K. J. Somaiya College of Engineering, Mumbai-77

(Autonomous College Affiliated to University of Mumbai)

Batch: B1 **Roll No.: 1711072**

Experiment No. 6

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of the Staff In-charge with date

Title: Implementation of Knapsack Problem using Dynamic Programming

Objective To learn the Dynamic Programming using Knapsack Problem algorithm

CO to be achieved:

Sr. No	Objective
CO 1	Compare and demonstrate the efficiency of algorithms using asymptotic complexity notations.
CO 2	Analyze and solve problems for divide and conquer strategy, greedy method, dynamic programming approach and backtracking and branch & bound policies.
CO 3	Analyze and solve problems for different string matching algorithms.

Books/ Journals/ Websites referred:

1. **Ellis horowitz, Saratmarket Sahni, S.Rajsekaran," Fundamentals of computer algorithm", University Press**
2. **T.H.Cormen ,C.E.Leiserson,R.L.Rivest and C.Stein," Introduction to algortihtms",2nd Edition ,MIT press/McGraw Hill,2001**
3. **en.wikipedia.org/wiki/Knapsack_problem**
4. **www.es.ele.tue.nl/education/5MC10/Solutions/knapsack.pdf**
5. **cse.unl.edu/~ylu/raik283/notes/0-1-knapsack.ppt**
6. **www.es.ele.tue.nl/education/5MC10/Solutions/knapsack.pdf**
7. **cse.unl.edu/~ylu/raik283/notes/0-1-knapsack.ppt**

Pre Lab/ Prior Concepts:
Data structures, Concepts of algorithm analysis

Historical Profile:
Dynamic Programming (DP) is used heavily in optimization problems (finding the maximum and the minimum of something). Applications range from financial models and operation research to biology and basic algorithm research. So the good news is that understanding DP is profitable. However, the bad news is that DP is not an algorithm or a data structure that you can memorize. It is a powerful algorithmic design technique.

Principle of Optimality: It states that an optimal sequence of decisions has the property that whatever the initial state and decision are, the remaining decisions must constitute an optimal decision sequence with regard to the state resulting from the first decision.

The solution to the knapsack problem can be viewed as the result of a sequence of decisions. We have to decide the values of xi, 1<= i<+n. First we make a decision on x_i , then on x_2 . then on x_3 ,etc. An optimal sequence of decisions maximizes the objective function $\sum p_i x_i$.

New Concepts to be learned:
Application of algorithmic design strategy to any problem, dynamic Programming method of problem solving Vs other methods of problem solving, optimality of the solution,

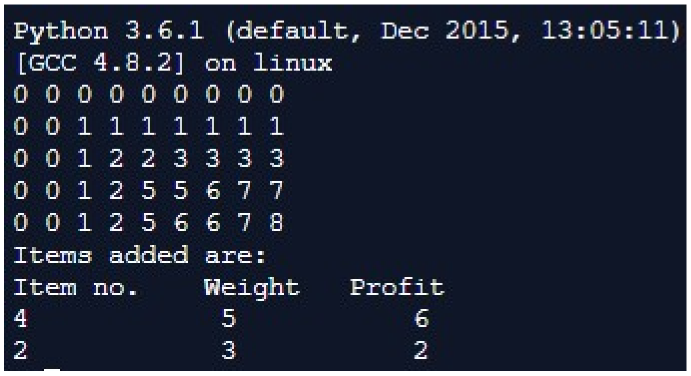
Algorithm

```
// Input:
// Values (stored in array v)
// Weights (stored in array w)
// Number of distinct items (n)
// Knapsack capacity (W)
for j from 0 to W do
  m[0, j] := 0
end for
for i from 1 to n do
  for j from 0 to W do
    if w[i] <= j then
      m[i, j] := max(m[i-1, j], m[i-1, j-w[i]] + v[i])
    else
      m[i, j] := m[i-1, j]
    end if
  end for
end for
```

Implementation:

```python
pw,cap,n=[[1,2],[2,3],[5,4],[6,5]],8,4
table=[[0 for x in range(cap+1)] for x in range(n+1)]
for i in range(1,n+1):
  for j in range(cap+1):
    if pw[i-1][1]<=j:
      table[i][j]=max(table[i-1][j], table[i-1][j-pw[i-1][1]]+pw[i-1][0])
    else:
      table[i][j]=table[i-1][j]
for i in table:
  print(*i, sep=' ')
i,j,sack=n,cap,[]
while i>0 and j>0:
  if table[i][j]!=table[i-1][j]:
    sack.append(i)
    i,j=i-1,j-pw[i-1][1]
  else:
    i=i-1
print('Items added are: \nItem no.\tWeight\t Profit')
for i in sack:
  print(i,'\t\t\t',pw[i-1][1],'\t\t\t',pw[i-1][0])
```

Output Screen:



```
Python 3.6.1 (default, Dec 2015, 13:05:11)
[GCC 4.8.2] on linux
0 0 0 0 0 0 0 0 0
0 0 1 1 1 1 1 1 1
0 0 1 2 2 3 3 3 3
0 0 1 2 5 5 6 7 7
0 0 1 2 5 6 6 7 8
Items added are:
Item no.     Weight   Profit
4             5            6
2             3            2
```

Analysis of 0/1 Knapsack algorithm:

The 0/1 Knapsack problem has two parts: one for finding the number of items that can be inserted and its profit, and second for finding which items are added to the knapsack. If the capacity of the knapsack is W and number of items considered is N, then the main algorithm consists of two for loops. The time complexity of the algorithm is given by:

$$T(n)=O(NW)$$

CONCLUSION: The 0/1 Knapsack problem was solved using dynamic programming technique and the program ran successfully for all test cases.