CHAROTAR UNIVERSITY OF SCIENCE & TECHNOLOGY

DEVANG PATEL INSTITUTE OF ADVANCE TECHNOLOGY & RESEARCH

Computer Science & Engineering

NAME: PARTH NITESHKUMAR PATEL

ID: 19DCS098

SUBJECT: DESIGN AND ANALYSIS OF ALGORITHM

CODE: CS 351

GREEDY APPROACH

PRACTICAL-4.1

AIM:

A cashier at any mall needs to give change of an amount to customers many times in a day. Cashier has multiple number of coins available with different denominations which is described by a set C. Implement the program for a cashier to find the minimum number of coins required to find a change of a particular amount A. Output should be the total number of coins required of given denominations. Check the program for following test cases:

Test Case	Coin denominations C	Amount A
1	₹1,₹2,₹3	₹ 5
2	₹18,₹17,₹5,₹1	₹ 22
3	₹100, ₹25, ₹10,₹5, ₹1	₹ 289

Is the output of Test case 2 is optimal? Write your observation.

PROGRAM CODE:

```
#include <bits/stdc++.h>
using namespace std;
void greedyApproach(int sum,int length,int coins[])
{
    int answer[sum];
    int i=0,count=0;

    for(i=length-1;i>=0;i--)
    {
        while(sum>=coins[i])
        {
            sum=sum-coins[i];
            answer[count]=coins[i];
            count++;
        }
    }
    cout<<"REQUIRED NUMBER OF COINS : "<<count<<endl;
    cout<<"THERE COIN VALUES : "<<endl;
    for(i=0;i<count;i++)
        cout<<answer[i]<<" ";
    cout<<endl;
}
int main()
{
    int n,i,sum;
    cout<<"ENTER TOTAL NUMBER OF COINS : ";
    cin>>n;
    int coins[n];
    sort(coins,coins+n);
```

```
    cout<<"ENTER THE VALUES OF COINS : ";
    for(i=0;i<n;i++)
        cin>>coins[i];

    cout<<"Enter the final sum of coins : ";
    cin>>sum;
    greedyApproach(sum,n,coins);
    cout<<"PARTH PATEL\n19DCS098"<<endl;
    return 0;
}
```

OUTPUT:

```
ENTER TOTAL NUMBER OF COINS : 4
ENTER THE VALUES OF COINS : 1 2 3 5
Enter the final sum of coins : 9
REQUIRED NUMBER OF COINS : 3
THERE COIN VALUES :
5 3 1
PARTH PATEL
19DCS098
```

TEST CASE-1:

```
ENTER TOTAL NUMBER OF COINS : 3
ENTER THE VALUES OF COINS : 1 2 3
Enter the final sum of coins : 5
REQUIRED NUMBER OF COINS : 2
THERE COIN VALUES :
3 2
PARTH PATEL
19DCS098
```

TEST CASE-2:

```
ENTER TOTAL NUMBER OF COINS : 4
ENTER THE VALUES OF COINS : 18 17 5 1
Enter the final sum of coins : 22
REQUIRED NUMBER OF COINS : 5
THERE COIN VALUES :
18 1 1 1 1
PARTH PATEL
19DCS098
```

REMARKS:

- **Expected Output:** (2 Coins) 17 & 5
- **Current Output:** 18 1 1 1 1 is correct but not optimal
- **Greedy Approach Failed to optimize the answer.**

TEST CASE-3:

```
ENTER TOTAL NUMBER OF COINS : 5
ENTER THE VALUES OF COINS : 100 25 10 5 1
Enter the final sum of coins : 289
REQUIRED NUMBER OF COINS : 10
THERE COIN VALUES :
100 100 25 25 25 10 1 1 1 1
PARTH PATEL
19DCS098
```

CONCLUSION:

By performing the above practical, we learnt the application of greedy algorithm and also about a corner case where it fails.

