

EXPERIMENT NO- 5

AIM: Implementation of knapsack problem using greedy method.

PROBLEM STATEMENT :

Write a program to implement Knapsack Problem using greedy method.

Resource Required: Pentium IV, Turbo C, Printer, Printout Stationary

THEORY:**Problem Definition :**

We are given an empty knapsack of capacity "W" and we are given "n" different objects from $i=1,2,\dots,n$. Each object "i" has some positive weight " w_i " and some profit value is associated with each object which is denoted as " p_i ".

We want to fill the knapsack with total capacity „W" such that profit earned is maximum. When we solve this problem, main goal is:

1. Choose only those objects that give maximum profit.
2. The total weight of selected objects should be $\leq W$

Most problem have n input and require us to obtain a subset that satisfies some constraints is called as a feasible solution. We are required to find a feasible solution that optimize (minimize or maximizes) a given objective function. The feasible solution that does this is called an optimal solution.

Example:

$$n = 3$$

$$\text{profit} \rightarrow P_i = (25, 24, 15)$$

$$\text{Weight (in kg)} \rightarrow W_i = (18, 15, 10)$$

where

$$i = 1, 2, 3 \text{ and knapsack capacity} = 20$$

Feasible solutions:

1. As knapsack capacity is 20, we first put W_1 (i.e. 18 kg) into knapsack, hence gained profit is 25; but now we can put only 2 kg, in knapsack, so we will put 2 kgs from W_2 , hence profit from this 2kg of w_2 is 3.2.

Thus, total profit is $25 + 3.2 = 28.2$

2. Now, we will put W3 (i.e. 10 KG) into knapsack, hence gained profit is 15, but we can put only 10 kg in knapsack, so we will put 10 kg from w2, hence profit from this 10 kg of w2 is 16. Thus, total profit is $15 + 16 = 31$
3. Now, we will put W2 (i.e. 15 KG) into knapsack, hence gained profit is 24, but we can put only 5 kg in knapsack, so we will put 5 kg from w3, hence profit from this 5 kg of w3 is 7.5. Thus, total profit is $24 + 7.5 = 31.5$

Hence, optimal solution for given problem is 3rd solution, which is giving total profit as 31.5.

The knapsack problems are often solved using dynamic programming, though no polynomial time algorithm is known. Both the knapsack problem(s) and the subset Sum problem are NP-hard.

Algorithm:

- a Let „W” be the maximum weight of the knapsack
- b Let „ w_i ” and „ p_i ” be the weight and profit of individual items i.e. for $i=1, \dots, n$
- c Calculate p_i / w_i ratio and arrange that in decreasing order.
- d initially weight=0 and profit = 0
- e for $i=1$ to n
{
 add item in knapsack till weight $\leq W$
 profit = profit + p_i
}
f Stop

CONCLUSION: Most problems have n inputs and require us to obtain a subset that satisfies some constraints is called as feasible solution. In knapsack problem, greedy strategy, we find out feasible solution that optimize (minimizes or maximizes) a given objective function. A, feasible solution that does this is called an optimal solution. Hence, in knapsack problem we found out an optimal solution.

The knapsack algorithm takes $\theta(nw)$ times, broken up as follows: $\theta(nw)$ times to fill the c -table, which has $(n+1) \cdot (w+1)$ entries, each requiring $\theta(1)$ time to compute. $O(n)$ time to trace the solution, because the tracing process starts in row n of the table and moves up 1 row at each step.

