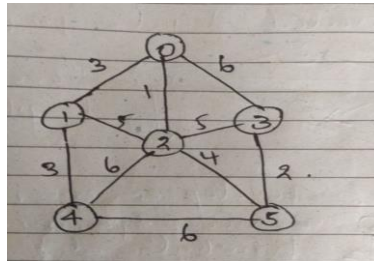


DATA STRUCTURE LAB

SHANA PARWEEN
S1 MCA
TKM20MCA-2036

1. Develop a program to generate a minimum spanning tree using Kruskals algorithm from the given graph and compute total cost.



ALGORITHM:

Algorithm :-
KRUSKAL (G):
1. $A = \phi$
2. For each vertex $v \in V[G]$
3. do MAKE-SET (v)
4. For each edge $(u,v) \in G.E$ ordered by increasing order by weight (u,v)
5. if FIND-SET (u) \neq FIND-SET (v) :
6. then $A \leftarrow A \cup \{(u,v)\}$
7. ~~union~~ UNION (u,v)
8. return A.

PROGRAM CODE:

```
#include<stdio.h>

#include<stdlib.h>

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

int min,mincost=0,cost[9][9],parent[9];

int find(int);

int uni(int,int);

int main()

{

    printf("\n*****Kruskal's algorithm*****\n");

    printf("\nEnter the no. of vertices:");
```

```

scanf("%d",&n);

printf("\nAdjacency 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;

    }

}

printf("Edges of Minimum Cost Spanning Tree:\n");

while(m < n)

{

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

    {

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

        {

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

            {

                min=cost[i][j];

                a=u=i;

                b=v=j;

            }

        }

    }

    u=find(u);

    v=find(v);

```

```

        if(uni(u,v))
        {
            printf("Edge (%d,%d) =%d\n",m++,a,b,min);

            mincost +=min;
        }

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

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

int find(int i)
{
    while(parent[i])

        i=parent[i];

    return i;
}

int uni(int i,int j)
{
    if(i!=j)
    {
        parent[j]=i;

        return 1;
    }

    return 0;
}

```

OUTPUT:

```

*****Kruskal's algorithm*****
Enter the no. of vertices: 5
Adjacency matrix:
0 3 1 0 0
3 0 5 0 3
1 0 5 6 4
0 5 0 0 2
0 3 0 0 6
0 4 2 0 0
Edges of Minimum Cost Spanning Tree:
Edge (1,1) = 3
Edge (2,4) = 6
Edge (3,1) = 2
Edge (4,2) = 5
Edge (5,3) = 6
Minimum cost = 13
(program exited with code: 0)
press return to continue

```

2. Develop a program to implement BFS and DFS.

ALGORITHM:

BFS():

1. Create a queue Q.
2. Mark v as visited and put v into Q.
3. While Q is non-empty
4. remove the head u of Q.
5. mark and enqueue all unvisited neighbours of u.

DFS:

DFS(G, u)

u-visited = true

for each v ∈ G.Adj[u]

if v-visited == false

DFS(G, v)

Unit 1 {

For each u ∈ G

u-visited = false

For each u ∈ G

DFS(G, u)

}

PROGRAM CODE:

1. BFS:-

```

#include<stdio.h>
int a[20][20], q[20], visited[20], n, i, j, f = 0, r = -1;

```

```

void bfs(int v)
{
    for(i = 1; i <= n; i++)
        if(a[v][i] && !visited[i])
            q[++r] = i;
            if(f <= r) {
                visited[q[f]] = 1;
                bfs(q[f++]);
            }
}

int main()
{
    int v;
    printf("\n Enter the number of vertices:");
    scanf("%d", &n);

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

    printf("\n Enter graph data in matrix form:\n");
    for(i=1; i<=n; i++)
    {
        for(j=1;j<=n;j++)
        {
            scanf("%d", &a[i][j]);
        }
    }

    printf("\n Enter the starting vertex:");
    scanf("%d", &v);
    bfs(v);
    printf("\n The node which are reachable are:\n");

    for(i=1; i <= n; i++)
    {
        if(visited[i])
            printf("%d\t", i);
        else
        {
            printf("\n Bfs is not possible. Not all nodes are reachable");
            break;
        }
    }
}

```

2. DFS:-

```

#include<stdio.h>
int a[20][20],reach[20],n;
int dfs(int v)

```

```

{
    int i;
    reach[v]=1;
    for (i=1;i<=n;i++)
        if(a[v][i] && !reach[i])
        {
            printf("\n %d->%d",v,i);
            dfs(i);
        }
}
}
int main()
{
    int i,j,count=0;
    printf("\n Enter number of vertices:");
    scanf("%d",&n);
    for (i=1;i<=n;i++)
    {
        reach[i]=0;
        for (j=1;j<=n;j++)
            a[i][j]=0;
    }
    printf("\n Enter the adjacency matrix:\n");
    for (i=1;i<=n;i++)
        for (j=1;j<=n;j++)
            scanf("%d",&a[i][j]);
    dfs(1);
    printf("\n");
    for (i=1;i<=n;i++)
    {
        if(reach[i])
            count++;
    }
    if(count==n)
        printf("\n Graph is connected"); else
        printf("\n Graph is not connected");
    return 0;
}

```

OUTPUT:

1. BFS:

```

Applications Places
File Edit View Search Terminal Help
Enter the number of vertices:6
Enter graph data in matrix form:
0 1 1 0 0
1 0 1 0 1 0
1 1 0 1 1 1
1 0 1 0 0 1
0 1 1 0 0 1
0 0 1 1 1 0
Enter the starting vertex:1
The node which are reachable are:
1 2 3 4 5 6
.....
(program exited with code: 0)
Press return to continue

```

2. DFS:

A screenshot of a Linux terminal window. The top bar shows the application name 'Applications - Places' and system status '11:59 AM' with battery level '97%'. The terminal title bar reads 'File Edit View Search Terminal Help'. The program output is as follows:

```
Enter number of vertices:6
Enter the adjacency matrix:
0 1 1 1 0 0
1 0 1 0 1 0
1 0 1 1 1 1
1 0 1 0 0 1
0 1 1 0 0 1
0 0 1 1 1 0

1->2
2->3
3->4
4->6
6->5

Graph is connected

-----
(program exited with code: 0)
Press return to continue
```

The program code is visible in the background, showing a recursive function 'isConnected' that traverses the graph starting from vertex 1. The code includes comments in Hindi and English, and a main function that reads the input and calls the 'isConnected' function.

GIT LINK: