EXPERIMENT 5

Experiment No: 5 **Date:** 10/04/2021

Aim: Implementation of Knapsack Problem (Greedy Approach) and obtain its step count

Theory:

GREEDY METHOD

- A greedy algorithm, as the name suggests, always makes the choice that seems to be the best at that moment.
- This means that it makes a locally-optimal choice in the hope that this choice will lead to a globally-optimal solution.

ADVANTAGES AND DISADVANTAGES OF GREEDY METHOD

- It is quite easy to come up with a greedy algorithm (or even multiple greedy algorithms) for a problem.
- Analysing the run time for greedy algorithms will generally be much easier than for other techniques (like Divide and conquer).
- For the Divide and conquer technique, it is not clear whether the technique is fast or slow.
- This is because at each level of recursion the size of gets smaller and the number of sub-problems increases.
- The difficult part is that for greedy algorithms you have to work much harder to understand correctness issues.

- Even with the correct algorithm, it is hard to prove why it is correct.
- Proving that a greedy algorithm is correct is more of an art than a science.

GREEDY METHOD SUGGESTS

- Algorithm can be devised in stages, considering one input at each stage
- At each stage a decision is made to include an input or not (If the input leads to OS then input is considered or else its ignored)
- This is done by considering the input in a particular order defined by selection procedure (as per objective function of the P)

KNAPSACK ALGORITHM

- We are given n objects and a Knapsack (or bag) of capacity m.
- Associated with each object i there is a weight wi and profit pi.
- The objective function is to fill up the bag so as to maximize the profit earned subject to the constraint is that the sum of the weights of the chosen objects put into the bag should not exceed the capacity of the bag.
- If a fraction xi (0 <= xi <= 1) of an object i is chosen to include into the bag then a profit of pixi is earned.

EXPERIMENT 5

- For this problem, we need to:
  - Maximize: $\Sigma 1 < i < n \; p_i x_i$
  - Subject to the condition: $\Sigma 1 < i < n \; w_i x_i < m$
  - And $0 < x_i < 1$ and $1 < i < n$
- The profits and weights are positive numbers. A feasible solution (or filling) is any set $(x_i, \ldots, x_n)$ satisfying 2 and 3 above.
- An optimal solution is a feasible solution for which 1 is maximized.
- In case the sum of all weights is $\le m$, then $x_i = 1$, $1 \le i \le n$ is an optimal solution.
- All optimal solutions will fill the knapsack exactly.
- In Knapsack, optimal solution is obtained when objects are selected in the decreasing order of p_i/w_i.

EXPERIMENT 5

ALGORITHM

```
Algorithm GreedyKnapsack (m, n)

//p[1:n] and w[1:n] contain the profits and weights respectively

// of the n objects ordered such that p[i]/w[i] >= p[i+1]/[wi+1]

//m is the knapsack size and x[1 : n] is the solution vector. for i := 1

to n do x[i]:= 0.0;

{
        for i:=1 to n do x[i] := 0.0;   //Initialize

        U := m;

        for i:=1 to n do

        {
                if(w[i] > U) then break;

                x[i] := 1.0; U = U - w[i];
        }

        if(i<=n) then x[i] := U/w[i];
}
```

ALGORITHM WRITING

- We accept 2 arrays: profit array p[1:n] and weight array w[1:n].
- We find the ratio of p[i]/w[i] for every element (where 1< i <n) and re-arrange in descending order according to value of p[i]/w[i] and compute solution vector and profit

EXPERIMENT 5

TRACING OF PROBLEM

Let n=5, m=12

(p1, p2, p3, p4, p5) = (10, 15, 8, 6, 7) and (w1, w2, w3, w4, w5) = (4, 6, 3, 4, 2)

