Rajan Gautam
19BCP101

Div. II, CE 19

SOT, PDEU

Pandit Deendayal Energy University
School of Technology

Design & Analysis of Algorithm (20CP209P)

B. Tech - Computer Science & Engineering (Sem-IV)

Table of Contents

Lab 3 Assignment: Implementation of Greedy Algorithm

AIM 1: To write a C/C++ Program to implement Greedy Algorithm for given problem.

Aim: Write a C/C++ Program (preferably) to implement Greedy Algorithm for given problem.

1. Consider the following 6 activities sorted by their finish time

Start[]	1	3	0	5	8	5
Finish[]	2	4	6	7	9	9

Print maximum set of activities that can be done by a single person, one at a time.

ALGORITHMS:

1. Insertion Sort Algorithm (Pseudocode)

INSERTION-SORT (A, n)

 for j ← 2 to n

  Do key ← A[j]

   i ← j – 1

   while i > 0 and A[i] > key

    Do A [i + 1] ← A[i]

     i ← i – 1

   A [i + 1] = key

2. Greedy Algorithm

Sort input activities in order by increasing finishing time.

n ← length[s]

A ← 1

j ← 1

for I ← 2 to n

 if $s_i \geq f_j$ then

 A ← A ∪ (i)

 j ← i

return A

CODE:

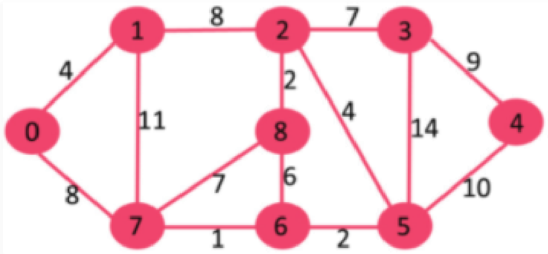
```
#include <stdio.h>
#include <stdlib.h>

// Algorithm to sort array
void insertion_sort(int A[], int n)
{
    int i, j, key;
    for(i = 1; i < n; i++)
    {
        key = A[i];
        j = i - 1;
        while (j >= 0 && A[j] > key)
        {

            A[j+1] = A[j];
            j = j -1;
        }
        A[j+1] = key;
    }
}

// Algorithm for Greedy
int GreedyAlgorithm(int S[], int F[], int n)
{
    int A = 1;
    int j = 1;
    for (int i = 2; i <= n; i++)
    {
        if(S[i] >= F[j])
        {
            A++;
            j = i;
        }
    }
    return A;
}

// Main Function
int main()
{
    // Scanning the total number of Activities
    int numActivity;
    printf("Enter total number of Activity: ");
    scanf("%d", &numActivity);

    // Declaring two arrays
    int start[numActivity];
    int finish[numActivity];

    // Getting the values of start Array
    printf("Enter starting time for %d Activities \n", numActivity);
```

```c

    for (int i = 0; i < numActivity; i++)
    {
        scanf("%d", &start[i]);
    }

    // Getting the values for final Array
    printf("Enter finishing time for %d Activities \n", numActivity);

    for (int i = 0; i < numActivity; i++)
    {
        scanf("%d", &finish[i]);
    }

    // Sorting the finish array using Insertion Sort Algorithm
    insertion_sort(finish, numActivity);

    // Getting the maximum set of activities that can be done by a single person
    int maxNum = GreedyAlgorithm(start, finish, numActivity);

    // Printing maximum number returned by our GreedyAlgorithm Function
    printf("Maximum Set of Activities that can be done by a single person: %d\n", maxNum);

    return 0;
}
```


OUTPUT:

AIM 2: To Implement Greedy Algorithm for computing Minimum Spanning Tree using Kruskal Algorithm.

2. Implement Greedy Algorithm for computing Minimum Spanning Tree Using (Kruskal/Prim's) Algorithm.



1. Print the number of edges and total cost of Minimum Spanning Tree.
2. Analyze the time complexity of your algorithm.

Kruskal Algorithm:

KRUSKAL(G):

A = ∅

For each vertex v ∈ G.V:

    MAKE-SET(v)

For each edge (u, v) ∈ G.E ordered by increasing order by weight (u, v):

    if FIND-SET(u) ≠ FIND-SET(v):

    A = A ∪ {(u, v)}

    UNION (u, v)

