**Practical 4**

**Aim: -** Greedy Approach

4.1 Making Change Problem (Implement)

4.2 0/1 and Fractional Knapsack Problem (Study/Implement)

4.3 Activity Selection Problem (Study/Implement)

**4.1 Making Change Problem: -**

**Theory:-**

The problem is of making a change for given amount using the smallest possible number of coins. Using the greedy choice property of this algorithm, optimum solution is obtained.

Example: Coins: {5, 10, 15, 20, 25, 50, 100}

        Amount: 500

       Solution: 100 x 5 = 500

**Algorithm: -**

MAKE-CHANGE (n)

step1. C ← Set of Available Coins

step2. Sol ← {};

step3. Sum ← 0

step4. WHILE sum not = n

(x = largest item in set C)

sum + x ≤ n

IF no such item THEN

RETURN "No Solution"

S ← S {value of x}

sum ← sum

**Program: -**

**Code: -**

#include<iostream>

using namespace std;

int i;

main()

{

int n,total=0,d,coins[7]={100,50,25,20,15,10,5},count[7]={0};

int i;

cout<<"Enter the amount"<<endl;

cin>>n;

while(n>0 && n<=1125)

{

while(n>coins[i])

{

n=n-coins[i];

count[i]++;

if(count[i]>=5)

if(n<5)

{

break;

}

i++;

}

if(n==0)

{

cout<<endl;

for(int i=0;i<7;i++){

total+=count[i];

cout<<"/n Rupee coins:"<<coins[i]<<count[i];

}

cout<<"\n\n Total number of coins"<<total;

}

else{

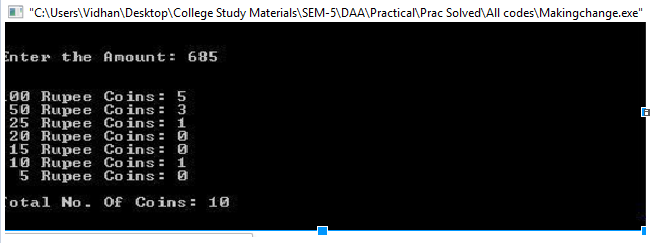
cout<<"Solution not possible"<<endl;

}

return 0;

}

**Output: -**



**Conclusion: -**

Making change is a greedy algorithm, with the time complexity of n (log (n)).

**4.2 0/1 and Fractional Knapsack Problem (Study/Implement)**

**Theory: -**

The Greedy algorithm could be understood very well with a well-known problem referred to as Knapsack problem

Given a set of items, each with a weight and a value, determine a subset of items to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible. The knapsack problem is in combinatorial optimization problem. Here the item can be taken in the fractions if needed to be.

Example:

N=7, W=15

|  |  |  |  |
| --- | --- | --- | --- |
| Sr No | Pi | Wi | Pi / Wi |
| 1 | 10 | 2 | 5 |
| 2 | 5 | 3 | 1.66 |
| 3 | 15 | 5 | 3 |
| 4 | 7 | 7 | 1 |
| 5 | 6 | 1 | 6 |
| 6 | 18 | 4 | 4.5 |
| 7 | 3 | 1 | 3 |

P5 / W5 -> P1 / W1 -> P6 / W6 -> P7 / W7 -> P3 / W3 -> P2 / W2 -> P4 / W4

|  |  |  |
| --- | --- | --- |
| Item No | Weight | Profit |
| 5 | 15-1=14 | 0+6=6 |
| 1 | 14-2=12 | 6+10=16 |
| 6 | 12-4=8 | 16+18=34 |
| 7 | 8-1=7 | 34+3=37 |
| 3 | 7-5=2 | 37+15=52 |
| 2 | 2-2=0 | 52 + 5\*(2/3)=55.32 |

Solution: (1, 2/3, 1, 0, 1, 1, 1)

**Algorithm: -**

step 1: Sort the n objects from large to small based on the ratios Pi/Wi.

step 2: x[1..n]=0, weight=0, i=1

step 3: while (i<=n and weight < W)

            if weight + w[i] <= W then x[i]=1

else x[i] = (W-weight) / w[i]

weight = weight + x[i] \* w[i]

**Program: -**

**Code: -**

#include<iostream>

using namespace std;

int main()

{

int W,n;

cout<<"Enter the total weight:"<<endl;

cin>>W;

cout<<"Enter the no. of objects:"<<endl;

cin>>n;

float w[n],p[n];

float r[n];

float z[n];

int q[n];

cout<<"Enter the individual weight:"<<endl;

