20MCA135-DATA STRUCTURES EXTERNAL LAB EXAMINATION

SUBMITTED BY,

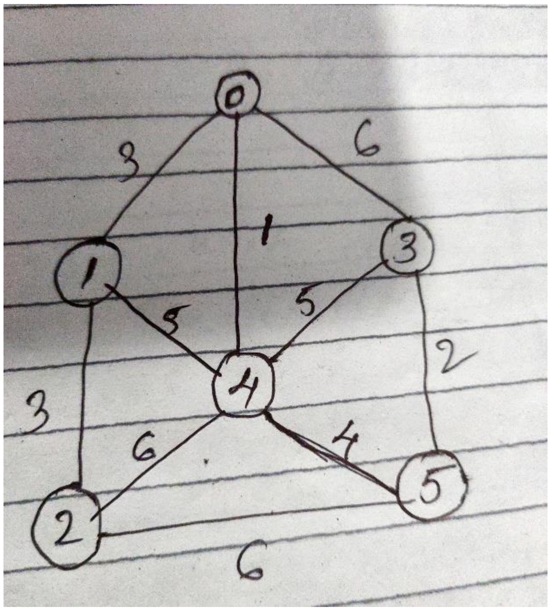
ANANYA B

SEMESTER I

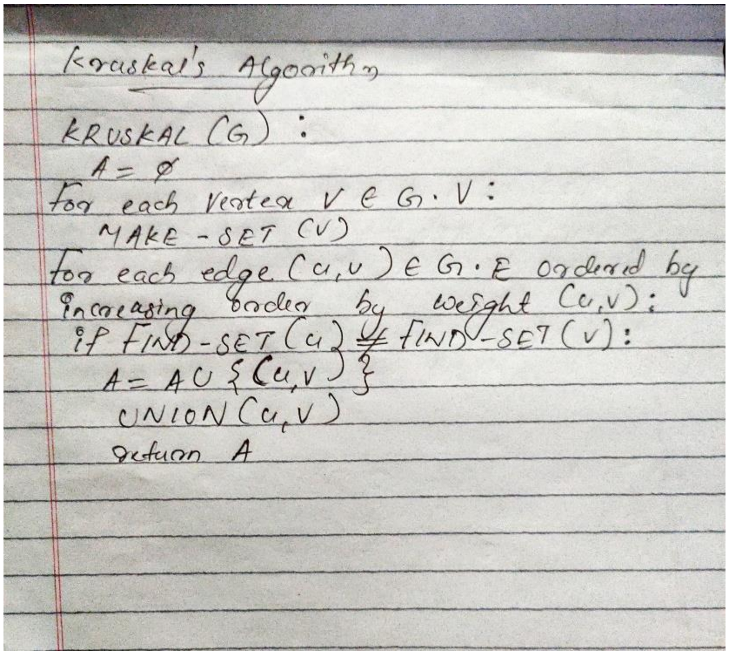
ROLL NO:20MCA208

REG NO: TKM20MCA-2008

QUESTION: Develop a program to generate a minimum cost spanning tree using Kruskal algorithm for the given graph and compute the minimum cost



ALGORITHM:



```
Kruskal's 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)
    return A
```

