LAB CYCLE 5

SUBMITTED TO :

FOUSIA MISS

SUBMITTED BY:

NANDANA ANIL

ROLL NO : MCA107

S1 MCA

1.PRIM'S ALGORITHM

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

#define infinity 9999
#define MAX 20

int G[MAX][MAX],spanning[MAX][MAX],n;

int prims();

int main()
{
	int i,j,total_cost;
	printf("Enter no. of vertices:");
	scanf("%d",&n);

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

	for(i=0;i<n;i++)
		for(j=0;j<n;j++)
			scanf("%d",&G[i][j]);

	total_cost=prims();
	printf("\nspanning tree matrix:\n");

	for(i=0;i<n;i++)
	{
		printf("\n");
		for(j=0;j<n;j++)
			printf("%d\t",spanning[i][j]);
	}

	printf("\n\nTotal cost of spanning tree=%d",total_cost);
	return 0;
}

int prims()
{
	int cost[MAX][MAX];
	int u,v,min_distance,distance[MAX],from[MAX];
	int visited[MAX],no_of_edges,i,min_cost,j;

	//create cost[][] matrix,spanning[][]
```

```
for(i=0;i<n;i++)
	for(j=0;j<n;j++)
	{
		if(G[i][j]==0)
			cost[i][j]=infinity;
		else
			cost[i][j]=G[i][j];
			spanning[i][j]=0;
	}

//initialise visited[],distance[] and from[]
distance[0]=0;
visited[0]=1;

for(i=1;i<n;i++)
{
	distance[i]=cost[0][i];
	from[i]=0;
	visited[i]=0;
}

min_cost=0;			//cost of spanning tree
no_of_edges=n-1;		//no. of edges to be added

while(no_of_edges>0)
{
	//find the vertex at minimum distance from the tree
	min_distance=infinity;
	for(i=1;i<n;i++)
		if(visited[i]==0&&distance[i]<min_distance)
		{
			v=i;
			min_distance=distance[i];
		}

	u=from[v];
	spanning[u][v]=distance[v];
	spanning[v][u]=distance[v];
	no_of_edges--;
	visited[v]=1;
	for(i=1;i<n;i++)
		if(visited[i]==0&&cost[i][v]<distance[i])
		{
			distance[i]=cost[i][v];
			from[i]=v;
		}
```

```
                min_cost=min_cost+cost[u][v];
        }

        return(min_cost);
}
```

2.KRUSKAL'S ALGORITHM

```
#include<stdio.h>

#define MAX 30

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

typedef struct edgelist
{
        edge data[MAX];
        int n;
}edgelist;

edgelist elist;

int G[MAX][MAX],n;
edgelist spanlist;

void kruskal();
int find(int belongs[],int vertexno);
void union1(int belongs[],int c1,int c2);
void sort();
void print();

void main()
{
        int i,j,total_cost;

        printf("\nEnter number of vertices:");

        scanf("%d",&n);

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

        for(i=0;i<n;i++)
                for(j=0;j<n;j++)
                        scanf("%d",&G[i][j]);

        kruskal();
        print();
}
```

```
void kruskal()
{
	int belongs[MAX],i,j,cno1,cno2;
	elist.n=0;

	for(i=1;i<n;i++)
		for(j=0;j<i;j++)
		{
			if(G[i][j]!=0)
			{
				elist.data[elist.n].u=i;
				elist.data[elist.n].v=j;
				elist.data[elist.n].w=G[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;
			union1(belongs,cno1,cno2);
		}
	}
}

int find(int belongs[],int vertexno)
{
	return(belongs[vertexno]);
}

void union1(int belongs[],int c1,int c2)
{
```

```
	int i;

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

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;
			}
}

void print()
{
	int i,cost=0;

	for(i=0;i<spanlist.n;i++)
	{
		printf("\n%d\t%d\t%d",spanlist.data[i].u,spanlist.data[i].v,spanlist.data[i].w);
		cost=cost+spanlist.data[i].w;
	}

	printf("\n\nCost of the spanning tree=%d",cost);
}
```

3.SINGLE SOURCE SHORTEST PATH ALGORITHM

```
#include <stdio.h>
#define INFINITY 9999
#define MAX 10
void Dijkstra(int Graph[MAX][MAX], int n, int start);
void Dijkstra(int Graph[MAX][MAX], int n, int start) {
int cost[MAX][MAX], distance[MAX], pred[MAX];
int visited[MAX], count, mindistance, nextnode, i, j;
// Creating cost matrix
for (i = 0; i < n; i++)
for (j = 0; j < n; j++)
if (Graph[i][j] == 0)
cost[i][j] = INFINITY;
else
cost[i][j] = Graph[i][j];
for (i = 0; i < n; i++) {
distance[i] = cost[start][i];
pred[i] = start;
visited[i] = 0;
}
distance[start] = 0;
visited[start] = 1;
count = 1;
while (count < n - 1) {
mindistance = INFINITY;
for (i = 0; i < n; i++)
if (distance[i] < mindistance && !visited[i]) {
mindistance = distance[i];
```

```
nextnode = i;
}
visited[nextnode] = 1;
for (i = 0; i < n; i++)
if (!visited[i])
if (mindistance + cost[nextnode][i] < distance[i]) {
distance[i] = mindistance + cost[nextnode][i];
pred[i] = nextnode;
}
count++;
}
// Printing the distance
for (i = 0; i < n; i++)
if (i != start) {
printf("\nDistance from source to %d: %d", i, distance[i]);
}
}
int main() {
int Graph[MAX][MAX], i, j, n, u;
n = 7;
Graph[0][0] = 0;
Graph[0][1] = 0;
Graph[0][2] = 1;
Graph[0][3] = 2;
Graph[0][4] = 0;
Graph[0][5] = 0;
Graph[0][6] = 0;
Graph[1][0] = 0;
Graph[1][1] = 0;
```

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

```
Graph[5][3] = 0;
Graph[5][4] = 2;
Graph[5][5] = 0;
Graph[5][6] = 1;
Graph[6][0] = 0;
Graph[6][1] = 0;
Graph[6][2] = 0;
Graph[6][3] = 1;
Graph[6][4] = 0;
Graph[6][5] = 1;
Graph[6][6] = 0;
u = 0;
Dijkstra(Graph, n, u);
printf("\n");
return 0;
}
```

LINK TO GITHUB REPOSITORY:

https://github.com/NandanaAnil/Data-Structures.git