S1 EXTERNAL LAB EXAMINATION

DATA STRUCTURES LAB

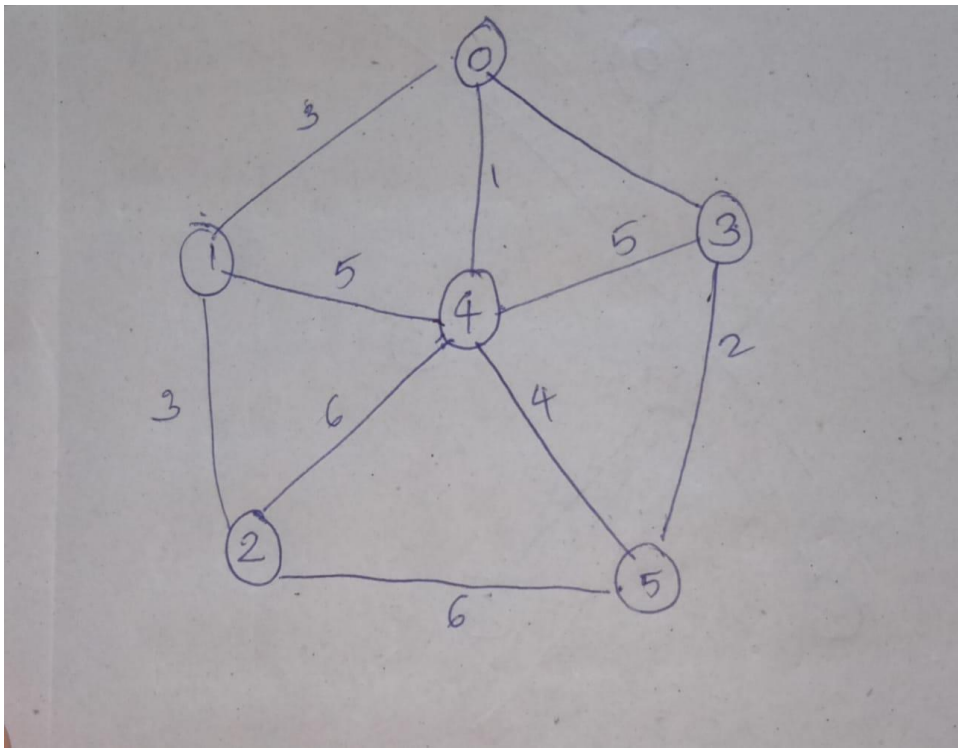
Submitted By

Nandana Anil

Roll No- TKM20MCA2024

QUESTION-1

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



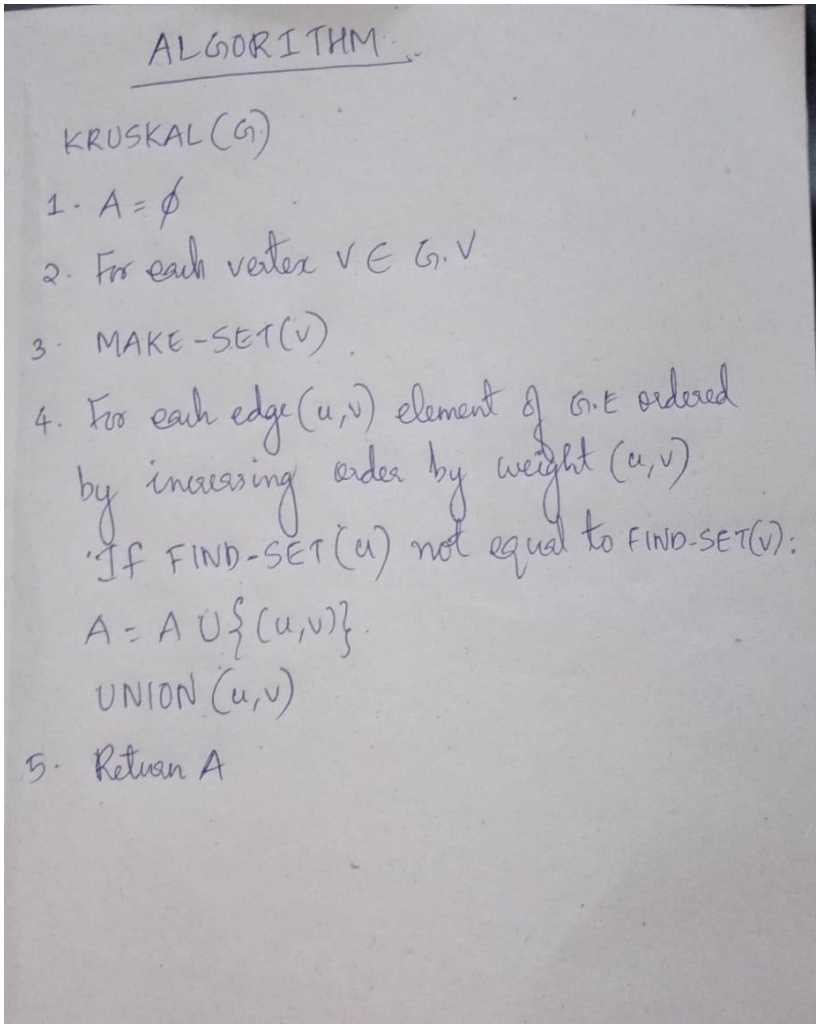
ALGORITHM

KRUSKAL(G)

1. A=phi
2. For each vertex v element-of G.V

   MAKE-SET(v)

3. For each edge(u,v) element of G.E ordered by increasing order by weight (u,v)
   If FIND-SET(u) not equal to FIND-SET(v):
   A=A union {(u,v)}
   UNION(u,v)
4. return A



PROGRAM CODE

```
