**PRACTICAL-4 GREEDY APPROCH**

4.1 a. A Burglar has just broken into the Fort! He sees himself in a room with n piles of gold dust. Because each pile has a different purity, each pile also has a different value (v[i]) and a different weight (w[i]). A Burglar has a bag that can only hold W kilograms. Given n=5, v={4,2,2,1,10}, c={12,1,2,1,4} and W=15,calculate which piles Burglar should completely put into his bag and which he should put only fraction into his bag. Design and implement an algorithm to get maximum piles of gold using given bag with W capacity, Burglar is also allowed to take fractional of pile.

**CODE:-**

#include<iostream>

using namespace std;

int l=0,m=0;

void knapsack(int n, float weight[], float profit[], float capacity) {

float x[20], tp = 0;

int i, j, u;

//bool a[20]={false};

u = capacity;

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

if (weight[i] > u)

break;

else {

tp = tp + profit[i];

u = u - weight[i];

l++;

// a[i]=true;

}

}

cout<<"Full gold piles are: ";

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

{

cout<<profit[j]<<endl;

}

cout<<endl;

if (i < n)

{

x[i] = u / weight[i];

cout<<"Fractional Gold pils(Values): "<<profit[i]<<endl;

m++;

}

tp = tp + (x[i] \* profit[i]);

cout<<"Full gold piles are: "<<l<<endl;

cout<<"Fractional gold piles are: "<<m<<endl;

cout<<"\nMaximum profit is:"<<tp;

}

int main() {

float weight[20], profit[20], capacity;

int num, i, j;

float ratio[20], temp;

cout<<"\nEnter the no. of Objects:- ";

cin>>num;

cout<<"\nEnter the weigths and profits of each object:- ";

for (i = 0; i < num; i++) {

cin>>weight[i]>>profit[i];

}

cout<<"\nEnter the capacity of knapsack:- ";

cin>>capacity;

for (i = 0; i < num; i++) {

ratio[i] = profit[i] / weight[i];

}

for (i = 0; i < num; i++) {

for (j = i + 1; j < num; j++) {

if (ratio[i] < ratio[j]) {

temp = ratio[j];

ratio[j] = ratio[i];

ratio[i] = temp;

temp = weight[j];

weight[j] = weight[i];

weight[i] = temp;

temp = profit[j];

profit[j] = profit[i];

profit[i] = temp;

}

}

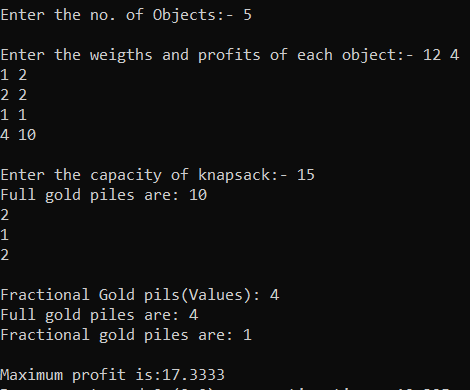
}

knapsack(num, weight, profit, capacity);

return 0;

}

**OUTPUT:-**



b. 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.

**CODE:-**

#include<iostream>

using namespace std;

int S[100];

int making\_change(int c[], int n, int amount)

{

int i,j, cnt = 0;

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

{

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

{

if (c[i]<c[j])

{

int t = c[i];

c[i]=c[j];

c[j]=t;

}

}

}

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

{

while(amount >= c[i])

{

amount -= c[i];

S[cnt] = c[i];

cnt++;

}

if(amount == 0)

break;

}

return cnt;

}

int main()

{

int amt,n,c[100];

cout<<"Enter the Number of coins :";

cin>>n;

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

{

cout<<"enter value of each coins: ";

cin>>c[i];

}

cout<<"Enter the Amount:";

cin>>amt;

int Coin\_Count = making\_change(c,n,amt);

cout<<"total coins needed :"<<Coin\_Count<<endl;

cout<<"Coins are:"<<endl;

for(int i = 0; i < Coin\_Count; i++)

{

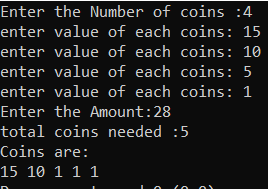
cout<<S[i]<<" ";

}

return 0;

}

**OUTPUT:-**



4.2 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:

**CODE:-**

#include<iostream>

using namespace std;

int l=0,m=0;

void knapsack(int n, float weight[], float profit[], float capacity , string object[]) {

float x[20], tp = 0;

int i, j, u;

u = capacity;

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

{

if (weight[i] > u)

break;

else {

tp = tp + profit[i];

u = u - weight[i];

l++;

}

}

cout<<"Full gold piles are: ";

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

{

cout<<object[j]<<endl;

}

cout<<endl;

if (i < n)

{

cout<<"Fractional Gold pils: "<<object[i]<<endl;

m++;

}

tp = tp + (x[i] \* profit[i]);

cout<<"Full gold piles are: "<<l<<endl;

cout<<"Fractional gold piles are: "<<m<<endl;

cout<<"\nMaximum profit is:"<<tp;

}

int main() {

string object[20];

float weight[20], profit[20], capacity;

int num, i, j;

float ratio[20], temp;

cout<<"\nEnter the no. of objects:- ";

cin>>num;

cout<<"\nEnter the Object,wts and profits of each object:- ";

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

{

cin>>object[i];

cin>>weight[i]>>profit[i];

}

cout<<"\nEnter the capacity of knapsack:- ";

cin>>capacity;

for (i = 0; i < num; i++) {

ratio[i] = profit[i] / weight[i];

