DAY-5 (Data-Structures) Lab Session

EXP - 1

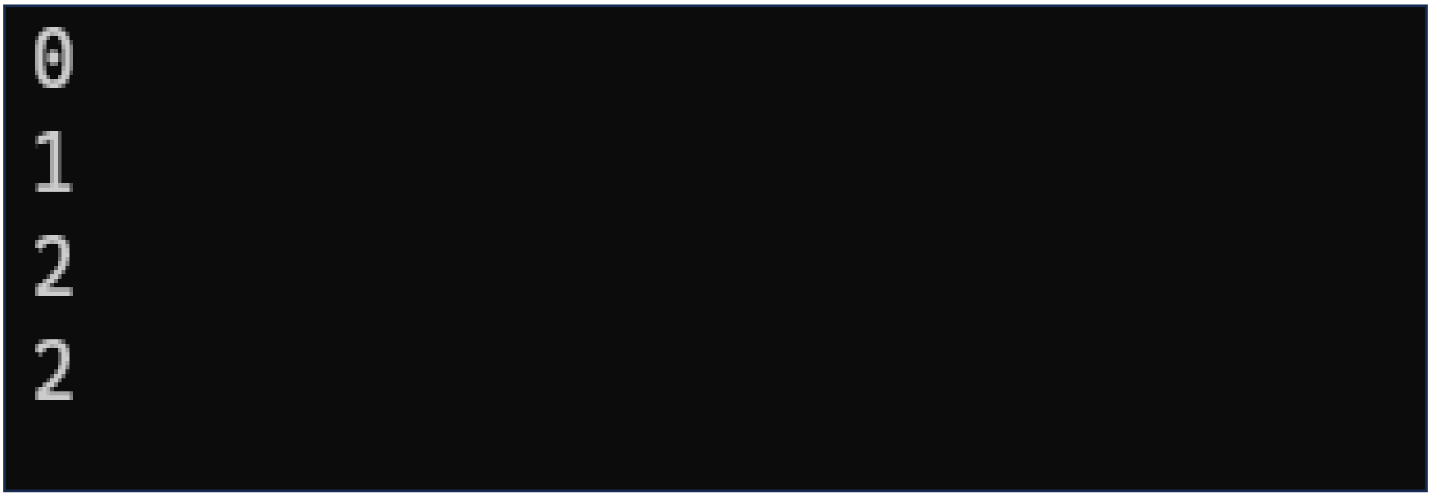
TITLE : C Program to Implement Dijikstra’s Algorithm

PROGRAM :

```
#include <stdio.h>
#include <limits.h>
#define V 4
int dis(int dv[],bool k[])
{
    int min=INT_MAX,minindex;
    for(int i=0;i<V;i++)
    {
        if(dv[i]<=min && k[i]==false)
        {
            min=dv[i];
            minindex=i;
        }
    }
    return minindex;
}
int dijikstra(int a[V][V],int src)
{
    int dv[V];
    bool k[V];
    for(int i=0;i<V;i++)
    {
        dv[i]=INT_MAX;
        dv[src]=0;
        k[i]=false;
    }
    for(int i=0;i<V-1;i++)
    {
        int u=dis(dv,k);
        k[u]=true;
        for(int j=0;j<V;j++)
        {
```

```
if(k[j]==false && dv[u]+a[u][j]<dv[j] && dv[u]!=INT_MAX && a[u][j])
            {
                dv[j]=dv[u]+a[u][j];
            }
        }


    }
    for(int i=0;i<V;i++)
    {
        printf("%d\n",dv[i]);
    }
}
int main()
{
    int a[V][V]={{0,1,2,3},{1,0,1,1},{2,1,0,2},{3,1,2,0}};
    dijikstra(a,0);
}
```

<u>INPUT AND OUTPUT</u> :



```
0
1
2
2
```

<u>RESULT</u> :

The C Program for Implementing Dijikstra's Algorithm is Compiled and Executed Using Dev-C++ and the Output is Verified.

