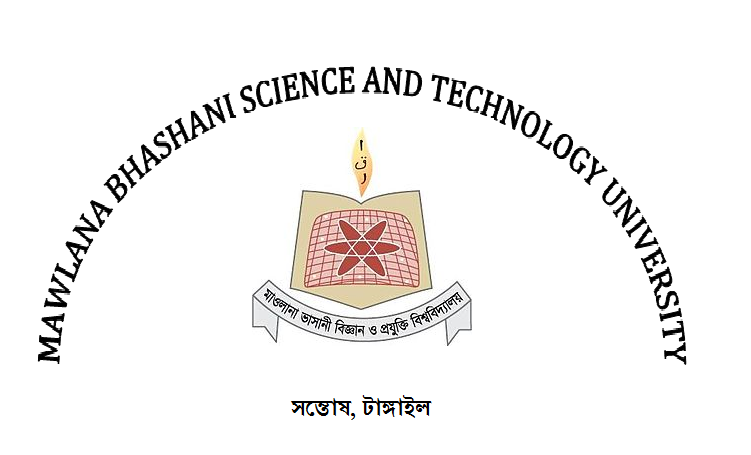
**Lab Report-04**

**Report Name : Some Problems Of Greedy Method**  
**Course Code : ICT-2108**  
**Course Title : Algorithm Design and Analysis Lab Course Credit : 1.00**

|  |  |
| --- | --- |
| Submitted by, **Name :Md Zayed Iqbal**  **ID : IT-16038**  **Batch :2ND YEAR 1st  SEMESTER**  **Dept. Of INFORMATION AND COMMUNICATION TECHNOLOGY** | Submitted to, **MOHAMMAD BADRUL ALAM MIAH**  **Associate PROFESSOR**  **Dept. Of INFORMATION AND COMMUNICATION TECHNOLOGY**  **MAWLANA BHASHANI SCIENCE AND TECHNOLOGY UNIVERSITY** |

**Date of performance: 10-10-2017 Date of Submission: 15.10.2017**

Problem no: 01

**Write a program of fractional Knapsack**

Pseudo code:

# include<stdio.h>

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

float x[20], tp = 0;

int i, j, u;

u = capacity;

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

x[i] = 0.0;

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

if (weight[i] > u)

break;

else {

x[i] = 1.0;

tp = tp + profit[i];

u = u - weight[i];

}

}

if (i < n)

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

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

printf("\nThe result vector is:- ");

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

printf("%f\t", x[i]);

printf("\nMaximum profit is:- %f", tp);

}

int main() {

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

int num, i, j;

float ratio[20], temp;

printf("\nEnter the no. of objects:- ");

scanf("%d", &num);

printf("\nEnter the wts and profits of each object:- ");

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

scanf("%f %f", &weight[i], &profit[i]);

}

printf("\nEnter the capacityacity of knapsack:- ");

scanf("%f", &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;

}

}

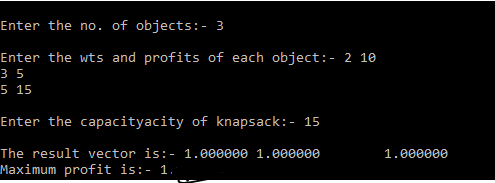
}

knapsack(num, weight, profit, capacity);

return(0);

}

Output:



Program no:02

**A program of MST using Prim’s Algorithm**

Pseudo code:

#include<stdio.h>

#include<conio.h>

int a,b,u,v,n,i,j,ne=1;

int visited[10]= {

0

};

int min;

int mincost=0,cost[10][10];

int main() {

printf("\n Enter the number of nodes:");

scanf("%d",&n);

printf("\n Enter the adjacency matrix:\n");

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

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

scanf("%d",&cost[i][j]);

if(cost[i][j]==0)

cost[i][j]=999;

}

visited[1]=1;

printf("\n");

while(ne<n) {

for (i=1,min=999;i<=n;i++)

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

if(cost[i][j]<min)

if(visited[i]!=0) {

min=cost[i][j];

a=u=i;

b=v=j;