#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;h

	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);
}
```

OUTPUT

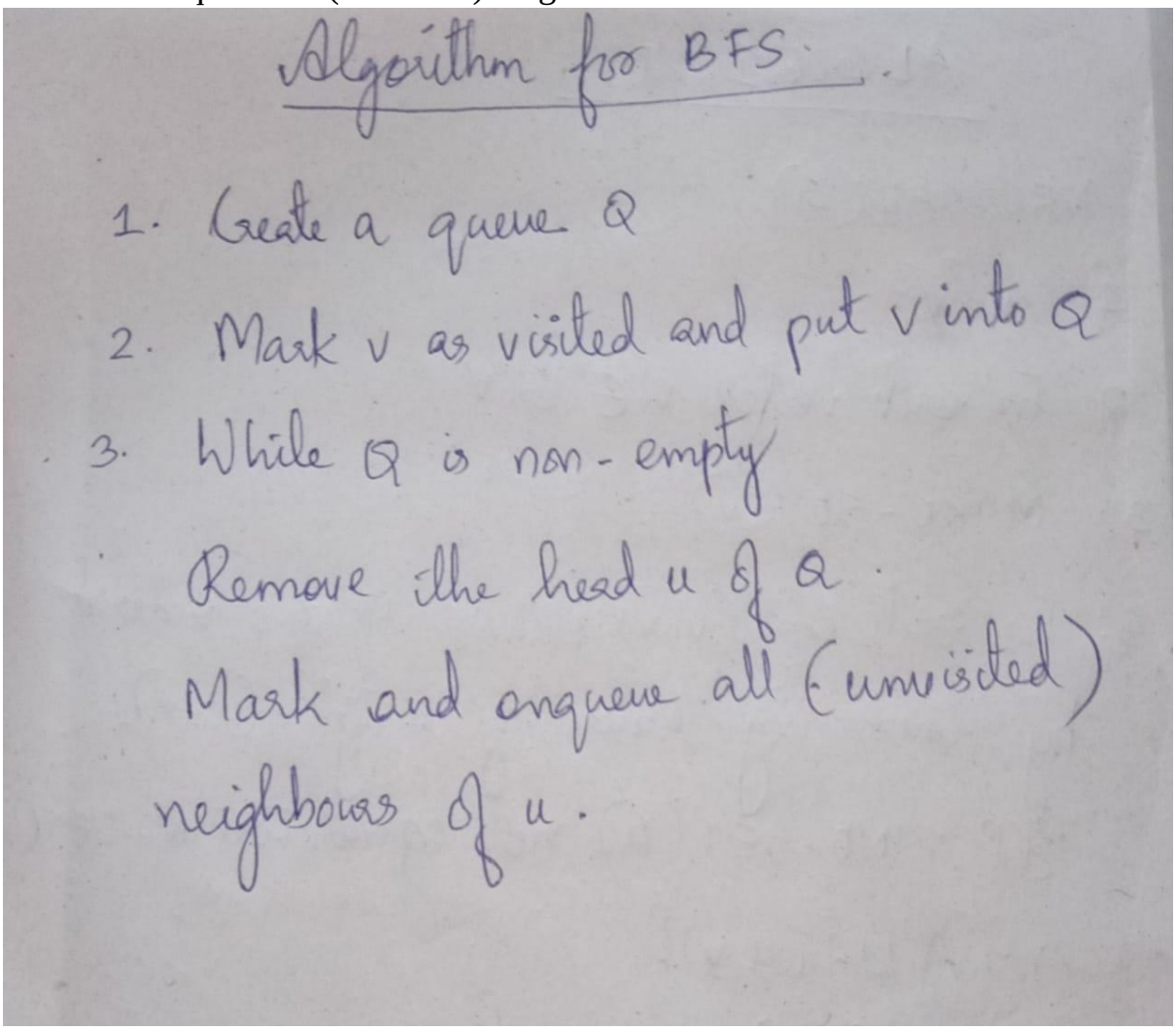
```
codebind@codebind:~$ gcc kruskal.c -o kruskal.out
codebind@codebind:~$ ./kruskal.out

Enter number of vertices:6

Enter the adjacency matrix:
0
3
1
6
0
0
3
0
5
0
3
0
1
5
0
5
6
4
6
0
5
0
0
2
0
3
6
0
0
6
0
0
4
2
6
0

2       0       1
5       3       2
1       0       3
4       1       3
5       2       4

Cost of the spanning tree=13codebind@codebind:~$
```