CODE:

```
#include <stdio.h>
#define MAX 30

typedef struct edge
{
    int u, v, w;
}   edge;

typedef struct edge_list
{
    edge data[MAX];
    int n;
```

```
}   edge_list;

edge_list elist;

int Graph[MAX][MAX], n;
edge_list spanlist;

void kruskalAlgo();
int find(int belongs[], int vertexno);
void applyUnion(int belongs[], int c1, int c2);
void sort();
void print();

// Applying Krushkal Algo
void kruskalAlgo()
{
    int belongs[MAX], i, j, cno1, cno2;
    elist.n = 0;

    for (i = 1; i < n; i++)
        for (j = 0; j < i; j++)
        {
            if (Graph[i][j] != 0)
            {
                elist.data[elist.n].u = i;
                elist.data[elist.n].v = j;
                elist.data[elist.n].w = Graph[i][j];
                elist.n++;
            }
        }

    sort();

    for (i = 0; i < n; i++)
        belongs[i] = i;

    spanlist.n = 0;

    for (i = 0; i < elist.n; i++)
    {
        cno1 = find(belongs, elist.data[i].u);
        cno2 = find(belongs, elist.data[i].v);

        if (cno1 != cno2)
        {
            spanlist.data[spanlist.n] = elist.data[i];
            spanlist.n = spanlist.n + 1;
            applyUnion(belongs, cno1, cno2);
        }
    }
}

int find(int belongs[], int vertexno)
```

