



**Jawahar Education Societys Annasaheb Chudaman Patil College of  
Engineering, Kharghar, Navi Mumbai**

**NAME: PRIYUSH BHIMRAO KHOBRADE**

**PRN NO: 211112018**

**Roll No: 52**

**SUBJECT: Analysis of Algorithms Lab**

EXPERIMENT: 06

PAGE NO.:

DATE: / / 20

Experiment No-06

• Aim :- Implement a C-program for 0/1 Knapsack using Dynamic programming.

• Objectives :- To implement and analyze complexity of 0/1 Knapsack using dynamic programming.

• Outcomes :- Students will be able to compute the complexity of 0/1 Knapsack using dynamic programming.

• Hardware/Software Required : 'C' Compiler

• Theory

Dynamic programming :

The dynamic programming is an algorithm design method that can be used when the solution to the program can be viewed as the result of a sequence of decisions. It avoids recomputing solutions that have already been computed. DP uses "principle of optimality."

The principle of optimality states that "in an optimal sequence of decisions or choices, each subsequence must also be optimal."

• Problem definition :-

There are  $n$  objects and a knapsack or bag of capacity  $M$ . Each object  $i$  has weight  $w_i$  and profit  $p_i$ . The objective is to obtain a filling of knapsack the maximum the total profit earned. The total weight of all chosen objects must be at most  $M$ .

Since this is a 0-1 knapsack problem so we can either take an entire item or reject it completely. We cannot break an item to fill the knapsack.

1

Teachers Signature \_\_\_\_\_

## 0/1 Knapsack using Dynamic Programming

PAGE NO.:

DATE.: / / 20

### • Procedure / Algorithm:

Algorithm Knapsack ( $P, w, n, M$ )

for  $j \leftarrow 0$  to  $M$

$V(0, j) \leftarrow 0$

for  $i \leftarrow 0$  to  $n$

$V(i, 0) \leftarrow 0$

for  $i \leftarrow 1$  to  $n$

for  $j \leftarrow 1$  to  $M$

if ( $w(i) > j$ )

$V(i, j) = V(i-1, j)$

Else

$V(i, j) = \max \{ V(i-1, j), P(i) + V(i-1, j-w(i)) \}$

$i \leftarrow n$ ;

$j \leftarrow M$ ;

while ( $i > 0$ ) do

if ( $V(i, j) \neq V(i-1, j)$ )

$x(i) \leftarrow 1$ ;

$j \leftarrow j - w(i)$ ;

$i \leftarrow i - 1$

print  $V$  and  $x$

### • Analysis

$$\begin{aligned} T(n) &= \sum_{i=1}^n \sum_{j=1}^M 1 \\ &= mn = O(mn) \end{aligned}$$

• Conclusion :- Thus, it is observed that the complexity of 0/1 Knapsack problem is  $O(Mn)$ .

2

Teachers Signature \_\_\_\_\_

## 0/1 Knapsack using Dynamic Programming

### Input:

```
1 #include<stdio.h>
2 void kp(int n,int pro[],int wt[],int m);
3 int max(int a,int b);
4 void main()
5 {
6     int i,j,n;
7     int wt[20],pro[20];
8     int m;
9     printf("Enter no. of weights:");
10    scanf("%d",&n);
11    printf("Enter the capacity:");
12    scanf("%d",&m);
13    printf("Enter the weights: \n");
14    for(i=1;i<=n;i++)
15    {
16        printf("\nWT %d:",i);
17        scanf("%d",&wt[i]);
18    }
19    printf("Enter the profit\n");
20    for(i=1;i<=n;i++)
21    {
22        printf("\npro %d:",i);
23        scanf("%d",&pro[i]);
24    }
25    kp(n,pro,wt,m);
26 }
27
28 int max(int a,int b)
29 {
30     return((a>b)?a:b);
31 }
32
33 void kp(int n,int pro[],int wt[],int m)
34 {
35     int x[20],v[20][20],i,j;
36     for(i=1;i<=n;i++)
37     {
38         x[i]=0;
39     }
40     for(i=0;i<=n;i++)
41     {
42         v[i][0]=0;
43     }
44     for(i=0;i<=m;i++)
45     {
46         v[0][i]=0;
47     }
48     for(i=1;i<=n;i++)
49     {
50         for(j=1;j<=m;j++)
51         {
52             if(wt[i]>j)
53             {
54                 v[i][j]=v[i-1][j];
55             }
56             else
57             {
58                 v[i][j]=max(v[i-1][j],pro[i]+v[i-1][j]-wt[i]);
59             }
60         }
61     }
62     printf("Output:\n");
63     for(i=0;i<=n;i++)
64     {
65         for(j=0;j<=m;j++)
66         {
67             printf("%d ",v[i][j]);
68         }
69         printf("\n");
70     }
71
72     printf("THE PROFIT IS: %d",v[n][m]);
73     printf("\n");
74     i=n;
75     j=m;
76     while(i!=0)
77     {
78         if(v[i][j]!=v[i-1][j])
79         {
80             x[i]=1;
81             j=j-wt[i];
82         }
83         i=i-1;
84     }
85     for(i=1;i<=n;i++)
86     printf("%d ",x[i]);
87 }
```

## 0/1 Knapsack using Dynamic Programming

### Output:

```
C:\Users\Rupesh\Documents\OS\AOAEX06\bin\Debug\AOAEX06.exe
Enter no. of weights:5
Enter the capacity:9
Enter the weights:

WT 1:2
WT 2:3
WT 3:4
WT 4:5
WT 5:6
Enter the profit

pro 1:1
pro 2:2
pro 3:5
pro 4:6
pro 5:4
Output:
0 0 0 0 0 0 0 0 0 0
0 0 1 1 1 1 1 1 1 1
0 0 1 2 2 3 3 3 3 3
0 0 1 2 5 5 6 7 7 8
0 0 1 2 5 6 6 7 8 11
0 0 1 2 5 6 6 7 8 11
THE PROFIT IS:11
0 0 1 1 0
Process returned 6 (0x6)   execution time : 44.837 s
Press any key to continue.
```

**Conclusion:** Thus it is Observed that the complexity of **0/1 Knapsack Problem** is  **$O(Mn)$** .