}

if(visited[u]==0 || visited[v]==0) {

printf("\n Edge %d:(%d %d) cost:%d",ne++,a,b,min);

mincost+=min;

visited[b]=1;

}

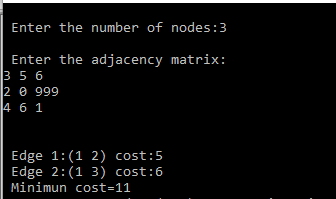
cost[a][b]=cost[b][a]=999;

}

printf("\n Minimun cost=%d",mincost);

getch();

}

Output:

Problem no: 03

**A program to MST using Kruskal**

Pseudo code:

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

#define edge pair<int,int>

class Graph {

private:

vector<pair<int, edge> > G; // graph

vector<pair<int, edge> > T; // mst

int \*parent;

int V; // number of vertices/nodes in graph

public:

Graph(int V);

void AddWeightedEdge(int u, int v, int w);

int find\_set(int i);

void union\_set(int u, int v);

void kruskal();

void print();

};

Graph::Graph(int V) {

parent = new int[V];

//i 0 1 2 3 4 5

//parent[i] 0 1 2 3 4 5

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

parent[i] = i;

G.clear();

T.clear();

}

void Graph::AddWeightedEdge(int u, int v, int w) {

G.push\_back(make\_pair(w, edge(u, v)));

}

int Graph::find\_set(int i) {

// If i is the parent of itself

if (i == parent[i])

return i;

else

// Else if i is not the parent of itself

// Then i is not the representative of his set,

// so we recursively call Find on its parent

return find\_set(parent[i]);

}

void Graph::union\_set(int u, int v) {

parent[u] = parent[v];

}

void Graph::kruskal() {

int i, uRep, vRep;

sort(G.begin(), G.end()); // increasing weight

for (i = 0; i < G.size(); i++) {

uRep = find\_set(G[i].second.first);

vRep = find\_set(G[i].second.second);

if (uRep != vRep) {

T.push\_back(G[i]); // add to tree

union\_set(uRep, vRep);

}

}

}

void Graph::print() {

cout << "Edge :" << " Weight" << endl;

for (int i = 0; i < T.size(); i++) {

cout << T[i].second.first << " - " << T[i].second.second << " : "

<< T[i].first;

cout << endl;

}

}

int main() {

Graph g(6);

g.AddWeightedEdge(0, 1, 4);

g.AddWeightedEdge(0, 2, 4);

g.AddWeightedEdge(1, 2, 2);

g.AddWeightedEdge(1, 0, 4);

g.AddWeightedEdge(2, 0, 4);

g.AddWeightedEdge(2, 1, 2);

g.AddWeightedEdge(2, 3, 3);

g.AddWeightedEdge(2, 5, 2);

g.AddWeightedEdge(2, 4, 4);

g.AddWeightedEdge(3, 2, 3);

g.AddWeightedEdge(3, 4, 3);

g.AddWeightedEdge(4, 2, 4);

g.AddWeightedEdge(4, 3, 3);

g.AddWeightedEdge(5, 2, 2);

g.AddWeightedEdge(5, 4, 3);

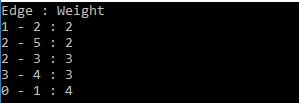
g.kruskal();

g.print();

return 0;

}

Output:



}

}

}

}

fac[q][p] = pow(-1, q + p) \* determinant(b, f - 1);

}

}

transpose(num, fac, f);

}

void transpose(float num[25][25], float fac[25][25], float r)

{

int i, j;

float b[25][25], inverse[25][25], d;

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

{

for (j = 0; j < r; j++)

b[i][j] = fac[j][i];

}

d = determinant(num, r);

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

{

for (j = 0; j < r; j++)

{

inverse[i][j] = b[i][j] / d;

}

}

printf("\n\n\nThe inverse of matrix is : \n");

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

{

for (j = 0; j < r; j++)

{

printf("\t%f", inverse[i][j]);