Solution: Feasible solutions can be obtained by using different strategies

1. ***Random choosing:*** Elements are chosen at random so that we fill the sack

   X1=1 m=12 > 4(w1) m=12-4 = 8

   X2=1 m=8 > 6(w2) m=8-6 = 2

   X5=1 m=2 = 2(w5) m=2-2 = 0 ……sack full

   X3=0, X4=0

   Therefore: (x1, x2, x3, x4, x5) = (1, 1, 0, 0, 1)

   Therefore: Σ wixi = 12 and Σ pixi = 32

EXPERIMENT 5

2. ***Max Profit Strategy:*** Elements are chosen according to which one offers maximum profit

Profit: p2 > p1 > p3 > p5 > p4

X2=1	m=12 > 6(w2)	m=12-6 = 6
X1=1	m=6 > 4(w1)	m=6-4 = 2
X3=2/3	m=2 < 3(w3)	.....sack full

X5=0, X4=0

Therefore: (x1, x2, x3, x4, x5) = (1, 1, 2/3, 0, 0)

Therefore: Σ wixi = 12 and Σ pixi = 30.33

3. **Least weight strategy:** Elements are chosen according to their weight(lightest is given first preference)

Weights: w5 < w3 < w1 < w4 < w2

X5=1	m=12 > 2(w5)	m=12-2 = 10
X3=1	m=10 > 3(w3)	m=10-3 = 7
X1=1	m=7 > 4(w1)	m=7-4 = 3
X4=3/4	m=3 < 4(w4)	.....sack full

X5=0

Therefore: (x1, x2, x3, x4, x5) = (1, 0, 1, 3/4, 1)

Therefore: Σ wixi = 12 and Σ pixi = 29.5

EXPERIMENT 5

4. ***<u>Decreasing order of pi/wi Strategy:</u>*** Elements are arranged in descending order of pi/wi

(p1/w1) = 10/4 = 2.5 (p2/w2) = 15/6 = 2.5

(p3/w3) = 8/3 = 2.67

(p4/w4) = 6/4 = 1.5 (p5/w5) = 7/2 = 3.5

Hence, (p5/w5) > (p3/w3) > (p2/w2) > (p1/w1) > (p4/w4)

X5=1	m=12 > 2(w5)	m=12-2 = 10
X3=1	m=10 > 3(w3)	m=10-3 = 7
X2=1	m=7 > 6(w2)	m=7-6 = 1
X1=1/4	m=1 < 4(w1)	.....sack full
X4=0		

Therefore: (x1, x2, x3, x4, x5) = (1/4, 1, 1, 0, 1)

Therefore: Σ wixi = 12 and Σ pixi = 32.5

(x1, x2, x3, x4, x5)	$\Sigma w_i x_i$	$\Sigma p_i x_i$	Strategy Used
(1, 1, 0, 0, 1)	12	32	Randomly chosen
(1, 1, 2/3, 0, 0)	12	30.33	Max Profit
(1, 0, 1, ¾, 1)	12	29.5	Least Weight
(1/4, 1, 1, 0, 1)	12	***32.5***	Descending order of p_i/w_i

The optimal solution(OS) is the strategy that gives maximum value of p_i/w_i

Therefore, The OS is selecting elements in descending order of p_i/w_i that gives a profit of ***<u>32.5</u>***

EXPERIMENT 5

CODE

```
#include<iostream>

using namespace std;

int ctr; float static pr[20], wt[20], x[20];

void GreedyKnapsack(int m, int n)

{

        ctr++;

        int u=m;       ctr++;

        float sum=0.0;       ctr++;

        for(int i=0; i<n; i++)

        {

                x[i]=0.0; ctr++;

        }

        ctr++;

        int i;

        for( i=0; i<n; i++)

        {

                ctr++;

                ctr++;

                if(wt[i] > u)
```