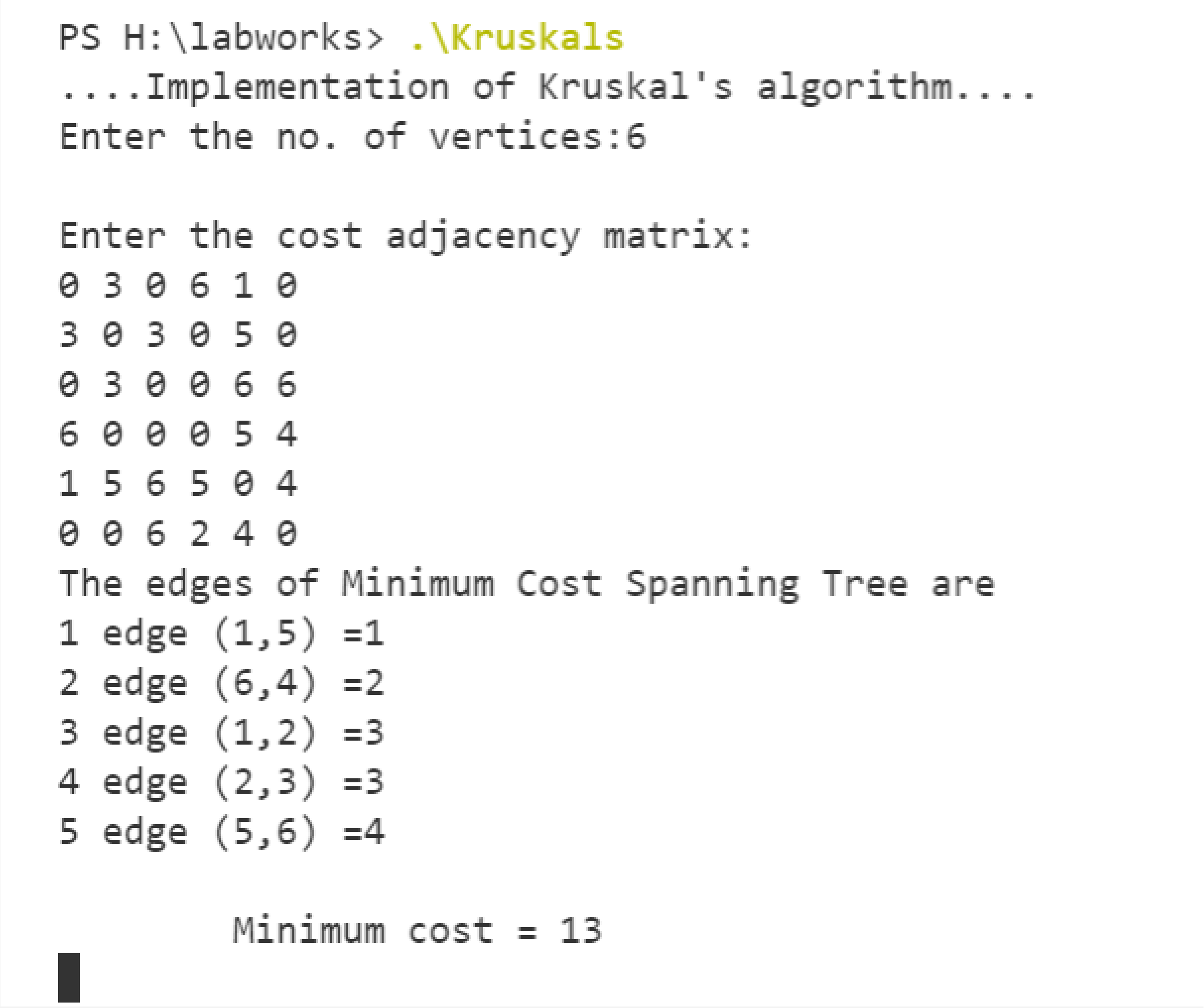
PROGRAM CODE:

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int i,j,k,x,y,u,v,n,nofed=1;
int Min,Mincost=0,cost[9][9],Parent[9];
int Find(int);
int Union(int,int);
void main()
{