DAY-5 (Data-Structures) Lab Session

EXP - 2

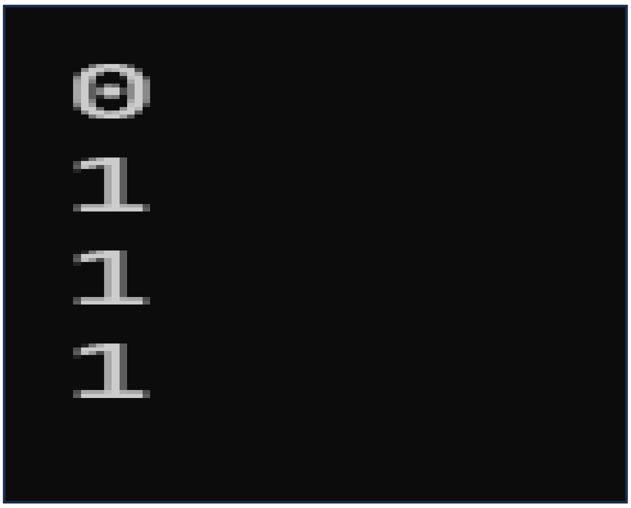
TITLE : C Program to Implement Prims's Algorithm

PROGRAM :

```
#include <stdio.h>
#include <limits.h>
#define V 4
int dis(int dv[],bool k[])
{
        int min=INT_MAX,minindex;
        for(int i=0;i<V;i++)
        {
                if(dv[i]<=min && k[i]==false)
                {
                        min=dv[i];
                        minindex=i;
                }
        }
        return minindex;
}
int prims(int a[V][V],int src)
{
        int dv[V];
        bool k[V];
        for(int i=0;i<V;i++)
        {
                dv[i]=INT_MAX;
                dv[src]=0;
                k[i]=false;
        }
        for(int i=0;i<V-1;i++)
        {
                int u=dis(dv,k);
                k[u]=true;
                for(int j=0;j<V;j++)
                {
```

```
if(k[j]==false && a[u][j]<dv[j] && dv[u]!=INT_MAX && a[u][j])
                        {
                                dv[j]=a[u][j];
                        }
                }


        }
        for(int i=0;i<V;i++)
        {
                printf("%d\n",dv[i]);
        }
}
int main()
{
        int a[V][V]={{0,1,2,3},{1,0,1,1},{2,1,0,2},{3,1,2,0}};
        prims(a,0);
}
```

INPUT AND OUTPUT :



```
0
1
1
1
```

RESULT :

The C Program for Implementing Prims's Algorithm is Compiled and Executed Using Dev-C++ and the Output is Verified.

DAY-5 (Data-Structures) Lab Session

EXP - 3

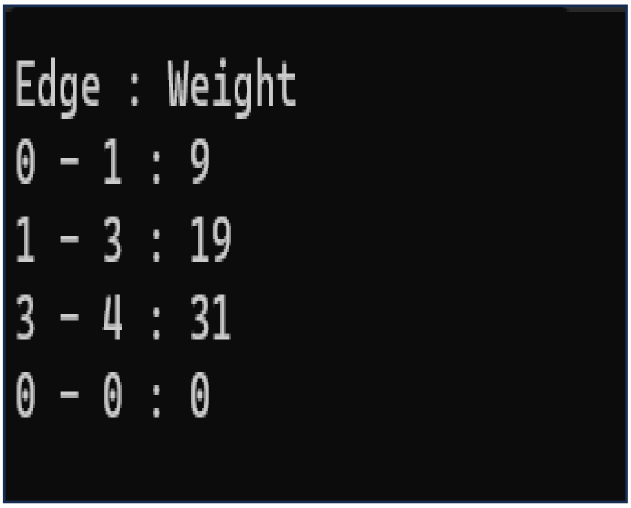
TITLE : C Program to Implement Kruskal's Algorithm