EXPERIMENT 5

```
        break;

        x[i]=1.0;       ctr++;

        u = u-wt[i];   ctr++;

        sum= sum+(pr[i]*x[i]);     ctr++;

}

ctr++;

ctr++;

if(i<=n)

{

        x[i] = float(u/wt[i]);          ctr++;

        sum= sum+(pr[i]*x[i]);     ctr++;

}

cout<<"\n\n********************"<<endl;

cout<<"SOLUTION VECTOR: ";   ctr++;


for(int i=0; i<n; i++)

{

        ctr++;

        cout<<x[i]<<"  ";ctr++;

}
```

EXPERIMENT 5

```
        cout<<"\n*********************"<<endl;

        ctr++;

        cout<<"\n\n*********************"<<endl;

        cout<<"MAXIMUM PROFIT: "<<sum<<endl;  ctr++;

        cout<<"*********************"<<endl;

        ctr++;

}

int main()

{

        int n, m;

        cout<<"\nENTER NUMBER OF ELEMENTS(n): \n";   ctr++;

        cin>>n;         ctr++;

        cout<<"\nENTER CAPACITY(m): \n";    ctr++;

        cin>>m;         ctr++;

        cout<<"\nENTER PROFITS: "<<"\n";     ctr++;

        for(int i=0; i<n; i++)

        {

                cin>>pr[i];    ctr++;

        }

        ctr++;
```

EXPERIMENT 5

```
cout<<"\nENTER WEIGHTS: "<<"\n";   ctr++;
for(int i=0; i<n; i++)
{
        cin>>wt[i];    ctr++;
}
ctr++;
float ratio[n], temp;          ctr++;
for (int i = 0; i < n; i++)
{
        ctr++;
ratio[i] = pr[i] / wt[i];      ctr++;
}
ctr++;
for (int i = 0; i < n; i++)
{
        ctr++;
        for (int j = 0; j < n; j++)
                {
                        ctr++;
                        ctr++;
```