    printf("....Implementation of Kruskal's algorithm....");
    printf("\nEnter the no. of vertices:");
    scanf("%d",&n);
    printf("\nEnter the cost 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;
        }
    }
    printf("The edges of Minimum Cost Spanning Tree are\n");
    while(nofed < n)
    {
        for(i=1,Min=999;i<=n;i++)
        {
            for(j=1;j <= n;j++)
            {
                if(cost[i][j] < Min)
                {
                    Min=cost[i][j];
                    x=u=i;
                    y=v=j;
                }
            }
        }
        u=Find(u);
        v=Find(v);
        if(Union(u,v))
        {
            printf("%d edge (%d,%d) =%d\n",nofed++,x,y,Min);
            Mincost +=Min;
        }
        cost[x][y]=cost[y][x]=999;
    }
    printf("\n\tMinimum cost = %d\n",Mincost);
    getch();
}
int Find(int i)
{
    while(Parent[i])
    i=Parent[i];
    return i;
}
int Union(int i,int j)
{
    if(i!=j)
    {
        Parent[j]=i;
        return 1;
    }
    return 0;
}
```

RESULT: The program was successfully executed and obtained the outpu

OUTPUT:



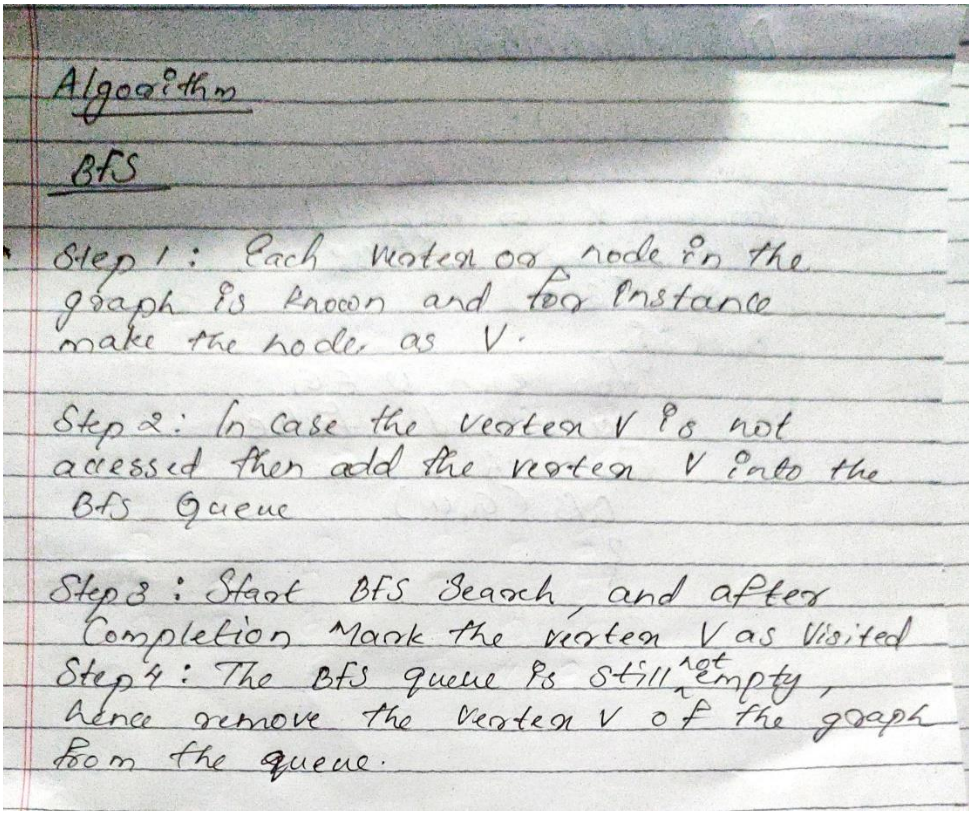
```
PS H:\labworks> .\Kruskals
....Implementation of Kruskal's algorithm....
Enter the no. of vertices:6

Enter the cost adjacency matrix:
0 3 0 6 1 0
3 0 3 0 5 0
0 3 0 0 6 6
6 0 0 0 5 4
1 5 6 5 0 4
0 0 6 2 4 0
The edges of Minimum Cost Spanning Tree are
1 edge (1,5) =1
2 edge (6,4) =2
3 edge (1,2) =3
4 edge (2,3) =3
5 edge (5,6) =4

