**PRACTICAL - 4**

**4.1 AIM: Program to implement Kruskal’s algorithm using greedy method.**

**PROGRAM CODE**:

#include <bits/stdc++.h>

using namespace std;

+ rank by union

class DSU

{

int \*parent;

int \*rank;

public:

DSU(int n)

{

parent = new int[n];

rank = new int[n];

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

{

parent[i] = -1;

rank[i] = 1;

}

}

int find(int i)

{

if (parent[i] == -1)

return i;

return parent[i] = find(parent[i]);

}

void unite(int x, int y)

{

int s1 = find(x);

int s2 = find(y);

if (s1 != s2)

{

if (rank[s1] < rank[s2])

{

parent[s1] = s2;

rank[s2] += rank[s1];

}

else

{

parent[s2] = s1;

rank[s1] += rank[s2];

}

}

}

};

class Graph

{

vector<vector<int>> edgelist;

int V;

public:

Graph(int V) { this->V = V; }

void addEdge(int x, int y, int w)

{

edgelist.push\_back({w, x, y});

}

void kruskals\_mst()

{

sort(edgelist.begin(), edgelist.end());

DSU s(V);

int ans = 0;

cout << "Following are the edges in the "

"constructed MST"

<< endl;

for (auto edge : edgelist)

{

int w = edge[0];

int x = edge[1];

int y = edge[2];

if (s.find(x) != s.find(y))

{

s.unite(x, y);

ans += w;

cout << x << " -- " << y << " == " << w

<< endl;

}

}

cout << "Minimum Cost Spanning Tree: " << ans;

}

};

int main()

{

Graph g(4);

g.addEdge(0, 1, 10);

g.addEdge(1, 3, 15);

g.addEdge(2, 3, 4);

g.addEdge(2, 0, 6);

g.addEdge(0, 3, 5);

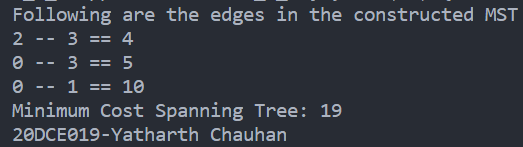
g.kruskals\_mst();

cout << "\n20DCE019-Yatharth Chauhan";

return 0;

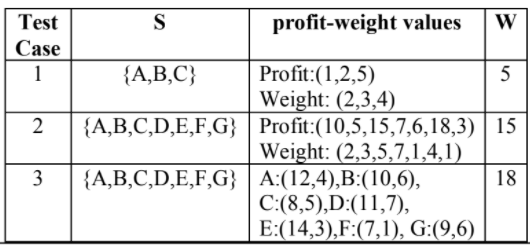
}

**OUTPUT**:



**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 CODE 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 " << i + 1 << " V ";

cin >> items[i].v;

cout << "Enter " << i + 1 << " W ";

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 of 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 bag " << totalWeight << endl;

return totalValue;

}

int main()

{

int W, n;

cout << "Enter total number of items :";

cin >> n;

Item items[n];

input(items, n);

cout << "Entered data \n";

display(items, n);

cout << "Enter Knapsack weight \n";

cin >> W;

double mxVal = knapsack(items, n, W);

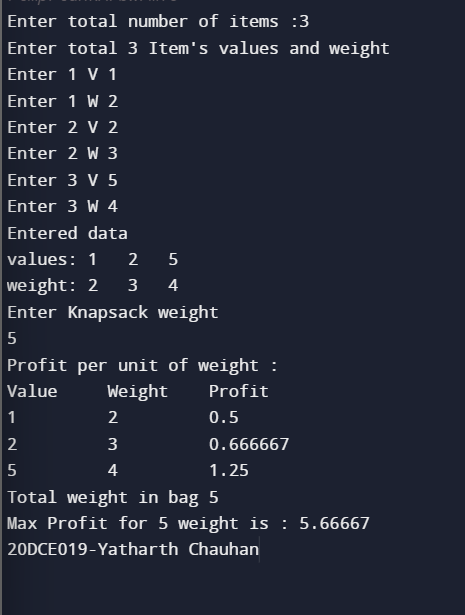
cout << "Max Profit for " << W << " weight is : " << mxVal;

cout << "\n20DCE019-Yatharth Chauhan";

return 0;