PRACTICAL-4.2

AIM:

Let S be a collection of objects with profit-weight values. Implement the fractional knapsack problem for S assuming we have a sack that can hold objects with total weight W. Check the program for following test cases:

Test Case	S	profit-weight values	W
1	{A,B,C}	Profit:(1,2,5) Weight: (2,3,4)	5
2	{A,B,C,D,E,F,G}	Profit:(10,5,15,7,6,18,3) Weight: (2,3,5,7,1,4,1)	15
3	{A,B,C,D,E,F,G}	A:(12,4),B:(10,6), C:(8,5),D:(11,7), E:(14,3),F:(7,1), G:(9,6)	18

PROGRAM CODE:

```
#include <iostream>
#include <bits/stdc++.h>
using namespace std;

typedef struct {
   double v;
   double w;
} Item;

void input(Item items[],int sizeOfItems) {
   cout << "Enter total "<< sizeOfItems <<" Item's values and
weight" <<
   endl;
   for(int i = 0; i < sizeOfItems; i++) {
      cout << "ENTER V : "<<i+1<<" : ";
      cin >> items[i].v;
      cout << "ENTER W : "<< i+1 << " : ";
      cin >> items[i].w;
   }
}
void display(Item items[], int sizeOfItems) {
   int i;
   cout << "values: ";
   for(i = 0; i < sizeOfItems; i++) {
      cout << items[i].v << "\t";
   }
   cout << endl << "weight: ";
   for (i = 0; i < sizeOfItems; i++) {
      cout << items[i].w << "\t";
```

```
   }
   cout << endl;
}
bool compare(Item a, Item b) {
    double r1 = (double)(a.v / a.w);
    double r2 = (double)(b.v / b.w);
    return r1 > r2;
}
double knapsack(Item items[], int sizeOfItems, int W) {
   int i, j;
   double totalValue = 0, totalWeight = 0;

   cout<<"PROFIT PER UNIT WEIGHT :\n";
   cout<<"Value     Weight    Profit\n";
   for (int i = 0; i < sizeOfItems; i++)
    {
        cout << items[i].v << "          " << items[i].w <<
"          "
           << ((double)items[i].v / items[i].w) << endl;
    }
   sort(items, items+sizeOfItems, compare);
   for(i=0; i<sizeOfItems; i++) {
      if(totalWeight + items[i].w<= W) {
         totalValue += items[i].v ;
         totalWeight += items[i].w;
      } else {
         int wt = W-totalWeight;
         totalValue += items[i].v*((double)wt / items[i].w);
         totalWeight += wt;
         break;
      }
   }
   cout << "TOTAL WEIGHT IN THE BAG: " << totalWeight<<endl;
```

```
    return totalValue;
}
int main() {
    int W,n;
    cout<<"ENTER THE TOTAL NUMBER OF ITEMS:";
    cin>>n;
    Item items[n];
    input(items, n);
    cout << "DATA :\n";
    display(items,n);
    cout<< "ENTER THE KNAPSACK WEIGHT: \n";
    cin >> W;
    double mxVal = knapsack(items, n, W);
    cout << "MAXIMUM PROFIT FOR "<< W <<" WEIGHT : "<< mxVal;
    cout<<endl;
    cout<<"PARTH PATEL\n19DCS098";
    return 0;
}
```

OUTPUT:

TEST CASE-1:

```
ENTER THE TOTAL NUMBER OF ITEMS:3
Enter total 3 Item's values and weight
ENTER V : 1 : 1
ENTER W : 1 : 2
ENTER V : 2 : 2
ENTER W : 2 : 3
ENTER V : 3 : 5
ENTER W : 3 : 4
DATA :
values: 1       2       5
weight: 2       3       4
ENTER THE KNAPSACK WEIGHT:
5
PROFIT PER UNIT WEIGHT :
Value     Weight    Profit
1         2         0.5
2         3         0.666667
5         4         1.25
TOTAL WEIGHT IN THE BAG: 5
MAXIMUM PROFIT FOR 5 WEIGHT : 5.66667
PARTH PATEL
19DCS098
```

