Experiment 18 **Date:15/12/2023**

Graph Traversal

Aim:

Write a program to implement BFS and DFS on a connected undirected graph

Algorithm:

main()

1.start
2. Declare variables n, i, s, ch, j, c, dummy,a[20][20],vis[20]
3. input n
4. for i = 1 to n do
    for j = 1 to n do
        input a[i][j]
5.repeat step 6,7,8 till ch not equal to 'n'
6.for i=0 to n do
    vis[i]=0
7.input ch
  a.if(ch==1)then
    {
      Call bfs(s,n)
    }
  b.if(ch==2)then
    {
      Call b=dfs(s,n)
    }
8.input ch
9.stop

void bfs(int s,int n)

1.start
2.declare p,i
3.call enqueue(s)
4.set vis[s]=1
5.p=dequeue()
6.if(p!=0)then
    print p
7.while(p!=0)do
  {
    for i = 1 to n do
    {
      if((a[p][i]!=0)&&(vis[i]==0))then
      {

```
                        call enqueue(i)
                        set vis[i]=1
                    }
            set p=dequeue()
            if(p!=0)then
               print p

      }
    8.for i = 1 to n do
            if(vis[i]==0)then
                    call bfs(i,n)
    9.exit
```

void enqueue(int item)

```
    1.start
    2.if(rear==19)then
            print "QUEUE FULL"
      else then
      {

            if(rear==-1)then
            {
                    set q[++rear]=item
                    set front++
            }
            else then
                    set q[++rear]=item
      }
     3.exit
```

int dequeue()

```
    1.start
    2.declare k
    3.if((front>rear)||(front==-1))then
            return(0)
    else then
    {
            set k=q[front++]
            return(k)
    }
    4.exit
```

void dfs(int s,int n)

```
    1.start
    2.declare i,k
    3.call push(s)
    4.set vis[s]=1
```

```
5.set k=pop()
6.if(k!=0)then
        print k
7.while(k!=0)do
  {
        for i=1 to n do
        {
              if((a[k][i]!=0)&&(vis[i]==0))then
              {
                    call push(i)
                     vis[i]=1
              }
        }
        set k=pop()
        if(k!=0)then
            print k
}
8.for i = 1 to n do
  {
        if(vis[i]==0)then
             call dfs(i,n)
 }
9.exit
```

void push(int item)

```
1.start
2.if(top==19)then
        print "Stack overflow"
  else then
        set stack[++top]=item
3.exit
```

int pop()

```
1.start
2.declare k
3.if(top==-1)then
        return(0)
   else then
   {
        set k=stack[top--]
        return(k)
   }
 4.exit
```

Program

```
#include<stdio.h>
int q[20],top=-1,front=-1,rear=-1,a[20][20],vis[20],stack[20];
```

```
int dequeue();
void enqueue(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]);
}
printf("\n");
}
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",&c);
}while((c=='y')||(c=='Y'));
}

//***************BFS(breadth-first search) code***************//
void bfs(int s,int n)
{
int p,i;

enqueue(s);
vis[s]=1;
p=dequeue();
if(p!=0)
printf(" %d",p);
while(p!=0)
{
for(i=1;i<=n;i++){
if((a[p][i]!=0)&&(vis[i]==0))
{
enqueue(i);
vis[i]=1;
}}
p=dequeue();
if(p!=0)
printf(" %d ",p);
}
for(i=1;i<=n;i++){
if(vis[i]==0)
bfs(i,n);}
}
void enqueue(int item)
{

if(rear==19)

        printf("QUEUE FULL");

else {
if(rear==-1)
{
        q[++rear]=item;


        front++;
}
else
```

```
        q[++rear]=item;
}
}
int dequeue()
{
int k;
if((front>rear)||(front==-1))
return(0);
else
{
k=q[front++];
return(k);
}
}

//****************DFS(depth-first search) code******************//
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);
else
{
k=stack[top--];
return(k);
}
}
```