QUESTION-2

Develop a program to implement DFS and BFS

ALGORITHM FOR BFS

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



PROGRAM CODE FOR BFS

```
#include<stdio.h>

int a[20][20],q[20],visited[20],n,i,j,f=0,r=-1;

void bfs(int v);

int main(){
```

```
        int v;
        printf("Enter the number of vertices:");
        scanf("%d",&n);
        printf("Enter the adjacency matrix:");
        for(i=0;i<n;i++){
            for(j=0;j<n;j++)
                scanf("%d",&a[i][j]);

        }
        printf("Enter the starting vertex:");
        scanf("%d",&v);
        for(i=0;i<n;i++){
              q[i]=0;
               visited[i]=0;
         }
        bfs(v);
        printf("The reachable nodes are:");
        for(i=0;i<n;i++){
           if(visited[i])
                    printf("%d\t",i);

         }
return 0;
}
void bfs(int v){
        for(int i=0;i<n;i++){
                if(a[v][i] && !visited[i])
```

```
                    q[++r]=i;
        }
        if(f<=r){
                visited[q[f]]=1;
                bfs(q[++f]);
         }
}
```

OUTPUT

ALGORITHM FOR DFS

DFS(G,u)

u.visited=true

for each v element of G.Adj[u]

if v.visited == false

DFS(G,v)

Init()