TEST CASE-2:

```
DATA :
values: 10      5       15      7       6       18      3
weight: 2       3       5       7       1       4       1
ENTER THE KNAPSACK WEIGHT:
15
PROFIT PER UNIT WEIGHT :
Value     Weight    Profit
10         2         5
5         3         1.66667
15         5         3
7         7         1
6         1         6
18         4         4.5
3         1         3
TOTAL WEIGHT IN THE BAG: 15
MAXIMUM PROFIT FOR 15 WEIGHT : 55.3333
PARTH PATEL
19DCS098
```

TEST CASE-3:

```
DATA :
values: 12      10      8       11      14      7       9
weight: 4       6       5       7       3       1       6
ENTER THE KNAPSACK WEIGHT:
18
PROFIT PER UNIT WEIGHT :
Value     Weight    Profit
12         4         3
10         6         1.66667
8         5         1.6
11         7         1.57143
14         3         4.66667
7         1         7
9         6         1.5
TOTAL WEIGHT IN THE BAG: 18
MAXIMUM PROFIT FOR 18 WEIGHT : 49.4
PARTH PATEL
19DCS098
```

CONCLUSION:

By performing the above practical, we learnt the concept and application of knapsack problem using greedy algorithm.

PRACTICAL-4.3

AIM:

Suppose you want to schedule N activities in a Seminar Hall. Start time and Finish time of activities are given by pair of (si,fi) for ith activity.

Implement the program to maximize the utilization of Seminar Hall. (Maximum activities should be selected.)

Test Case	Number of activities (N)	(si,fi)
1	9	(1,2), 1,3),(1,4),(2,5),(3,7), (4,9), (5,6), (6,8), (7,9)
2	11	(1,4),(3,5),(0,6),(3,8),(5,7), (5,9), (6,10), (8,12),(8,11) (12,14), (2,13)

PROGRAM CODE:

```
#include <iostream>
#include <bits/stdc++.h>
using namespace std;
typedef struct {
   int s;
   int f;
} activ;

bool compare(activ a, activ b) {
    return a.f < b.f;
}

void MaxAct(activ arr[], int n)
{
    sort(arr, arr+n, compare);

    cout << "Following activities are selected :\n";
    int i = 0;
    cout << "(" << arr[i].s << ", " << arr[i].f << "), ";
    for (int j = 1; j < n; j++)
    {
      if (arr[j].s >= arr[i].f)
      {
          cout << "(" << arr[j].s << ", " << arr[j].f << "), ";
          i = j;
      }
    }
}
```

```
void input(activ arr[],int lng) {
   cout << "Enter total "<< lng <<" Item's Start and Finish time :-
\n\n";
   for(int i = 0; i < lng; i++) {
      cout << "Enter Start Time For Activity "<< i+1<<":" ;
      cin >> arr[i].s;
      cout << "Enter Finish Time For Activity "<< i+1<<":";
      cin >> arr[i].f;
      cout<<"\n";
   }
}
void display(activ arr[], int lng) {
   int i;
   cout << "Start Time: ";
   for(i = 0; i < lng; i++) {
      cout << "\t"<< arr[i].s ;
   }
   cout << endl << "Finish Time: ";
   for (i = 0; i < lng; i++) {
      cout << "\t"<< arr[i].f ;
   }
   cout << endl;
}


int main() {
   int n;
   cout<<"Enter total number of Activities :";
   cin>>n;
   activ arr[n];
   input(arr, n);
   cout << "Entered data \n";
   display(arr,n);
```

```
  MaxAct(arr, n);

  cout<<"PARTH PATEL\n19DCS098"<<endl;
  return 0;

  }
```

OUTPUT:

TEST CASE-1:

```
Entered data
Start Time:     1       1       1       2       3       4       5       6
   7
Finish Time:    2       3       4       5       7       9       6       8
   9
Following activities are selected :
(1, 2), (2, 5), (5, 6), (6, 8),
PARTH PATEL
19DCS098
```

TEST CASE-2:

```
Entered data
Start Time:     1       3       0       3       5       5       6       8
   8    12      2
Finish Time:    4       5       6       8       7       9       10      12
   11   14      13
Following activities are selected :
(1, 4), (5, 7), (8, 11), (12, 14),
PARTH PATEL
19DCS098
```

CONCLUSION:

By implementing the above practical, we learnt one more application of greedy algorithm