Output

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

MENU
1.B.F.S
2.D.F.S
ENTER YOUR CHOICE1
ENTER THE SOURCE VERTEX :1
1 2  3 DO U WANT TO CONTINUE(Y/N) ? y

MENU
1.B.F.S
2.D.F.S
ENTER YOUR CHOICE2
ENTER THE SOURCE VERTEX :1
1  3  2 DO U WANT TO CONTINUE(Y/N) ? n
```

Experiment 19 **Date:20/12/2023**

Prim's Algorithm

Aim:

Program to implement Prim's Algorithm for finding the minimum cost spanning tree.

Algorithm:

1.start

2. declare and initialise
vertex_array[MAX],counter,vertex_count=0,row,column,cost_matrix[MAX][MAX],visited[MAX]={0},edge_count=0,count=1,sum_cost=0,min_cost=0,row_no,column_no,vertex1,vertex2
3.input vertex_count
4. for i= 1 to vertex_count do
    input vertex_array[counter]
5.for row=1 to vertex_count do
  {
    for coloumn=1 to vertex_count do
    {
      input cost_matrix[row][column]
      if(cost_matrix[row][column] == 0)then
      {
        set cost_matrix[row][column] = 999
      }
    }
  }
6.set visited[1]=1
7.set edge_count = vertex_count-1
8.while(count <= edge_count) do
  {
   for min_cost=999,row=1 to vertex_count do{
     for column=1 to vertex_count do{
       if(cost_matrix[row][column] < min_cost) then{
         if(visited[row] != 0) then{
           set min_cost = cost_matrix[row][column]
           set vertex1 = row_no = row
           set vertex2 = column_no = column}
         }
       }
    }
  }
9.if(visited[row_no] == 0 || visited[column_no] ==0) then
  {
    print count++,vertex_array[vertex1],vertex_array[vertex2],min_cost)
    set sum_cost = sum_cost + min_cost

```
            set visited[column_no]=1
            set cost_matrix[vertex1][vertex2] = cost_matrix[vertex2][vertex1] = 999
        }
 10.print sum_cost
 11.stop
```

Program

```
#include<stdio.h>
#define MAX 10
int main(){
        int vertex_array[MAX],counter;
        int vertex_count=0;
        int row,column;
        int cost_matrix[MAX][MAX];
        int visited[MAX]={0};
        int edge_count=0,count=1;
        int sum_cost=0,min_cost=0;
        int row_no,column_no,vertex1,vertex2;
        printf("Total no of vertex :: ");
        scanf("%d",&vertex_count);
        printf("\n-- Enter vertex -- \n\n");
        for(counter=1;counter<=vertex_count;counter++){
                printf("vertex[%d] :: ",counter);
                scanf("%d",&vertex_array[counter]);
        }

        printf("\n--- Enter Cost matrix of size %d x %d  ---
\n\n",vertex_count,vertex_count);
        printf("\n\t-- format is --\n");
        for(row=1;row<=vertex_count;row++){
                for(column=1;column<=vertex_count;column++){
                        printf("x ");
                }
                printf("\n");
        }
        printf("\n-- MATRIX --\n\n");
        //Get edge weight matrix from user
        for(row=1;row<=vertex_count;row++){
        for(column=1;column<=vertex_count;column++){
                        scanf("%d",&cost_matrix[row][column]);
                        if(cost_matrix[row][column] == 0){
                                cost_matrix[row][column] = 999;}
                }
        }
```

```
        printf("\n");
        visited[1]=1;
        edge_count = vertex_count-1;
        while(count <= edge_count){
                for(row=1,min_cost=999;row<=vertex_count;row++){
                        for(column=1;column<=vertex_count;column++){
                                if(cost_matrix[row][column] < min_cost){
                                        if(visited[row] != 0){
                                                min_cost = cost_matrix[row][column];
                                                vertex1 = row_no = row;
                                                vertex2 = column_no = column;
                                        }
                                }
                        }
                }

                if(visited[row_no] == 0 || visited[column_no] ==0){
                        printf("\nEdge %d is (%d -> %d) with cost : %d
",count++,vertex_array[vertex1],vertex_array[vertex2],min_cost);
                        sum_cost = sum_cost + min_cost;
                        visited[column_no]=1;
                }
                cost_matrix[vertex1][vertex2] = cost_matrix[vertex2][vertex1] = 999;
        }
        printf("\n\nMinimum cost=%d",sum_cost);
        return 0;
}
```

Output

```
Total no of vertex :: 3

-- Enter vertex --
vertex[1] :: 1
vertex[2] :: 2
vertex[3] :: 3

--- Enter Cost matrix of size 3 x 3  ---
-- format is --

        x x x

        x x x

        x x x
```

```
-- MATRIX --

    0 5 7

    5 0 1

    7 1 0


Edge 1 is (1 -> 2) with cost : 5
Edge 2 is (2 -> 3) with cost : 1
Minimum cost=6
```

Experiment 20 **Date:21/12/2023**

Kruskal's Algorithm

Aim:

Program to implement Kruskal's algorithm..

Algorithm:

main()