PROGRAM :

```
#include <stdio.h>
#define MAX 30
typedef struct edge {
  int u, v, w;
} edge;
typedef struct edge_list {
  edge data[MAX];
  int n;
} edge_list;

edge_list elist;

int Graph[MAX][MAX], n;
edge_list spanlist;

void kruskalAlgo();
int find(int belongs[], int vertexno);
void applyUnion(int belongs[], int c1, int c2);
void sort();
void print();
void kruskalAlgo() {
  int belongs[MAX], i, j, cno1, cno2;
  elist.n = 0;

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

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

void applyUnion(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 - %d : %d", spanlist.data[i].u, spanlist.data[i].v, spanlist.data[i].w);
    cost = cost + spanlist.data[i].w;
  }

  printf("\nSpanning tree cost: %d", cost);
}
int main() {
  int i, j, total_cost;
  n = 6;

  Graph[0][0] = 0;
  Graph[0][1] = 4;
  Graph[0][2] = 4;
  Graph[0][3] = 0;
  Graph[0][4] = 0;
  Graph[0][5] = 0;
  Graph[0][6] = 0;
Graph[1][0] = 4;
  Graph[1][1] = 0;
  Graph[1][2] = 2;
  Graph[1][3] = 0;
  Graph[1][4] = 0;
  Graph[1][5] = 0;
  Graph[1][6] = 0;
Graph[2][0] = 4;
  Graph[2][1] = 2;
  Graph[2][2] = 0;
  Graph[2][3] = 3;
  Graph[2][4] = 4;
  Graph[2][5] = 0;
  Graph[2][6] = 0;
Graph[3][0] = 0;
  Graph[3][1] = 0;
  Graph[3][2] = 3;
  Graph[3][3] = 0;
  Graph[3][4] = 3;
  Graph[3][5] = 0;
  Graph[3][6] = 0; Graph[4][0] = 0;
  Graph[4][1] = 0;
  Graph[4][2] = 4;
  Graph[4][3] = 3;
  Graph[4][4] = 0;
  Graph[4][5] = 0;
  Graph[4][6] = 0;Graph[5][0] = 0;
  Graph[5][1] = 0;
  Graph[5][2] = 2;
  Graph[5][3] = 0;
  Graph[5][4] = 3;
  Graph[5][5] = 0;
  Graph[5][6] = 0; kruskalAlgo();
  print();
}
```

INPUT AND OUTPUT :



```
Edge : Weight
0 - 1 : 9
1 - 3 : 19
3 - 4 : 31
0 - 0 : 0
```

RESULT :

The C Program for Implementing kruskal’s Algorithm is Compiled and Executed Using Dev-C++ and the Output is Verified.

DAY-5 (Data-Structures) Lab Session

EXP – 4&5

TITLE : C Program to Implement BFS and DFS.

PROGRAM :