}

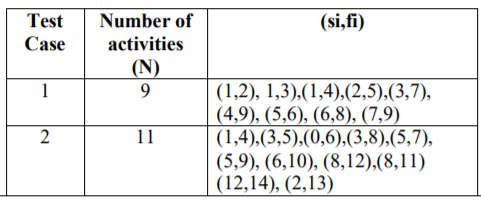
**OUTPUT**:



**CONCLUSION:**

In this practical we have tried to solve the fractional knapsack problem. The problem is based on getting maximum profit items in the knapsack having fixed carry capacity and also to utilize maximum (whole, in fractional knapsack) carrying capacity of the knapsack. Here greedy approach is used to solve the problem.

**4.3 AIM: Suppose you want to schedule N activities in a SeminarHall. Start time and Finish time of activities are given bypair of (si,fi) for ith activity.Implement the PROGRAM CODE to maximize the utilization ofSeminar Hall. (Maximum activities should be selected.)**



**PROGRAM CODE**:

#include <iostream>

#include <bits/stdc++.h>

using namespace std;

typedef struct

{

int s;

int f;

} activ;

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;

}

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;

}

}

}

int main()

{

int n;

cout << "Enter total number of Activities :";

cin >> n;

activ arr[n];

input(arr, n);

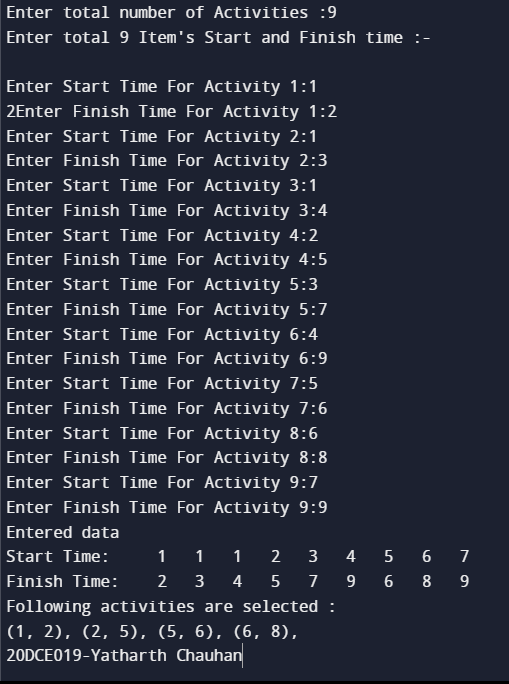
cout << "Entered data \n";

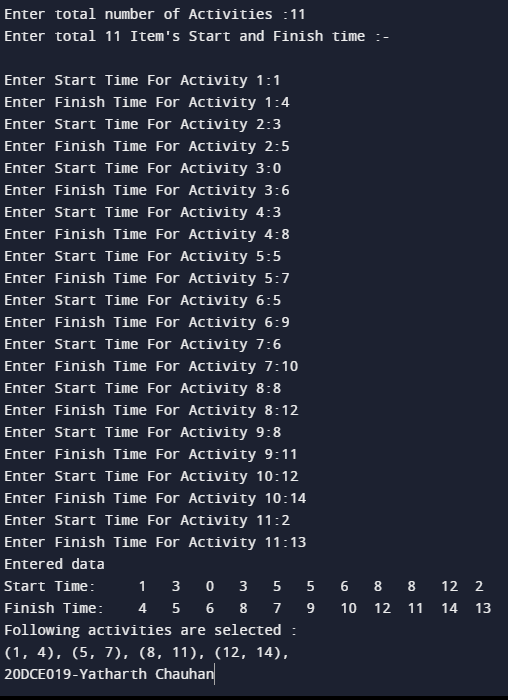
display(arr, n);

MaxAct(arr, n);

cout << "\n20DCE019-Yatharth Chauhan";

}

**OUTPUT**:



**CONCLUSION:**

In this practical we have attempted to solve the activity selection problem. The problem is based on getting maximum activities completed such that maximum amount of the schedule can be utilized .In the problem definition we are given to solve this problem for seminar hall. Here greedy approach is used to solve the problem which allows to complete maximum activities so that maximum schedule time of seminar hall can be used.