{

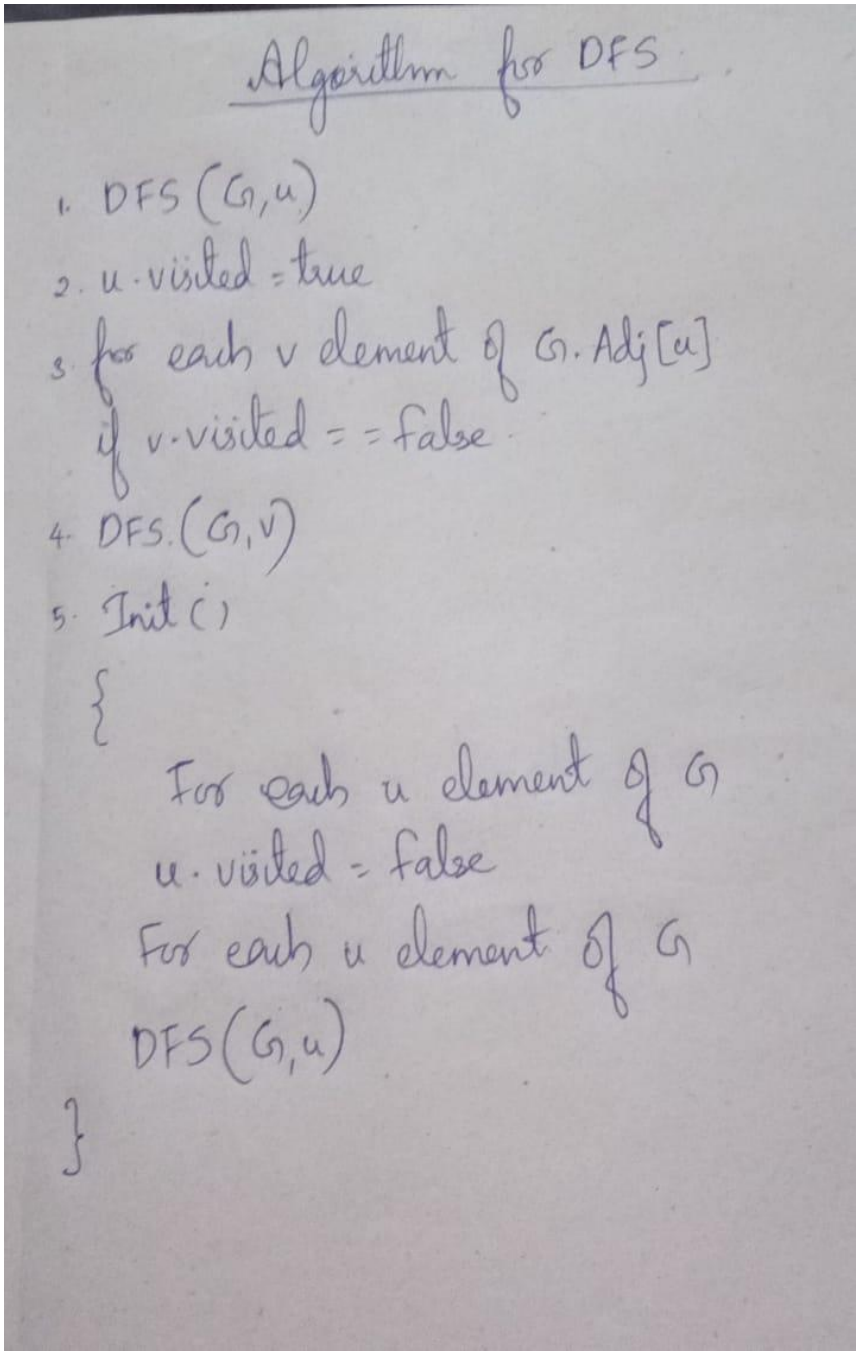
For each u element of G

u.visited = false

For each u element of G

DFS(G,u)

}

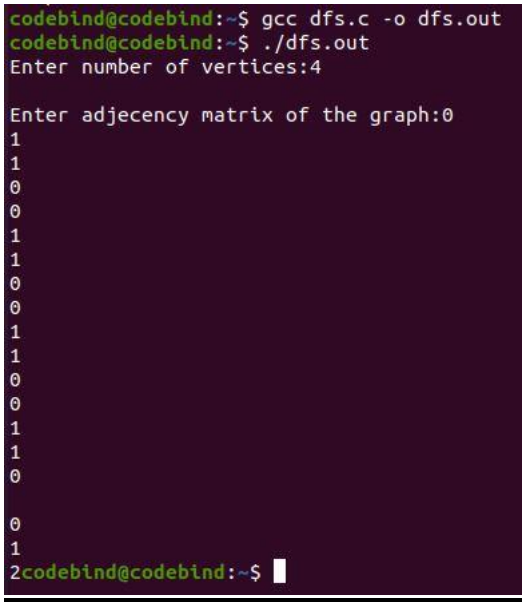

Algorithm for DFS
1. DFS (G,u)
2. u.visited = true
3. for each v element of G.Adj[u]
if v.visited == false
4. DFS.(G,v)
5. Init ()
{
For each u element of G
u.visited = false
For each u element of G
DFS(G,u)
}

PROGRAM CODE FOR DFS

```
#include<stdio.h>

void DFS(int);
int G[10][10],visited[10],n;
void main()
{
        int i,j;
        printf("Enter number of vertices:");
scanf("%d",&n);
printf("\nEnter adjecency matrix of the graph:");

for(i=0;i<n;i++)
                for(j=0;j<n;j++)
scanf("%d",&G[i][j]);
for(i=0;i<n;i++)
                visited[i]=0;
        DFS(0);
}
void DFS(int i)
{
        int j;
printf("\n%d",i);
        visited[i]=1;
for(j=0;j<n;j++)
                if(!visited[j]&&G[i][j]==1)
                                DFS(j);
```

}

OUTPUT



GITHUB LINK:

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