```
1.start
2.declare and initialize i,j,k,a,b,u,v,n,ne=1,min,mincost=0,cost[9][9],parent[9]
3.input n
4.for i=1 to n do
  {
        for j=1 to n do
        {
              input cost[i][j]
              if(cost[i][j]==0)then
                 set cost[i][j]=999
        }
  }
5.while(ne<n) do
  {
        for(i=1,min=999;i<=n;i++)
        {
              for(j=1;j<=n;j++)
              {
                if(cost[i][j]<min)
                {
                      set min=cost[i][j]
                      set a=u=i
                      set b=v=j
                }
              }
        }
  set u=find(u)
  set v=find(v)
  if(uni(u,v))then
  {
   print ne++,a,b,min
   set mincost +=min
  }
  set cost[a][b]=cost[b][a]=999;
  }
6.print mincost
7.stop
```

int find(int i)

1.start
2.while(parent[i])do
{
set i=parent[i]
}
3.return i
4.exit

int uni(int i,int j)

1.start
2.if(i!=j)then
{
set parent[j]=i;
return 1
}
3.return 0
4.exit

Program

```
#include<stdio.h>
#include<stdlib.h>
int i,j,k,a,b,u,v,n,ne=1;
int min,mincost=0,cost[9][9],parent[9];
int find(int);
int uni(int,int);
void main()
{
 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("\nThe edges of Minimum Cost Spanning Tree are\n");
while(ne<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("\n%d edge (%d,%d) =%d\n",ne++,a,b,min);
        mincost +=min;
        }
        cost[a][b]=cost[b][a]=999;
        }
         printf("\n\tMinimum 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

```
Enter the no. of vertices:5
Enter the cost adjacency matrix
0 0 3 0 0
0 0 10 4 0
3 10 0 2 6
0 4 2 0 1
0 0 6 1 0
```

The edges of Minimum Cost Spanning Tree are
1 edge (4,5) =1
2 edge (3,4) =2
3 edge (1,3) =3
4 edge (2,4) =4
Minimum cost = 10

Experiment 21 **Date:04/01/2024**

Disjoint Set Operations

Aim:

Program to perform disjoint set operations create union and find.

Algorithm:

```
struct node
{
        declare struct node *rep,struct node *next,data
}
```

main()

1.start

2.declare struct node*heads[50],*tails[50]

3.declare and initialize countRoot=0,choice,x,i,j,y,flag=0

4.repeat step 5 till choice not equal to 5
5.input choice
a.if(choice==1)then
  {
        input x
        if(search(x)==1)then
                print Element already present in the disjoint set DS
        else then
                call makeSet(x)
  }
 b.if(choice==2)then
   {
        for i=0 to i<countRoot do
                print heads[i]->data
   }
c.if(choice==3)then
  {
        input x
        input y
        call unionSets(x,y)
  }
d.if(choice==4)then
  {
        input x
        create a node rep dynamically
        set rep=find(x)
        if(rep==NULL)then
                print Element not present in the DS

```
                else then
                        print rep->data
    }
  6.stop
```

void unionSets(int a,int b)

```
  1.start
  2.declare and initialize i,pos,flag=0,j,struct node *rep1=find(a)
  3.create a node tail2 dynamically
  4. set struct node *rep2=find(b)
  5.if(rep1==NULL||rep2==NULL)then
    {
        print Element not present in the DS
        return;
    }
  6.if(rep1!=rep2)then
   {
        for j=0 to j<countRoot do
        {
        if(heads[j]==rep2)then
        {
                set pos=j
                set flag=1
                set countRoot-=1
                set tail2=tails[j]
                for i=pos to countRoot do
                {
                        set heads[i]=heads[i+1]
                        set tails[i]=tails[i+1]
                }
         }

        if(flag==1)then
                break
        }
        for j=0 to j<countRoot do
        {
        if(heads[j]==rep1)then
        {
                set tails[j]->next=rep2
                set tails[j]=tail2
                break
        }
        }
        while(rep2!=NULL)do
        {
        set rep2->rep=rep1
```

```
            set rep2=rep2->next
        }
        call displaySet(rep1)
        }
        7.exit
```

struct node* find(int a)

```
        1.start
        2.declare i
        3.create a node tmp dynamically
        4.for i=0 to countRoot do
          {
            set tmp=heads[i]
            while(tmp!=NULL)do
            {
                if(tmp->data==a)then
                    return tmp->rep
                tmp=tmp->next
            }
            return NULL
          }
        5.exit
```