EXPERIMENT 5

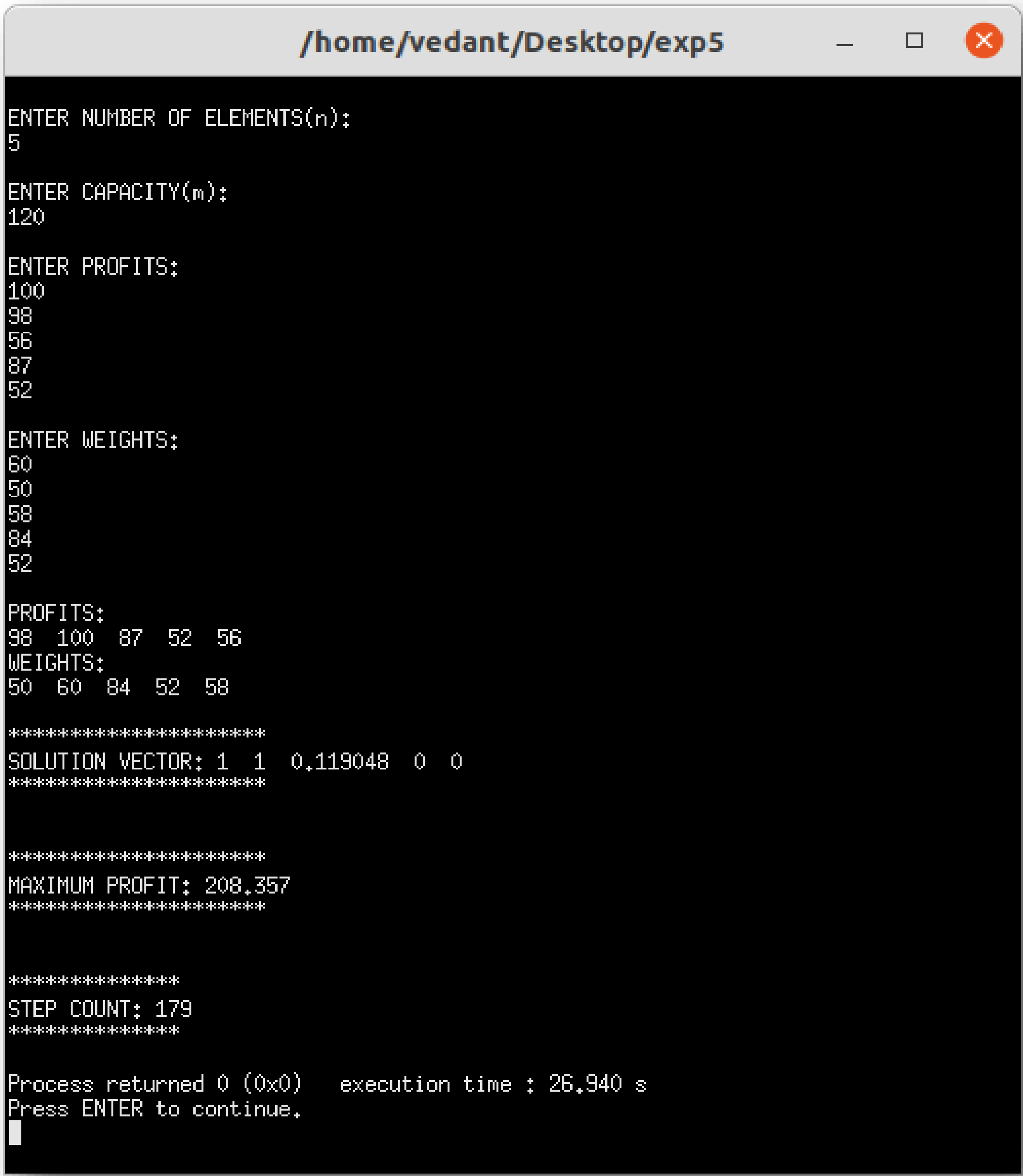
```
                if (ratio[j] < ratio[j+1])
                        {
                    temp = ratio[j];      ctr++;
                    ratio[j] = ratio[j+1];          ctr++;
                    ratio[j+1] = temp;   ctr++;

                    temp = wt[j];         ctr++;
                    wt[j] = wt[j+1];      ctr++;
                    wt[j+1] = temp;       ctr++;

                    temp = pr[j];ctr++;
                    pr[j] = pr[j+1];      ctr++;
                    pr[j+1] = temp;       ctr++;
                }
        }
        ctr++;
}
ctr++;
cout<<"\nPROFITS: "<<endl;      ctr++;
for(int i=0; i<n; i++)
```

EXPERIMENT 5

```
        {
                ctr++;
                cout<<pr[i]<<"  ";ctr++;
        }
        cout<<"\nWEIGHTS: "<<endl;    ctr++;
        for(int i=0; i<n; i++)
        {
                ctr++;
                cout<<wt[i]<<"  ";ctr++;
        }
        GreedyKnapsack(m, n);
        cout<<"\n\n**************"<<endl;
        cout<<"STEP COUNT: "<<ctr<<endl;
        cout<<"**************"<<endl;
        return 0;
}
```

EXPERIMENT 5

OUTPUT



```
/home/vedant/Desktop/exp5

ENTER NUMBER OF ELEMENTS(n):
5

ENTER CAPACITY(m):
120

ENTER PROFITS:
100
98
56
87
52

ENTER WEIGHTS:
60
50
58
84
52

PROFITS:
98  100  87  52  56
WEIGHTS:
50  60  84  52  58

*********************
SOLUTION VECTOR: 1  1  0.119048  0  0
*********************


*********************
MAXIMUM PROFIT: 208.357
*********************


**************
STEP COUNT: 179
**************

Process returned 0 (0x0)   execution time : 26.940 s
Press ENTER to continue.
```

CONCLUSION

- Detailed concept of Knapsack Problem (Greedy Method) was studied successfully.
- Knapsack program was executed successfully.
- The step count for the Quick Sort algorithm was obtained