        Minimum cost = 13
```

QUESTION 2: Develop a program to implement BFS and BFS of above graph

ALGORITHM



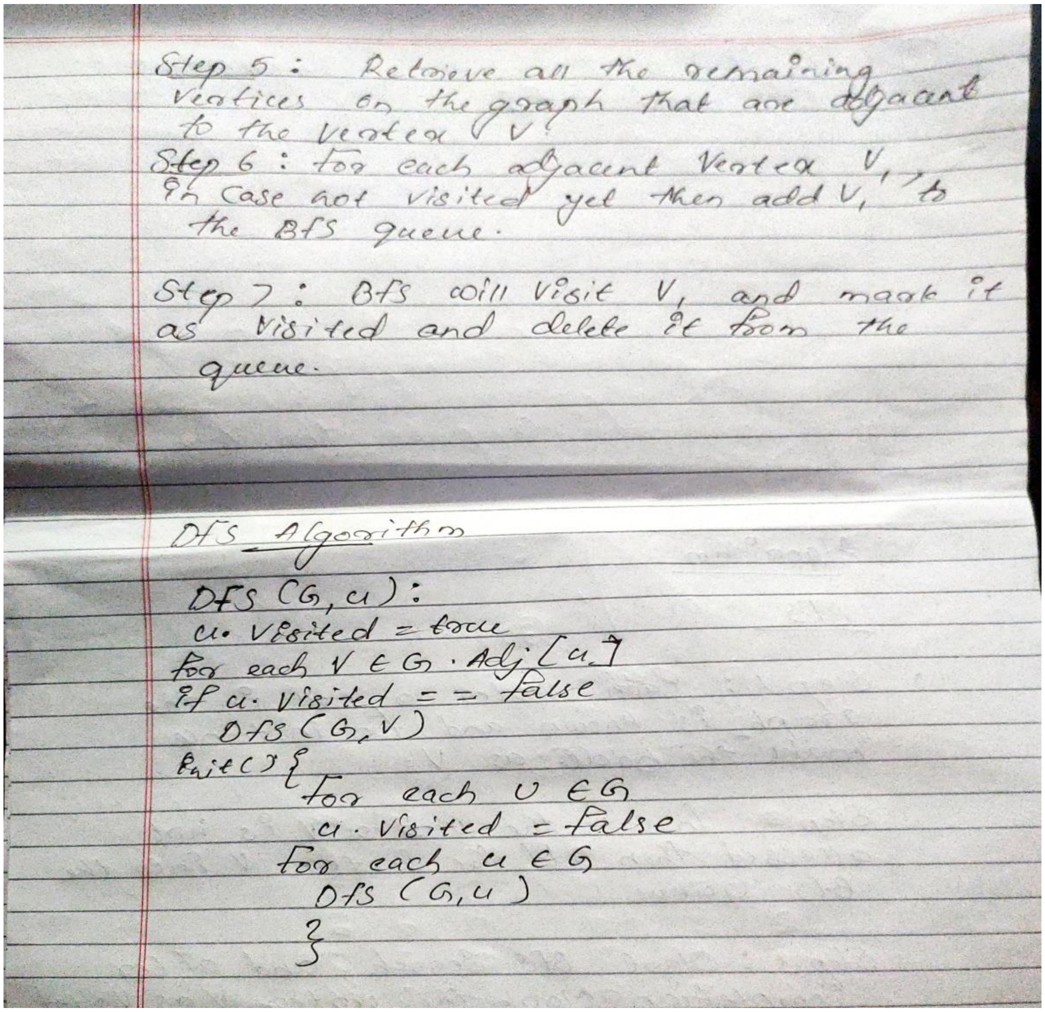
Algorithm

BFS

Step 1: Each vertex or node in the graph is known and for instance make the node as V.

Step 2: In case the vertex V is not accessed then add the vertex V into the BFS Queue

Step 3: Start BFS Search, and after Completion Mark the vertex V as Visited
Step 4: The BFS queue is still not empty, hence remove the vertex V of the graph from the queue.



Step 5 : Retrieve all the remaining vertices on the graph that are adjacent to the vertex 'V'.

Step 6 : for each adjacent vertex V_1, in case not visited yet then add V_1 to the BFS queue.

Step 7 : BFS will visit V_1 and mark it as visited and delete it from the queue.

DFS Algorithm

```
DFS (G, u):
 u. visited = true
for each V ∈ G . Adj [u]
if u. visited == false
   DFS (G, V)
Init() {
     for each U ∈ G
       u. visited = false
     for each u ∈ G
       DFS (G, u)
     }
```

PROGRAM

```
#include<stdio.h>
int top=-1,queue[20],stack[20],front=-1,rear=-1,arr[20][20],visited[20]={0};
void add(int item);
void BFS(int s,int n);
void DFS(int s,int n);
void Push(int item);
int Pop();
int delete();

void main()
{
int i,j,n,choice,s;

printf("Enter the Number of Vertices:");
scanf("%d",&n);

printf("\nEnter adjacency matrix:\n");
    for(i=0;i<n;i++){
        for(j=0;j<n;j++){
            scanf("%d",&arr[i][j]);
        }
    }
printf("Enter Choice 1.BFS 2.DFS");
scanf("%d",&choice);
printf("Enter stating vertex:");
scanf("%d",&s);
while(choice!=3)
{
switch(choice)
{
case 1:BFS(s,n);
break;
case 2:DFS(s,n);
break;
}
printf("\nEnter Choice 1.BFS 2.DFS \n");
scanf("%d",&choice);
for(i=0;i<=n;i++){visited[i]=0;}
}
}

void add(int item)
    {
```

```
    if(rear==19)
      printf("QUEUE IS FULL...");
    else
      {
       if(rear==-1)
          {
            queue[++rear] = item;
            front++;
          }
        else
          queue[++rear]=item;
      }
  }

int delete()
{
  int k;

  if ((front>rear)||(front==-1))
    return (0);
  else
    {
     k=queue[front++];
```