}

for (i = 0; i < num; i++) {

for (j = i + 1; j < num; j++) {

if (ratio[i] < ratio[j]) {

temp = ratio[j];

ratio[j] = ratio[i];

ratio[i] = temp;

temp = weight[j];

weight[j] = weight[i];

weight[i] = temp;

temp = profit[j];

profit[j] = profit[i];

profit[i] = temp;

string t=object[j];

object[j]=object[i];

object[i]=t;

}

}

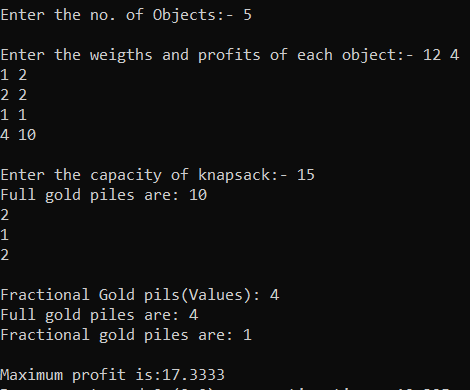
}

knapsack(num, weight, profit, capacity,object);

return 0;

}

**Output:-**



4.3 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.)

**CODE:-**

#include<iostream>

using namespace std;

void sort(int start[], int finish[], string AN[], int n)

{

int i, j;

for (i = 0; i < n-1; i++)

{

for (j = 0; j < n-i-1; j++)

{

if (finish[j] > finish[j+1])

{

string temp1=AN[j];

AN[j]=AN[j+1];

AN[j+1]=temp1;

int temp=start[j];

start[j]=start[j+1];

start[j+1]=temp;

temp=finish[j];

finish[j]=finish[j+1];

finish[j+1]=temp;

}

}

}

}

void ActivitySelection(int start[], int finish[],string AN[], int n)

{

cout<<"The following activities are selected: "<<endl;

sort(start,finish,AN,n);

int j = 0;

cout<<"Selected Activity is: "<<AN[j]<<endl;

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

{

if (start[i] >= finish[j])

{

cout<<"Selected Activity is: "<<AN[i]<<endl;

j = i;

}

}

}

int main()

{

int start[20],finish[20];

string AN[20];

int n;

cout<<"Enter Activities:-";

cin>>n;

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

{

cout<<"Enter Activity Name,Start time and end time: ";

cin>>AN[i]>>start[i]>>finish[i];

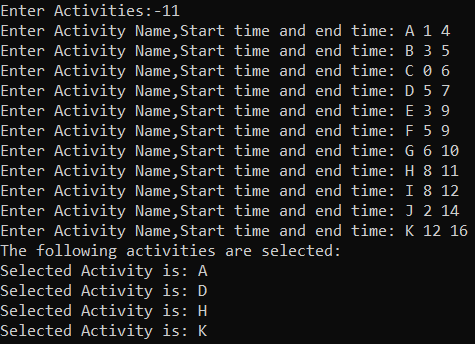
}

ActivitySelection(start, finish, AN, n);

return 0;

}

**OUTPUT:-**



4.4 In a conference, N people from a company XYZ are present to attend maximum number of presentations. Each presentation is scheduled between 8:00 and 8:00. Here the second 8:00 means 8:00 pm. Our task is to assign people to presentations such that the number of unique presentations attended by XYZ as a company is maximized. Input Format Input is provided in the form a file (taken as command line argument). The first line contains N i.e # of people. Second line contains M i.e # of presentation on that day. M lines follow each containing start and end time of presentation. Time will be in format of HH:MM. Sample Input#01 2 5 09:00 08:00 08:00 12:00 12:00 08:00 08:00 08:00 08:00 08:00 Sample Output#01 3 Explanation#01 One person can cover the 8- 12 presentation and the 12-8 presentation. The other person can cover one of the all-day presentations.

**CODE:-**

#include<bits/stdc++.h>

using namespace std;

struct node

{

int s;

int e;

};

bool comp(node a, node b)

{

return (a.e < b.e);

}

int main()

{

int n,p;

cin>>n;

cin>>p;

vector<int> v[n];

vector<node> v2;

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

{

string str;

cin>>str;

char h1,h2,m1,m2;

h1 = str[0];

h2 = str[1];

int t = 0;

t = t\*10 + h1 - '0';

t = t\*10 + h2 - '0';

int s = 0;

if(t >= 8)

{

s += (t-8)\*60;

}

else

{

s += (t+4)\*60;

}

char z,z2;

m1 = str[3];

m2 = str[4];

t = 0;

t = t\*10 + m1 - '0';

t = t\*10 + m2 - '0';

s += t;

cin>>str;

h1 = str[0];

h2 = str[1];

m1 = str[3];

m2 = str[4];

t = 0;

t = t\*10 + h1 - '0';

t = t\*10 + h2 - '0';

int e = 0;

if(t > 8)

{

e += (t-8)\*60;

}

else

{

e += (t+4)\*60;

}

int t2 = 0;

t2 = t2\*10 + m1 - '0';

t2 = t2\*10 + m2 - '0';

if(t == 8 && t2 > 0)

{

e = 0;

e += (t-8)\*60;

}

e += t2;

node temp;

temp.s = s;

temp.e = e;

v2.push\_back(temp);

}

sort(v2.begin(),v2.end(),comp);

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

{

v[i].push\_back(0);

}

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

{

int f = 0;

int max = -1;

int ind = -1;

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

{

if(v2[i].s >= v[j][v[j].size()-1] && v[j][v[j].size()-1] > max)

{

max = v[j][v[j].size()-1];

ind = j;

}

}

if(ind != -1)

v[ind].push\_back(v2[i].e);

}

int ans = 0;

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

{

ans = ans + v[i].size() - 1;

}

cout<<ans<<endl;

return 0;

}

**OUTPUT:-**