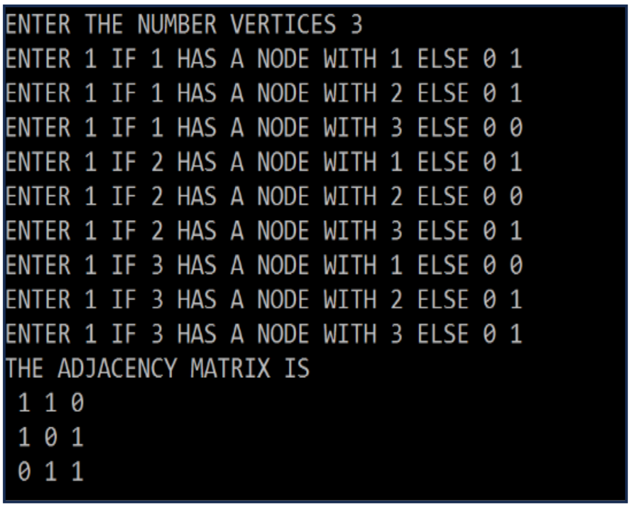
```
#include<stdio.h>


int q[20],top=-1,front=-1,rear=-1,a[20][20],vis[20],stack[20];
int delete();
void add(int item);
void bfs(int s,int n);
void dfs(int s,int n);
void push(int item);
int pop();


void main()
{
int n,i,s,ch,j;
char c,dummy;
printf("ENTER THE NUMBER VERTICES ");
scanf("%d",&n);
for(i=1;i<=n;i++)
{
for(j=1;j<=n;j++)
{
printf("ENTER 1 IF %d HAS A NODE WITH %d ELSE 0 ",i,j);
scanf("%d",&a[i][j]);
}
}
printf("THE ADJACENCY MATRIX IS\n");
for(i=1;i<=n;i++)
{
for(j=1;j<=n;j++)
{
printf(" %d",a[i][j]);


                                  }                  {
```

```
do
{
for(i=1;i<=n;i++)
vis[i]=0;
printf("\nMENU");
printf("\n1.B.F.S");
printf("\n2.D.F.S");
printf("\nENTER YOUR CHOICE");
scanf("%d",&ch);
printf("ENTER THE SOURCE VERTEX :");
scanf("%d",&s);

switch(ch)
{
case 1:bfs(s,n);
break;
case 2:
dfs(s,n);
break;
}
printf("DO U WANT TO CONTINUE(Y/N) ? ");
scanf("%c",&dummy);
scanf("%c",&c);
}while((c=='y')||(c=='Y'));
}
void bfs(int s,int n)
{
int p,i;
add(s);
vis[s]=1;
p=delete();
if(p!=0)
printf(" %d",p);
while(p!=0)
{
```

```
for(i=1;i<=n;i++)
if(vis[i]==0)
bfs(i,n);
}



void add(int item)
{
if(rear==19)
printf("QUEUE FULL");
else
{
if(rear==-1)
{
q[++rear]=item;
front++;
}
else
q[++rear]=item;
}
}
int delete()
{
int k;
if((front>rear)||(front==-1))
return(0);
else
{
k=q[front++];
return(k);
}
}
```

```
void dfs(int s,int n)
{
int i,k;
push(s);
vis[s]=1;
k=pop();
if(k!=0)
printf(" %d ",k);
while(k!=0)
{
for(i=1;i<=n;i++)
if((a[k][i]!=0)&&(vis[i]==0))
{
push(i);
vis[i]=1;
}
k=pop();
if(k!=0)
printf(" %d ",k);
}
for(i=1;i<=n;i++)
if(vis[i]==0)
dfs(i,n);
}
void push(int item)
{
if(top==19)
printf("Stack overflow ");
else
stack[++top]=item;
}
int pop()
{
int k;
if(top==-1)
return(0);}}
```

INPUT AND OUTPUT :



```
ENTER THE NUMBER VERTICES 3
ENTER 1 IF 1 HAS A NODE WITH 1 ELSE 0 1
ENTER 1 IF 1 HAS A NODE WITH 2 ELSE 0 1
ENTER 1 IF 1 HAS A NODE WITH 3 ELSE 0 0
ENTER 1 IF 2 HAS A NODE WITH 1 ELSE 0 1
ENTER 1 IF 2 HAS A NODE WITH 2 ELSE 0 0
ENTER 1 IF 2 HAS A NODE WITH 3 ELSE 0 1
ENTER 1 IF 3 HAS A NODE WITH 1 ELSE 0 0
ENTER 1 IF 3 HAS A NODE WITH 2 ELSE 0 1
ENTER 1 IF 3 HAS A NODE WITH 3 ELSE 0 1
THE ADJACENCY MATRIX IS
 1 1 0
 1 0 1
 0 1 1
```

RESULT :

The C Program for Implementing BFS and DFS is Compiled and Executed Using Dev-C++ and the Output is Verified.