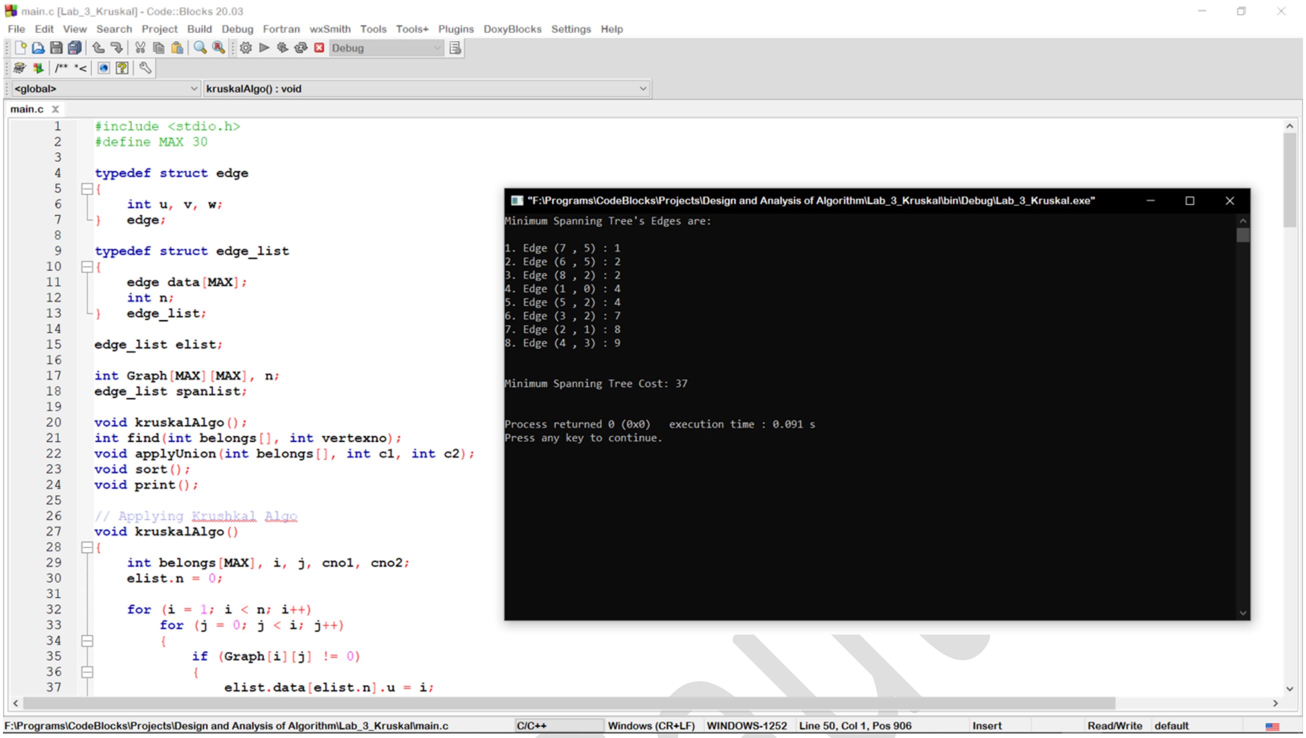
```
{
    return (belongs[vertexno]);
}

void applyUnion(int belongs[], int c1, int c2)
{
    int i;

    for (i = 0; i < n; i++)
        if (belongs[i] == c2)
            belongs[i] = c1;
}

// Sorting algo
void sort()
{
    int i, j;
    edge temp;

    for (i = 1; i < elist.n; i++)
        for (j = 0; j < elist.n - 1; j++)
            if (elist.data[j].w > elist.data[j + 1].w)
            {
                temp = elist.data[j];
                elist.data[j] = elist.data[j + 1];
                elist.data[j + 1] = temp;
            }
}

// Printing the result
void print()
{
    int i, cost = 0;
    printf("Minimum Spanning Tree's Edges are: \n\n");
    for (i = 0; i < spanlist.n; i++)
    {
        printf("%d. Edge (%d , %d) : %d\n", i+1, spanlist.data[i].u, spanlist.data[i].v, spanlist.data[i].w);
        cost = cost + spanlist.data[i].w;
    }

    printf("\n\nMinimum Spanning Tree Cost: %d\n\n", cost);
}

int main()
{
    n = 9;

    Graph[0][0] = 0;
    Graph[0][1] = 4;
    Graph[0][2] = 0;
    Graph[0][3] = 0;
    Graph[0][4] = 0;
```

```
    Graph[0][5] = 0;
    Graph[0][6] = 0;
    Graph[0][7] = 8;
    Graph[0][8] = 0;


    Graph[1][0] = 4;
    Graph[1][1] = 0;
    Graph[1][2] = 8;
    Graph[1][3] = 0;
    Graph[1][4] = 0;
    Graph[1][5] = 0;
    Graph[1][6] = 0;
    Graph[1][7] = 11;
    Graph[1][8] = 0;


    Graph[2][0] = 0;
    Graph[2][1] = 8;
    Graph[2][2] = 0;
    Graph[2][3] = 7;
    Graph[2][4] = 0;
    Graph[2][5] = 4;
    Graph[2][6] = 0;
    Graph[2][7] = 0;
    Graph[2][8] = 2;


    Graph[3][0] = 0;
    Graph[3][1] = 0;
    Graph[3][2] = 7;
    Graph[3][3] = 0;
    Graph[3][4] = 9;
    Graph[3][5] = 14;
    Graph[3][6] = 0;
    Graph[3][7] = 0;
    Graph[3][8] = 0;


    Graph[4][0] = 0;
    Graph[4][1] = 0;
    Graph[4][2] = 0;
    Graph[4][3] = 9;
    Graph[4][4] = 0;
    Graph[4][5] = 10;
    Graph[4][6] = 0;
    Graph[4][7] = 0;
    Graph[4][8] = 0;


    Graph[5][0] = 0;
    Graph[5][1] = 0;
    Graph[5][2] = 4;
```

```
    Graph[5][3] = 14;
    Graph[5][4] = 10;
    Graph[5][5] = 0;
    Graph[5][6] = 2;
    Graph[5][7] = 0;
    Graph[5][8] = 0;


    Graph[6][0] = 0;
    Graph[6][1] = 0;
    Graph[6][2] = 0;
    Graph[6][3] = 0;
    Graph[6][4] = 0;
    Graph[6][5] = 2;
    Graph[6][6] = 0;
    Graph[6][7] = 1;
    Graph[6][8] = 6;


    Graph[7][0] = 8;
    Graph[7][1] = 11;
    Graph[7][2] = 0;
    Graph[7][3] = 0;
    Graph[7][4] = 0;
    Graph[7][5] = 1;
    Graph[7][6] = 0;
    Graph[7][7] = 0;
    Graph[7][8] = 7;


    Graph[8][0] = 0;
    Graph[8][1] = 0;
    Graph[8][2] = 2;
    Graph[8][3] = 0;
    Graph[8][4] = 0;
    Graph[8][5] = 0;
    Graph[8][6] = 6;
    Graph[8][7] = 7;
    Graph[8][8] = 0;


    kruskalAlgo();
    print();
}
```


(i) Print the number of edges and total cost of Minimum Spanning Tree.

OUTPUT:



(ii) Analyze the time complexity of your algorithm.

➢ The Time Complexity of Kruskal Algorithm is O (E log E).

AIM 3: To solve the given problem.

3. A thief enters a house for robbing it. He can carry a maximal weight of 60 kg into his bag. There are 4 items in the house with the following weights and values.