Code:

```
#include<stdio.h>
#include <Windows.h>
#define MAX(x,y) x>y?x:y

enum{true = 1, false = 0}bool;

int knapsack(int items[],int weight[],int W,int profit,int n);

int fractknapsack(int items[],int weight[],int W,int profit,int n);

int main()
{
    int n;
    printf("Enter number of items you have\n");
    scanf("%d",&n);
    int items[n], weight[n],cases,profit,W;
    printf("Enter the weight of KnapSack \n");
    scanf("%d",&W);
    printf("Enter the items profit and their corresponding weight \n");
    for(int i = 0; i<n; i++)
    {
        scanf("%d %d",&items[i],&weight[i]);
    }

    while(true)
    {
        printf("1.0/1 KnapSack \n2.Fractional Knapsack \n3.Exit\n");
        scanf("%d",&cases);
        switch (cases)
        {
            case 1:
                knapsack(items,weight,W,profit,n);

                break;
```

```
case 2:
    fractknapsack(items,weight,W,profit,n);
    break;
case 3:
    printf("Aborting in 2 seconds");
    Sleep(2000);
    return 0;

default:
    printf("You entered wrong key please try again\n");
    break;
}
}
return 0;
}

int fractknapsack(int items[],int weight[],int W,int profit,int n)
{
    float dense[n],max_profit = 0,X[n];
    int L=W;
    for(int i=0;i<n;i++)
    {
        dense[i] = items[i]/weight[i];
        X[i] = 0;
    }
    for(int i=1;i<n;i++)
    {

        int mind = dense[i],mini = items[i],minw = weight[i];
        int j = i-1;
        while(j>=0&&dense[j]>mini)
        {
            dense[j+1]=dense[j];
            weight[j+1] = weight[j];
            items[j+1] = items[j];
            j--;
        }
    }
}
```

```
    weight[j+1] = minw;
    dense[j+1] = mind;
    items[j+1] = mini;
}
for(int i=0;i<n;i++)
{
    if(L-weight[i]>=0)
    {
        max_profit += items[i];
        L -=weight[i];
        X[i] = 1;
    }
    else
    {
        X[i] = dense[i]*L;
        max_profit += (dense[i]*L);
        break;
    }
}
printf("Max Profit= %f \n",max_profit);
printf("Items: \n");
for(int i=0;i<n;i++)
