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

NAME: PRIYUSH BHIMRAO KHOBRAGADE

PRN NO: 211112018

Roll No: 52

SUBJECT: Analysis of Algorithms Lab

0/1 Knapsack using Dynamic Programming

EXPERMINT: 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 the 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 & 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 i ← 0 to M
        V(0,i) ← 0
    for i ← 0 to n
        V(i,0) ← 0
    for i ← 1 to n
    for j ← 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 ← n;
    j ← M;
    while (i != 0) do
        if (V(i,j) != V(i-1, j))
            x(i) ← 1;
            j ← j - w[i];
            i ← i - 1
    print V and X
```

• Analysis

$$T(n) = \sum_{i=1}^{n} 1 \cdot \sum_{j=1}^{M} 1 \cdot$$

$$= mn = O(mn)$$

• 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:

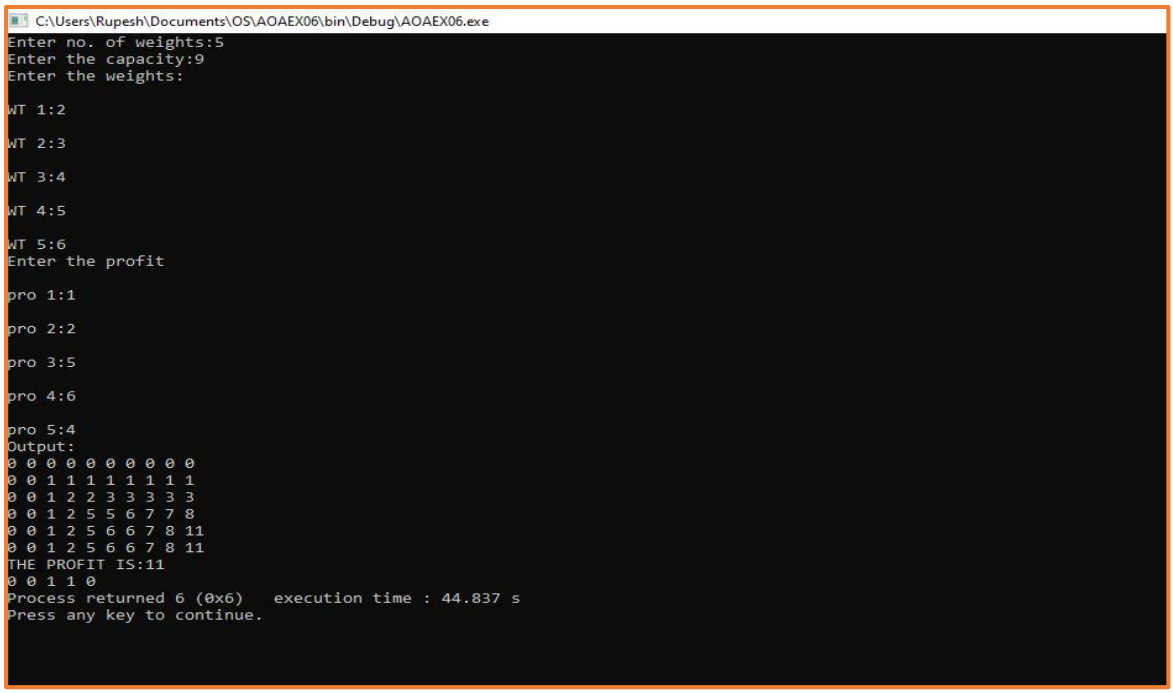
```
#include<stdio.h>
void kp(int n,int pro[],int wt[],int m);
int max(int a,int b);
void main()
{
int i,j,n;
int wt[20],pro[20];
int m;
printf("Enter no. of weights:");
scanf("%d",&n);
printf("Enter the capacity:");
scanf("%d",&m);
printf("Enter the weights: \n");
        for(i=1;i<=n;i++)
        {
        printf("\nWT %d:",i);
        scanf("%d",&wt[i]);
        }
printf("Enter the profit\n");
        for(i=1;i<=n;i++)
        {
        printf("\npro %d:",i);
        scanf("%d",&pro[i]);
        }
kp(n,pro,wt,m);
}

int max(int a,int b)
{
return((a>b)?a:b);
}

void kp(int n,int pro[],int wt[],int m)
        {
int x[20],v[20][20],i,j;
for(i=1;i<=n;i++)
        {
        x[i]=0;
        }
        for(i=0;i<=n;i++)
        {
        v[i][0]=0;
        }
        for(i=0;i<=m;i++)
        {
        v[0][i]=0;
        }
        for(i=1;i<=n;i++)
        {
        for(j=1;j<=m;j++)
        {
        if(wt[i]>j)
                {
                v[i][j]=v[i-1][j];
                }
        else
        {
                v[i][j]=max(v[i-1][j],pro[i]+v[i-1][j-wt[i]]);
        }
        }
        }
printf("Output:\n");
        for(i=0;i<=n;i++)
        {
        for(j=0;j<=m;j++)
                {
                printf("%d ",v[i][j]);
        }
        printf("\n");
}

printf("THE PROFIT IS:%d",v[n][m]);
printf("\n");
i=n;
j=m;
        while(i!=0)
        {
        if(v[i][j]!=v[i-1][j])
        {
                x[i]=1;
                j=j-wt[i];
        }
i=i-1;
}
for(i=1;i<=n;i++)
printf("%d ",x[i]);
}
```

Output:



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