Item	A	B	C	D
Profit	280	100	120	120
Weight	40	10	20	24

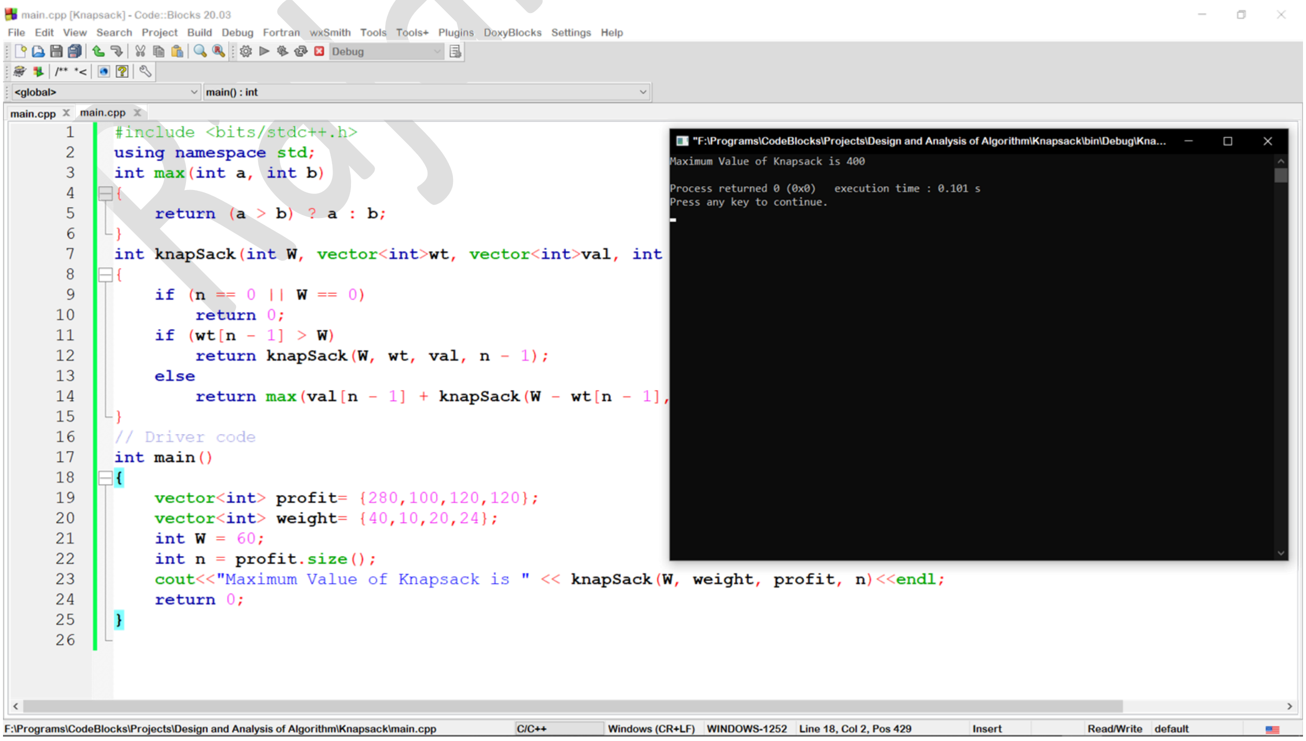
1. Perform the 0-1 Knapsack and print the maximum weight and value of Knapsack.
2. Perform Fractional Knapsack and print the maximum value of Knapsack.

1. Perform the 0-1 Knapsack and print the maximum weight and value of Knapsack.

CODE:

```
#include <bits/stdc++.h>
using namespace std;
int max(int a, int b)
{
    return (a > b) ? a : b;
}
int knapSack(int W, vector<int>wt, vector<int>val, int n)
{
    if (n == 0 || W == 0)
        return 0;
    if (wt[n - 1] > W)
        return knapSack(W, wt, val, n - 1);
    else
        return max(val[n - 1] + knapSack(W - wt[n - 1], wt, val, n - 1), knapSack(W, wt, val, n - 1));
 }
 // Driver code
 int main()
 {
    vector<int> profit= {280,100,120,120};
    vector<int> weight= {40,10,20,24};
    int W = 60;
    int n = profit.size();
    cout<<"Maximum Value of Knapsack is " << knapSack(W, weight, profit, n)<<endl;
    return 0;
 }
```


OUTPUT 1:

