**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 Coin denominations C Amount A** |
| **Case** |
| 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;

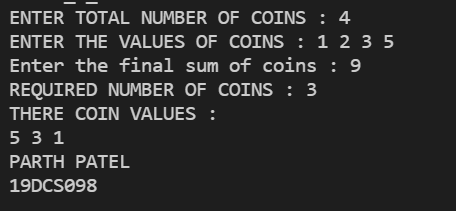
    greedyApproach(sum,n,coins);

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

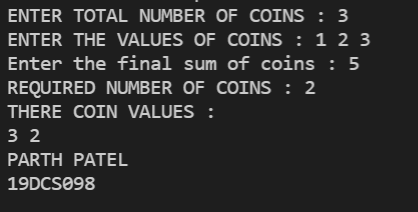
    return 0;

}

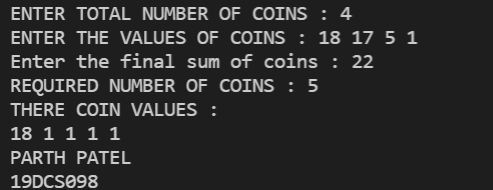
**OUTPUT:**



**TEST CASE-1:**



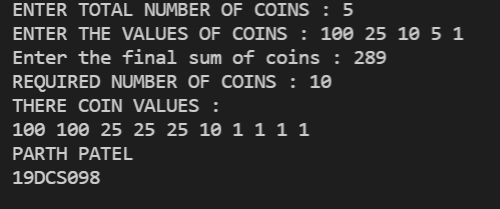
**TEST CASE-2:**



**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:**



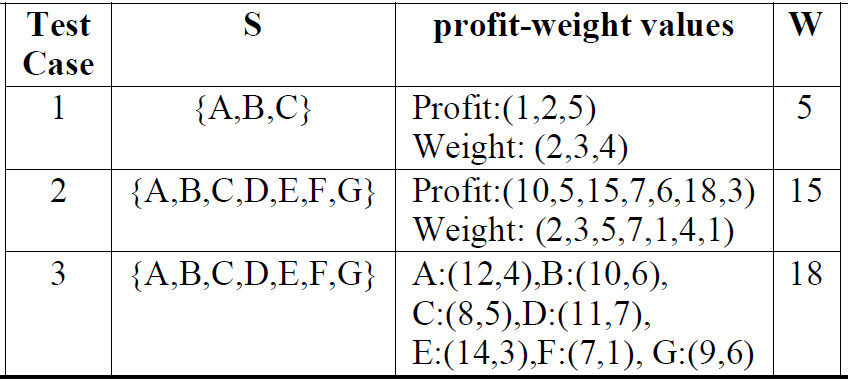
**CONCLUSION:**

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:



**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;

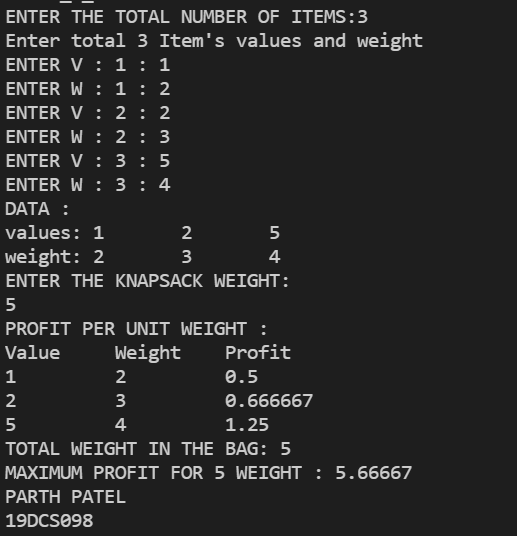
   cout<<"PARTH PATEL\n19DCS098";

   return 0;