```
            return(k);
        }
}

void Push( int item )
{
    if ( top == 19 )
        printf( "Stack OVERFLOW..... " );
    else
        stack[ ++top ] = item;
}

int Pop()
{
    int k;

    if ( top == -1 )
        return ( 0 );
    else
        {
            k = stack[ top-- ];
            return ( k );
        }
```

```
}

void BFS(int s,int n)
{
int i,p;
add(s);
visited[s]=1;
p=delete();
if(p!=0) printf("%d ",p);
while(p!=0)
{
for(i=1;i<=n;i++)
{
if((arr[p][i]!=0)&&(visited[i]==0))
{
add(i);
visited[i]=1;
}
}
p=delete();
if(p!=0) printf("%d ",p);
}
for(i=1;i<=n;i++)
```

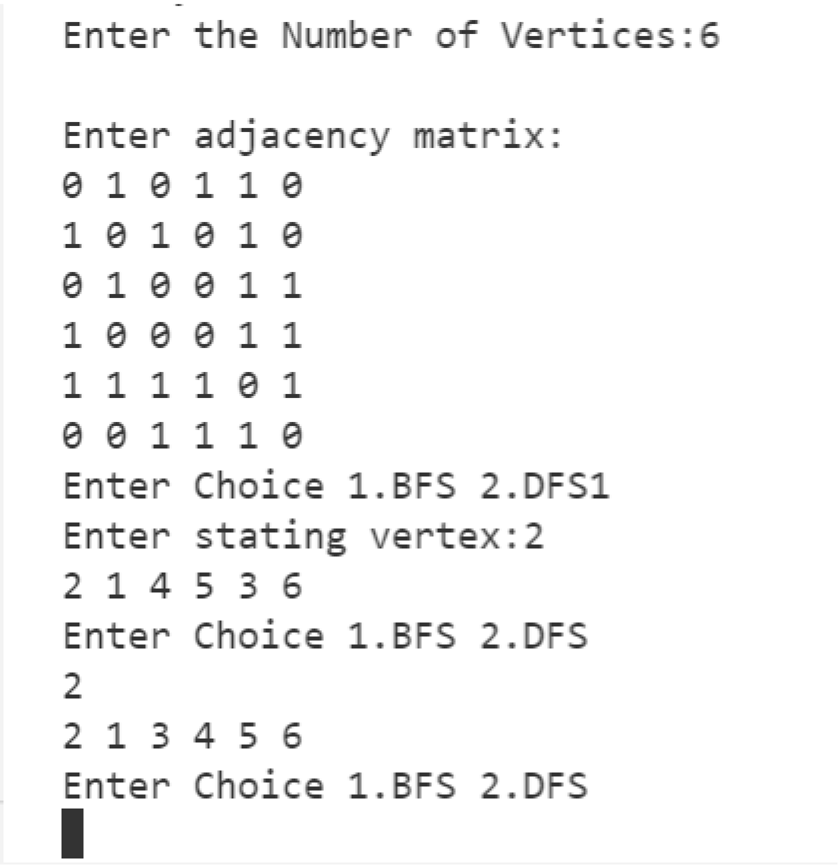
```
{
if(visited[i]==0) BFS(i,n);
}
}

 void DFS(int s,int n)
{
int k,i;
Push(s);
visited[s]=1;
k=Pop();
if(k!=0) printf("%d ",k);

while(k!=0)
{
for(i=1;i<=n;i++)
{
if((arr[k][i]!=0)&&(visited[i]==0))
{
Push(i);
visited[i]=1;
}
k=Pop();
if(k!=0) printf("%d ",k);
}
}
for(i=1;i<=n;i++){
if(visited[i]==0) DFS(i,n);
}
}
```

RESULT: The program is successfully executed and obtained the output

OUTPUT



```
Enter the Number of Vertices:6

Enter adjacency matrix:
0 1 0 1 1 0
1 0 1 0 1 0
0 1 0 0 1 1
1 0 0 0 1 1
1 1 1 1 0 1
0 0 1 1 1 0
Enter Choice 1.BFS 2.DFS1
Enter stating vertex:2
2 1 4 5 3 6
Enter Choice 1.BFS 2.DFS
2
2 1 3 4 5 6
Enter Choice 1.BFS 2.DFS
```

GITHUB LINK: https://github.com/Ananya31-tkm/DATA_STRUCTURE