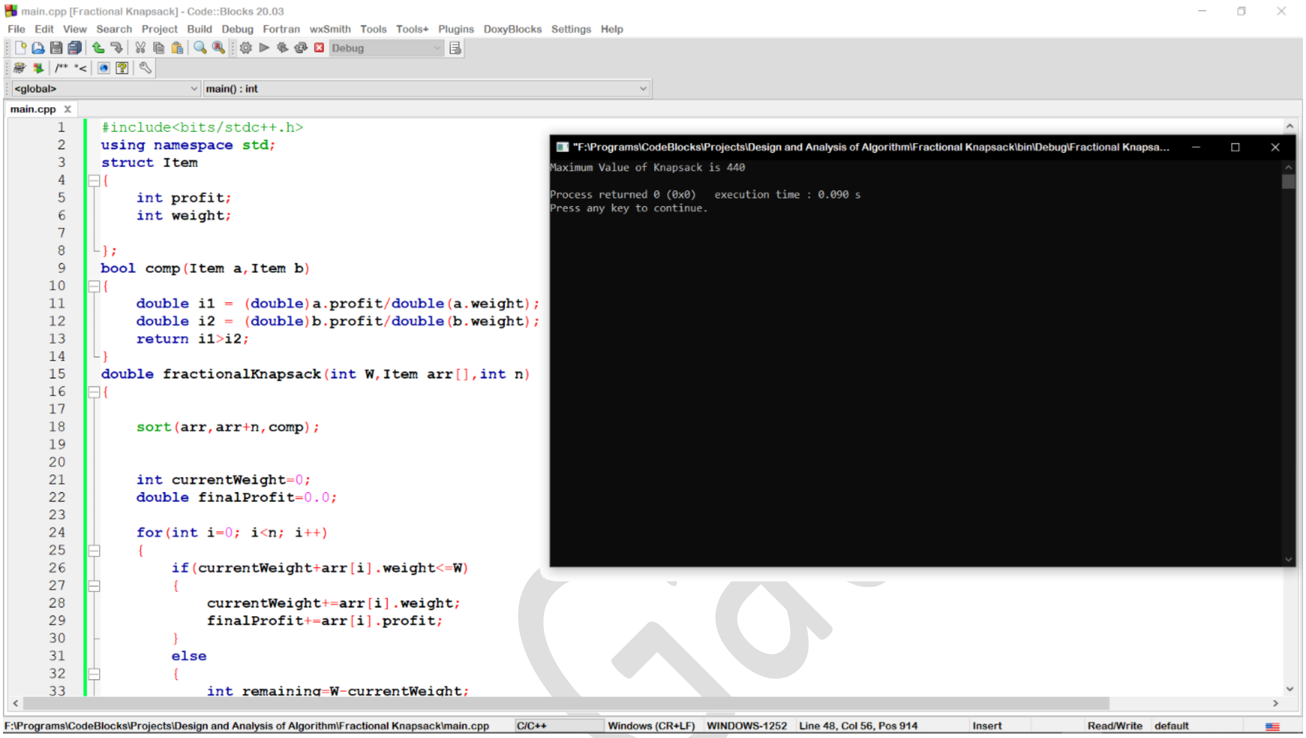


2. Perform Fractional Knapsack and print the maximum value of Knapsack.

```
#include<bits/stdc++.h>
using namespace std;
struct Item
{
    int profit;
    int weight;

};
bool comp(Item a,Item b)
 {
     double i1 = (double)a.profit/double(a.weight);
     double i2 = (double)b.profit/double(b.weight);
     return i1>i2;
 }
 double fractionalKnapsack(int W,Item arr[],int n)
 {

     sort(arr,arr+n,comp);


     int currentWeight=0;
     double finalProfit=0.0;

     for(int i=0; i<n; i++)
     {
         if(currentWeight+arr[i].weight<=W)
         {
             currentWeight+=arr[i].weight;
             finalProfit+=arr[i].profit;
         }
         else
         {
             int remaining=W-currentWeight;
             finalProfit+=(arr[i].profit*((double)remaining)/(double)arr[i].weight);

             break;
         }
     }

     return finalProfit;

 }
 int main()
 {

     int W=60;
     int n=4;
     Item arr[4]= {{280,40},{100,10},{120,20},{120,24}};
     cout<< "Maximum Value of Knapsack is " <<fractionalKnapsack(W,arr,n)
<<endl;
```

```
    return 0;
}
```

OUTPUT 2:



Link: https://github.com/rgautam320/Design-and-Analysis-of-Algorithm-Lab/tree/master/Lab_3_Greedy