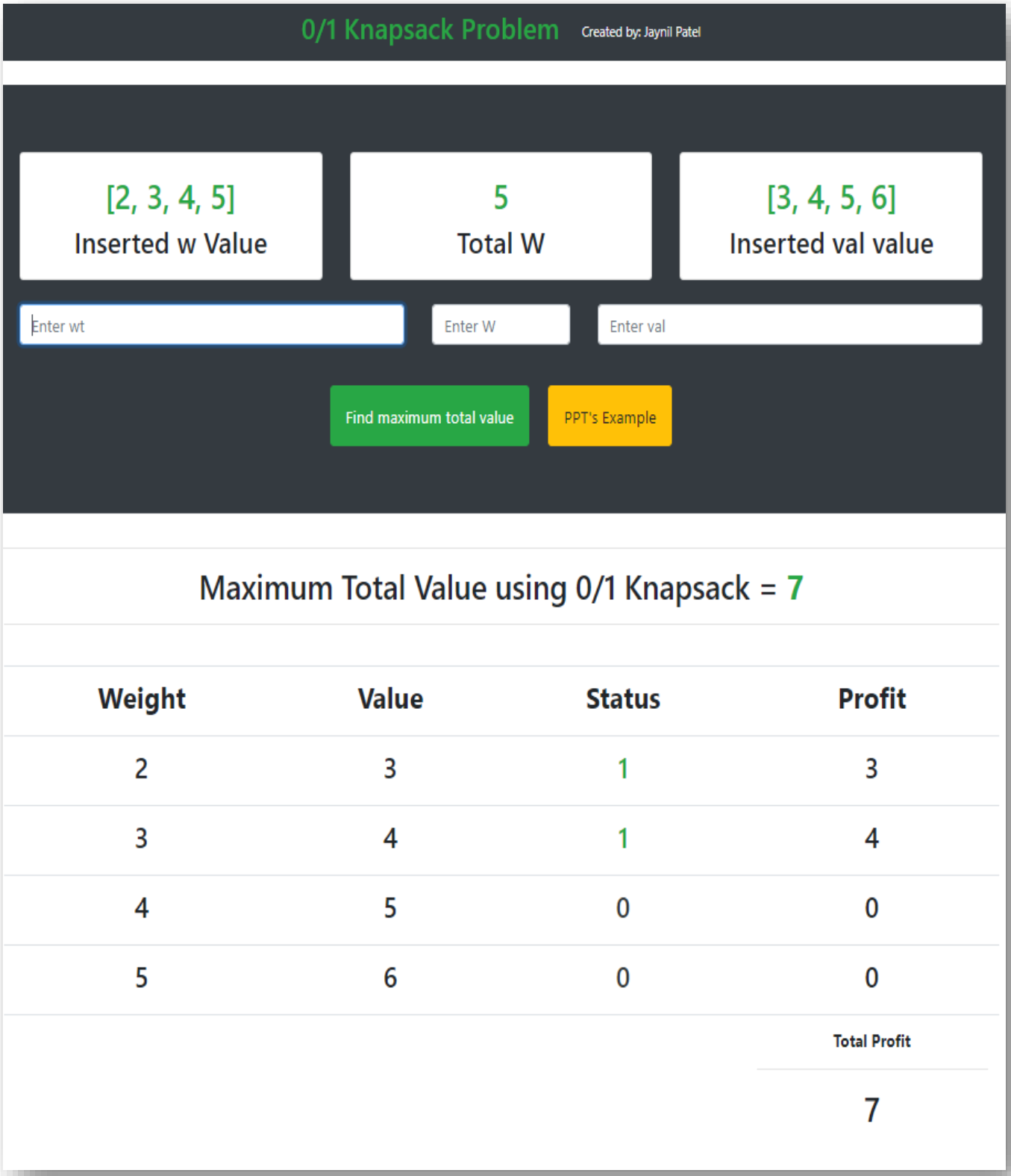
In other words, given two integer arrays val[0..n-1] and wt[0..n-1] which represent values and weights associated with n items respectively.

Also given an integer W which represents knapsack capacity, find out the maximum value subset of val[] such that sum of the weights of this subset is smaller than or equal to W.



i / w	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0	0	3	4	5	7
4	0	0	3	4	5	7

0/1 Knapsack(Simulation) using Python + Flask



0/1 Knapsack using Python

```
def ks01(W, wt, val, n):
    matrix = [[0 for x in range(W + 1)] for x in range(n + 1)]

    for i in range(n + 1):
        for w in range(W + 1):
            if i == 0 or w == 0:
                matrix[i][w] = 0
            elif wt[i-1] <= w:
                matrix[i][w] = max(val[i-1] + matrix[i-1][w-wt[i-1]], matrix[i-1][w])
            else:
                matrix[i][w] = matrix[i-1][w]
    res = matrix[n][W]
    profit = matrix[n][W]

    finalList = []
    w = W
    for i in range(n, 0, -1):
        if res <= 0:
            break
        if res == matrix[i - 1][w]:
            finalList.append([wt[i-1],val[i-1],0])
            continue
        else:
            finalList.append([wt[i-1],val[i-1],1])
#             print(wt[i - 1])
            res = res - val[i - 1]
            w = w - wt[i - 1]
    finalList.reverse()
    print("Max Profit = ",profit)
    print("\nWeight \t Value \t Status\n")
    for v in finalList:
        print(v[0],"\t",v[1],"\t",v[2])


#     print(finalList)
    return profit

# wt = [3,4,6,5]
# val = [2,3,1,4]
# W = 8

wt = [2,3,4,5]
val = [3,4,5,6]
W = 5

ans = ks01(W, wt, val, len(val))

# By Jaynil Patel (20MCED08)
```

Output :

```
Max Profit =  7

Weight     Value     Status

2             3          1
3             4          1
4             5          0
5             6          0
```

Observation:

Here we have used the dynamic approch to solve the knapsack Problem and we create a matrix (2d) for W columns and N = len(weight) rows so we are traversing all the W for all the Weights so the time complexity is O(N * W).