void displaySet(struct node *rep)

```
        1.start
        2.while (rep != NULL)do
          {
            print rep->data
            set rep = rep->next
          }
        3.exit
```

int search(int x)

```
        1.start
        2.declare i
        3.create a node tmp dynamically
        4.for i=0 to countRoot do
          {
            set tmp=heads[i]
            if(heads[i]->data==x)then
                return 1
            while(tmp!=NULL)do
            {
                if(tmp->data==x)then
                    return 1
                tmp=tmp->next
            }
```

```
  }
5.return 0
6.exit
```

void makeSet(int x)

```
1.start
2.create a node new dynamically
3.set new->rep=new
4.set new->next=NULL
5.set new->data=x
6.set heads[countRoot]=new
7.set tails[countRoot++]=new
```

Program

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
struct node{
   struct node *rep;
   struct node *next;
   int data;
}*heads[50],*tails[50];
static int countRoot=0;
void makeSet(int x){
      struct node *new=(struct node *)malloc(sizeof(struct node));
      new->rep=new;
      new->next=NULL;
      new->data=x;
      heads[countRoot]=new;
      tails[countRoot++]=new;
}
struct node* find(int a){
      int i;
      struct node *tmp=(struct node *)malloc(sizeof(struct node));
      for(i=0;i<countRoot;i++){
            tmp=heads[i];
            while(tmp!=NULL){
            if(tmp->data==a)
            return tmp->rep;
            tmp=tmp->next;
            }
      }
      return NULL;
}
void unionSets(int a,int b){
      int i,pos,flag=0,j;
```

```
        struct node *tail2=(struct node *)malloc(sizeof(struct node));
        struct node *rep1=find(a);

        struct node *rep2=find(b);
        if(rep1==NULL||rep2==NULL){
                printf("\nElement not present in the DS\n");
                return;
        }
        if(rep1!=rep2){
                for(j=0;j<countRoot;j++){
                        if(heads[j]==rep2){
                                pos=j;
                                flag=1;
                                countRoot-=1;
                                tail2=tails[j];
                                for(i=pos;i<countRoot;i++){
                                        heads[i]=heads[i+1];
                                        tails[i]=tails[i+1];
                                }
                        }
                        if(flag==1)
                                break;
                }
                for(j=0;j<countRoot;j++){
                        if(heads[j]==rep1){
                                tails[j]->next=rep2;
                                tails[j]=tail2;
                                break;
                        }
                }
                while(rep2!=NULL){
                rep2->rep=rep1;
                rep2=rep2->next;
                }
        displaySet(rep1);
        }
 }
 void displaySet(struct node *rep) {
    printf("Unioned Set: ");
    while (rep != NULL) {
        printf("%d ", rep->data);
        rep = rep->next;
    }
    printf("\n");
 }
 int search(int x){
        int i;
        struct node *tmp=(struct node *)malloc(sizeof(struct node));
```

```
	for(i=0;i<countRoot;i++){
		tmp=heads[i];
		if(heads[i]->data==x)
			return 1;
		while(tmp!=NULL){
			if(tmp->data==x)
				return 1;
			tmp=tmp->next;
		}
	}

	return 0;
}
void main(){
int choice,x,i,j,y,flag=0;

	do{
		printf("\n||||||||||||||||||||||||||||||||||||||||||||||||||||||||||\n");
		printf("\n.......MENU.......\n\n1.Make Set\n2.Display set
representatives\n3.Union\n4.Find Set\n5.Exit\n");
		printf("Enter your choice : ");
		scanf("%d",&choice);
		printf("\n||||||||||||||||||||||||||||||||||||||||||||||||||||||||||\n");
		switch(choice){
		case 1:
			printf("\nEnter new element : ");
			scanf("%d",&x);
			if(search(x)==1)
				printf("\nElement already present in the disjoint set DS\n");
			else
				makeSet(x);
			break;
		case 2:
			printf("\n");
			for(i=0;i<countRoot;i++)
				printf("%d ",heads[i]->data);
			printf("\n");
			break;
		case 3:
			printf("\nEnter first element : ");
			scanf("%d",&x);
			printf("\nEnter second element : ");
			scanf("%d",&y);
			unionSets(x,y);
			break;
		case 4:
			printf("\nEnter the element");
			scanf("%d",&x);
```

```
                        struct node *rep=(struct node *)malloc(sizeof(struct node));
                        rep=find(x);
                        if(rep==NULL)
                        printf("\nElement not present in the DS\n");
                        else
                        printf("\nThe representative of %d is %d\n",x,rep->data);
                        break;
                case 5:
                        exit(0);
                default:
                        printf("\nWrong choice\n");

                        break;
                }
        }
        while(1);
}
```

Output

```
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
.......MENU.......