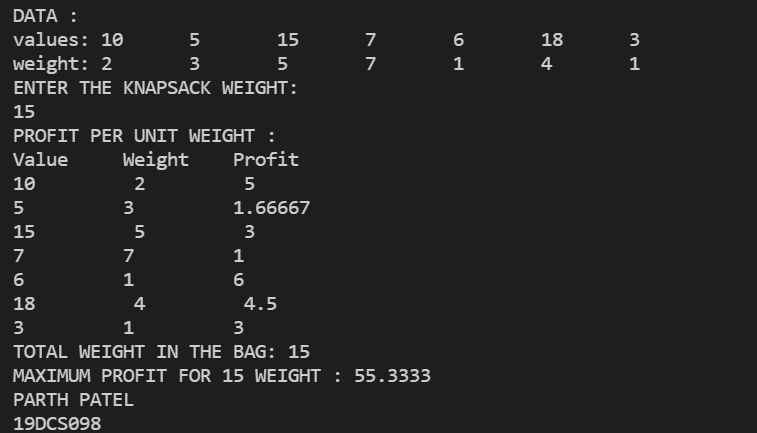
}

**OUTPUT:**

**TEST CASE-1:**



**TEST CASE-2:**



**TEST CASE-3:**



**CONCLUSION:**

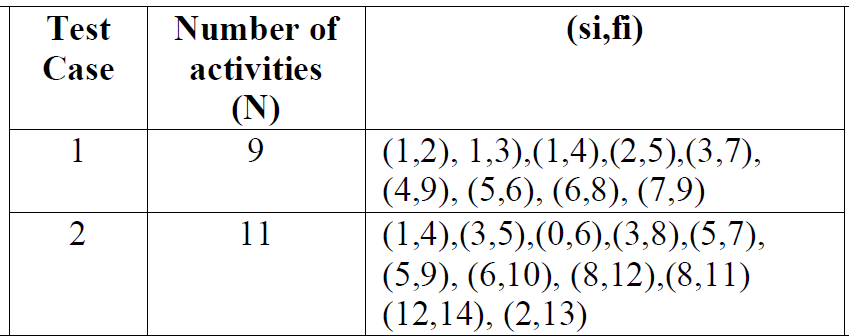
By performing the above practical, we learnt the concept and application of knapsack problem using greedy algorithm.

**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.)



**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);

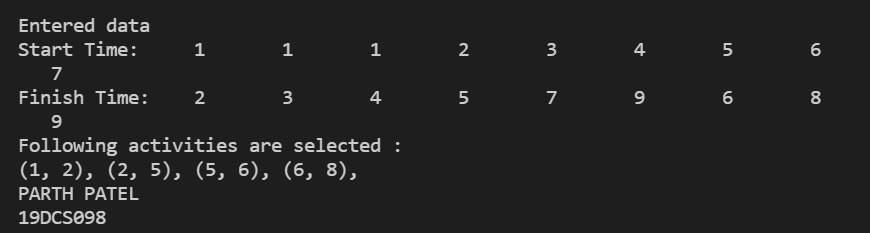
   cout<<"PARTH PATEL\n19DCS098"<<endl;

   return 0;

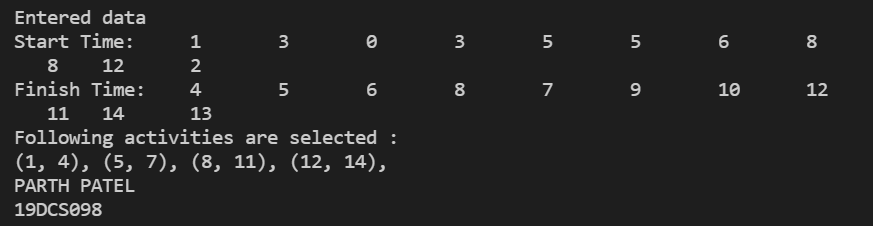
   }

**OUTPUT:**

**TEST CASE-1:**



**TEST CASE-2:**



**CONCLUSION:**

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