{
    if(X[i]==1)
        printf("Profit: %d \t Weight: %d \n",items[i],weight[i]);
    else if(X[i]>0)
        printf("Profit: %f \t Weight: %d \n",X[i],L);
    else
        break;
}
return 0;
}

int knapsack(int items[],int weight[],int W,int profit,int n)
{
    int v[n+1][W+1],X[n],k,max_profit = 0;
    for(int i = 0;i<n;i++)
```

```
{
    X[i] = 0;
}

for(int i = 0;i<=n;i++)
{
    v[i][0] = 0;
}

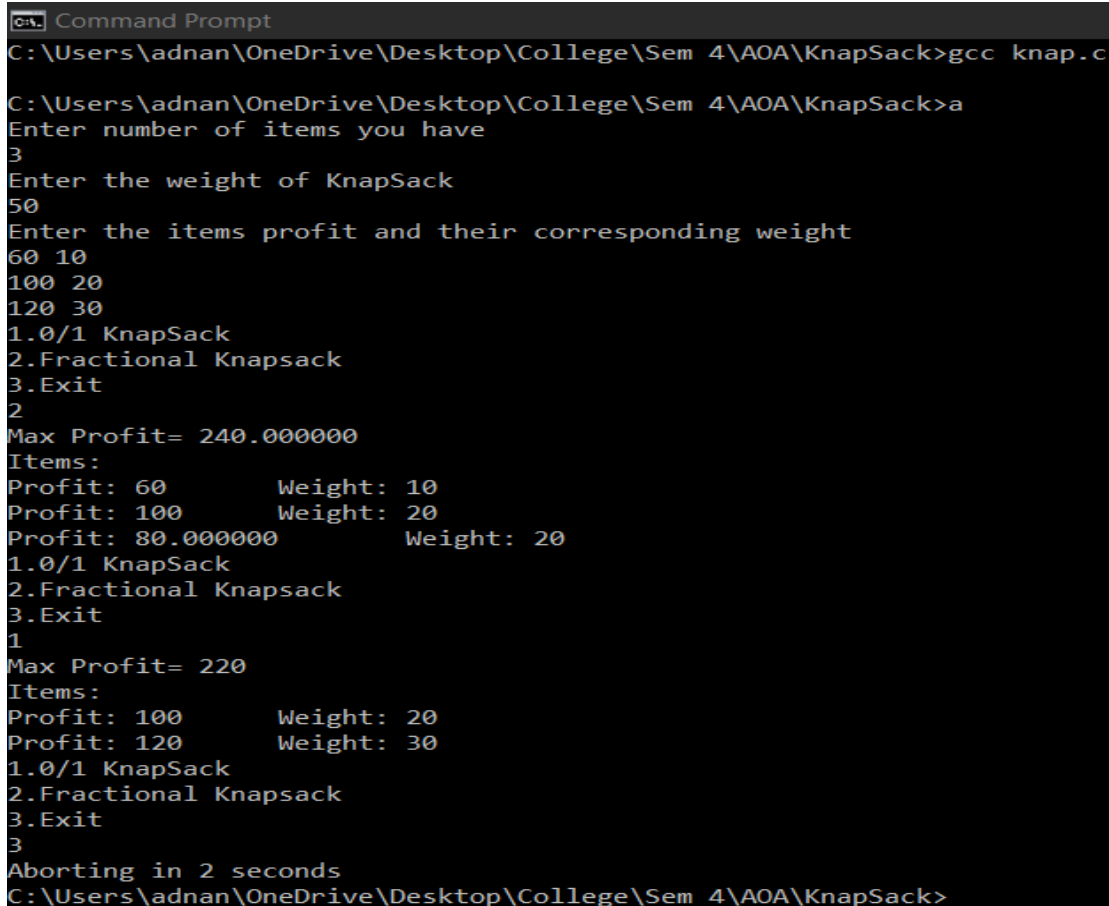
for(int i = 0;i<=W;i++)
{
    v[0][i] = 0;
}
for(int i=1;i<=n;i++)
{
    for(int j=1;j<=W;j++)
    {
        k = j-weight[i-1];
        if(k<0)
            v[i][j] = v[i-1][j];
        else{
            v[i][j] = MAX(v[i-1][j],(items[i-1]+v[i-1][k]));
        }
    }
}
max_profit = v[n][W];
int remain_profit = max_profit,L=n,M=W;
while(remain_profit>0 || L>0 || M>0)
{
    if(v[L][M]>v[L-1][M])
    {
        if(remain_profit >=v[L][M])
        {
            X[L-1]++;
            remain_profit -= items[L-1];
            L--;
        }
    }
}
```

```
        else
            M --;
    }
    else
    {
        L--;
    }

}

max_profit = v[n][W];
printf("Max Profit= %d \n",max_profit);
printf("Items: \n");
for(int i=0;i<n;i++)
{
    if(X[i]==1)
        printf("Profit: %d \t Weight: %d \n",items[i],weight[i]);
}
}
```

Output:



```
Command Prompt
C:\Users\adnan\OneDrive\Desktop\College\Sem 4\AOA\KnapSack>gcc knap.c
C:\Users\adnan\OneDrive\Desktop\College\Sem 4\AOA\KnapSack>a
Enter number of items you have
3
Enter the weight of KnapSack
50
Enter the items profit and their corresponding weight
60 10
100 20
120 30
1.0/1 KnapSack
2.Fractional Knapsack
3.Exit
2
Max Profit= 240.000000
Items:
Profit: 60      Weight: 10
Profit: 100     Weight: 20
Profit: 80.000000      Weight: 20
1.0/1 KnapSack
2.Fractional Knapsack
3.Exit
1
Max Profit= 220
Items:
Profit: 100     Weight: 20
Profit: 120     Weight: 30
1.0/1 KnapSack
2.Fractional Knapsack
3.Exit
3
Aborting in 2 seconds
C:\Users\adnan\OneDrive\Desktop\College\Sem 4\AOA\KnapSack>
```