1. Make Set
2. Display set representatives
3. Union
4. Find Set
5. Exit

Enter your choice: 1
Enter new element: 5

||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
.......MENU.......

1. Make Set
2. Display set representatives
3. Union
4. Find Set
5. Exit

Enter your choice: 1
Enter new element: 8
```

```
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
.......MENU.......

1. Make Set
2. Display set representatives
3. Union
4. Find Set
5. Exit

Enter your choice: 2
5 8
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
.......MENU.......

1. Make Set
2. Display set representatives
3. Union
4. Find Set
5. Exit

Enter your choice: 3
Enter first element: 5
Enter second element: 8
Unioned Set:5 8
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
.......MENU.......

1. Make Set
2. Display set representatives
3. Union
4. Find Set
5. Exit

Enter your choice: 4
Enter the element: 8
The representative of 8 is 5

||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
.......MENU.......

1. Make Set
2. Display set representatives
3. Union
4. Find Set
5. Exit

Enter your choice: 5
```

Experiment 22 **Date:05/01/2024**

Dijkstras Algorithm

Aim:

Program for single source shortest path algorithm using Dijkstras algorithm

Algorithm:

main()

1.start
2.declare and initialize INFINITY=9999,MAX=10,G[MAX][MAX],i,j,n,u
3.input n
4.for i=0 to n do
{
for j=0 to j<n do
{
input G[i][j]
}
}

5.input u
6.call dijkstra(G,n,u)
7.stop

void dijkstra(int G[MAX][MAX],int n,int startnode)

1.start
2.declare cost[MAX][MAX],distance[MAX],pred[MAX], visited[MAX],
count,mindistance,nextnode,i,j
3.for i=0 to n do
{
for j=0 to n do
{
if(G[i][j]==0)then
set cost[i][j]=INFINITY
else then
set cost[i][j]=G[i][j]
}
}

4.for i=0 to n do
{
set distance[i]=cost[startnode][i]
set pred[i]=startnode
set visited[i]=0

```
   }
5.set distance[startnode]=0
6.set visited[startnode]=1
7.set count=1
8.while(count<n-1) do
  {
            set mindistance=INFINITY
            for i=0 to n do
            {
            if(distance[i]<mindistance&&!visited[i])then
            {
                  set mindistance=distance[i]
                  set nextnode=i
            }
            set visited[nextnode]=1
            for i=0 to n do
            {
                  if(!visited[i])then
                  {
                        if(mindistance+cost[nextnode][i]<distance[i])then
                        {
                          set distance[i]=mindistance+cost[nextnode][i]
                          set pred[i]=nextnode
                        }
                  }
            }
            count++;
  }

9.for i= 0 to n do
  {
            if(i!=startnode)then
            {
                  print distance[i]
                  print i
                  set j=i

             while(j!=startnode)do
             {
                   set j=pred[j]
                   print j
             }
    }
10.exit
```

Program

```
#include<stdio.h>
#define INFINITY 9999
#define MAX 10

void dijkstra(int G[MAX][MAX],int n,int startnode);

int main() {
	int G[MAX][MAX],i,j,n,u;
	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]);

	printf("\nEnter the starting node:");
	scanf("%d",&u);
	dijkstra(G,n,u);
}

void dijkstra(int G[MAX][MAX],int n,int startnode){
	int cost[MAX][MAX],distance[MAX],pred[MAX];
	int visited[MAX],count,mindistance,nextnode,i,j;

	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];


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

	 distance[startnode]=0;
	 visited[startnode]=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++;
        }


        for(i=0;i<n;i++ )
                 if(i!=startnode){
                    printf("\nDistance of node %d = %d",i,distance[i]);
                    printf("\nPath = %d",i); j=i ;

                  do{
                    j=pred[j];
                     printf(" <- %d",j);
                  }while(j!=startnode);
                }
}
```

Output

```
Enter no. of vertices: 5
Enter the adjacency matrix:
  0 10 5 0 0
  0 0 2 1 0
  0 3 0 9 2
  0 0 0 0 4
  7 0 0 6 0
Enter the starting node: 0
Distance of node 1 = 8
Path = 1 <- 2 <- 0
Distance of node 2 = 5
Path = 2 <- 0
Distance of node 3 = 9
Path = 3 <- 2 <- 0
Distance of node 4 = 7
Path = 4 <- 2 <- 0
```