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

{

cin>>w[i];

q[i]=i+1;

}

cout<<"Enter the profit values:"<<endl;

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

{

cin>>p[i];

z[i]=0;

}

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

{

r[i]=p[i]/w[i];

}

for(int m=0;m<n-1;m++)

{

float t1,t2,t3;

float t;

float f=r[m];

for(int l=m+1;l<n;l++)

{

if(r[l]>f)

{

t=f;

f=r[l];

r[l]=t;

t1=w[m];

w[m]=w[l];

w[l]=t1;

t2=p[m];

p[m]=p[l];

p[l]=t2;

t3=q[m];

q[m]=q[l];

q[l]=t3;

}

}

}

float x=0,remain=W;

int i=0;

while(remain!=0 && n>=i)

{

if(remain>=w[i]){

z[q[i]]=1;

remain=remain-w[i];

x+=p[i];

}

else{

z[q[i]]=remain/w[i];

x+=(p[i]\*(remain/w[i]));

break;

}

i++;

}

for(int i=1;i<=n;i++)

{

cout<<" "<<z[i];

}

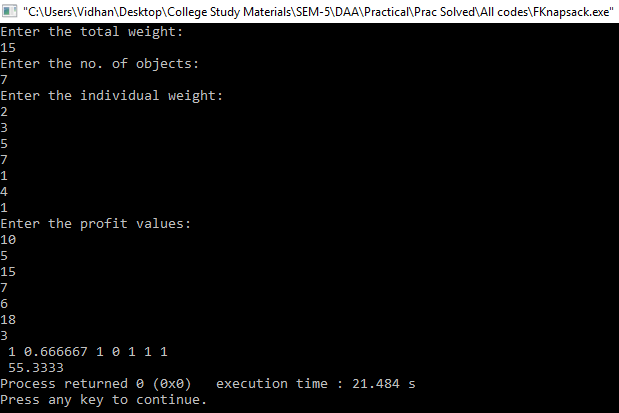
cout<<endl;

cout<<" "<<x;

return 0;

}

**Output: -**



**Conclusion: -**

Fractional Knapsack is a greedy algorithm, and it has time complexity of O (n log (n)). O (n) for the division (ratio) of the profit value and the weight of the items or objects. O (log (n)) for sorting the values.

**4.3 Activity Selection Problem (Study/Implement)**

**Theory: -**

An activity-selection is the problem of scheduling a resource among several competing activity. In activity selection problem, we are given a set of activities and the starting and finishing time of each activity, we need to find the maximum number of activities that can be performed by a single person assuming that a person can only work on a single activity at a time.

Example:

Input:

(1, 4), (3, 5), (0, 6), (5, 7), (3, 8), (5, 9), (6, 10), (8,11), (8,12), (2,13), (12, 14).

Output: -

(1 4), (5 7), (8 11), (12 14

**Algorithm: -**

1. n = length [s]
2. A={i}
3. j = 1
4. FOR i = 2 to n
5. do if  si ≥ fj
6. then A= AU{i}
7. j = i
8. Return A

**Program: -**

**Code: -**

#include<iostream>

#include<iomanip>

using namespace std;

int main()

{

int n;

cout<<"Enter total activity : ";

cin>>n;

int a[n][2],x=0,b[n],y=0;

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

{

cin>>a[i][0];

cin>>a[i][1];

if(x<=a[i][0])

b[y]=i;

y++;

x=a[i][1];

}

}

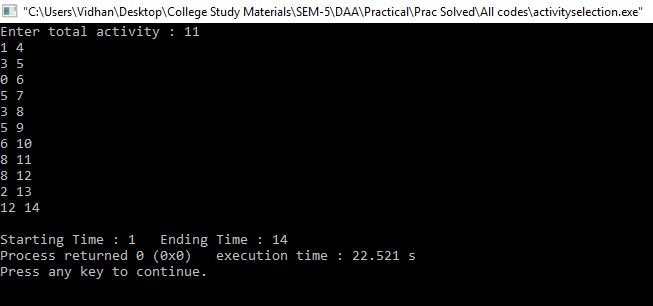
cout<<endl;

cout<<"Starting Time : "<<a[0][0]<<" Ending Time : "<<a[b[y-1]][1];

return 0;

}

**Output: -**



**Conclusion: -**

Activity selection is a greedy algorithm, which works at the time complexity of O (n log (n)).