**ASSIGNMENT NO: 1 DATE:2.3.2022**

**Problem statement:**

Write a program using c to find the shortest path in an unweighted undirected graph. Also check whether the graph is connected or disconnected.

**Theory:**

A graph G(V,E) is said to be undirected if the graph G is defined abstractly as an ordered pair (V,E) where V is a set and E is a set of multisets of two elements from V. An undirected graph can be represented geometrically as a set of marked points V with a set of lines E between the points.

A graph G(V,E) is said to be unweighted if it’s edges don’t have any weight associated with it.

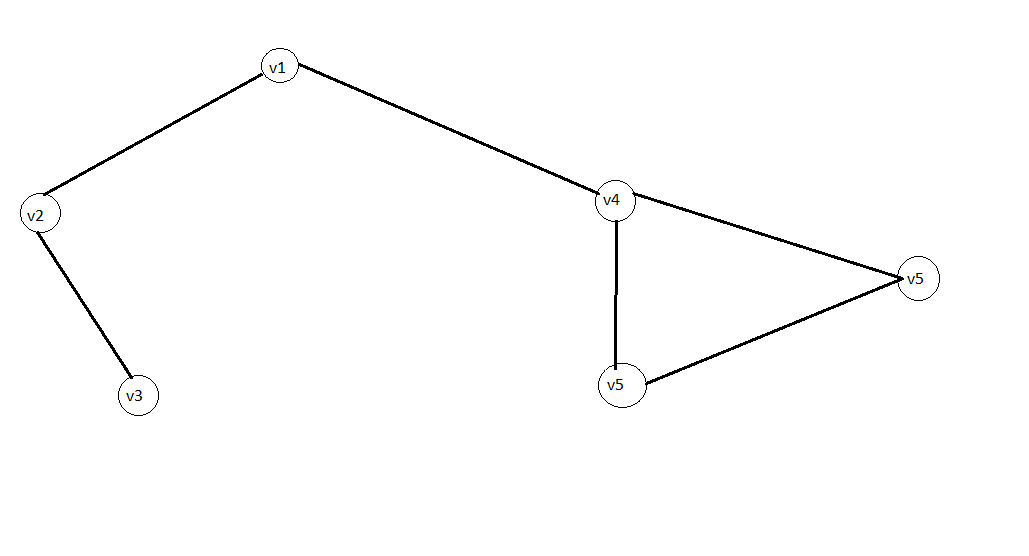
A graph G(V,E) is said to be connected if for every pair of distinct vertices vi,vj in G there is a path and if not then the graph G is said to be disconnected.

Shortest path form source s to destination v can be defined as the minimum number of edges in any path from s to v.

We will solve the given problem using Breadth First Search(BFS) and Depth First Search(DFS) algorithm.

BFS algorithm is one of the simplest algorithm for searching a graph and archetype for many important graph algorithms. Given a graph G=(V,E) and a source vertex s breadth first search systematically explores the edges of G to discover every vertex that is reachable from s. It computes the distance(smallest number of edges) from s to each reachable vertex. In this algorithm we use queue data structure to keep track of next visiting vertex and while we deque a vertex from the queue we increase a count variable by 1, after all the vertices are explored if the value of count is equal to the value of total number of vertices then we can say that our graph is connected else our graph is disconnected.

DFS (Depth-first search) algorithm as its name implies search deeper in the graph whenever possible. DFS algorithm is as same as BFS but instead of using a queue we are using a stack data structure here so that we can search deeper in the graph. Here we will also use a count variable and while we pop a vertex from the stack we will increment it by 1, after all the vertices are explored if the value of count is equal to the value of total number of vertices then we can say that our graph is connected else our graph is disconnected.



The above figure is a undirected unweighted graph

**Algorithms:**

The algorithm for BFS(Breadth first search) is as follows:

Algo\_BFS(G(V,E),s)

{ // G(V,E) is the connected graph given as input where V is the set of vertices and E is

// set of edges.

// s is the source vertex

n =|V|

for(i=1 to n)

{ // initializing the distance array d and parent array p to -1

d[i]= -1

parent[i]= -1

}

d[s]= 0 // initializing the source index of distance array to 0

Q 🡨 an empty queue

count 🡨 0

enqueue(Q,s) // inserting the source to queue

While(Not empty (Q))

{

v 🡨dequeue(Q) // dequeue a element from queue and storing it in v

count= count+1

for all u neighbour(v) // finding neighbours of v and storing in u

{

if( d[u] <0 )

{

d[u] = d[v] +1

parent[u]= v

enqueue(Q,u)

}

}

}

if(count == n)

{

print graph is connected

else

print graph is disconnected

}

// printing the shortest path from source to destination

for(i=1 to n)

{

if( i != s)

{

print i

k=i

while( parent[k] != -1)

{

k = parent[k]

print k

}

}

}

The algorithm for DFS(Depth first search) is as follows:

Algo\_DFS(G(V,E),s)

{ // G(V,E) is the connected graph given as input where V is the set of vertices and E is

// set of edges.

// s is the source vertex

n =|V|

for(i=1 to n)

{ // initializing the distance array d and parent array p to -1

d[i]= -1

parent[i]= -1

}

d[s]= 0 // initializing the source index of distance array to 0

STK🡨 an empty stack

count 🡨 0

Push(STK,s) // inserting the source to stack

While(Not empty (STK))

{

V 🡨pop(STK) // poping a element from queue and storing it in v

count= count+1

for all u neighbour(v) // finding neighbours of v and storing in u

{

if( d[u] <0 )

{

d[u] = d[v] +1

parent[u]= v

push(STK,u)

}

}

}

if(count == n)

{

print graph is connected

else

print graph is disconnected

}

// printing the shortest path from source to destination

for(i=1 to n)

{

if( i != s)

{

print i

k=i

while( parent[k] != -1)

{

k = parent[k]

print k

}

}

}

**Code:**

The code for BFS and DFS is as follows

#include<stdlib.h>

#include<stdio.h>

void bfs(int\*\*,int,int);

void dfs(int\*\*,int,int);

int main()

{

int \*\*a,n,s,d,i,j;

do

{

printf("enter the no of vertices in the graph: ");

scanf("%d",&n);

}while(n<=0);

a=(int\*\*) calloc(n,sizeof (int\*));

for( i=0;i<n;i++)

{

a[i]=(int\*) calloc(n,sizeof (int));

}

printf("\nenter the edges:\n");

do

{

printf("\n\tenter the vertex pair for which there is an edge: ");

scanf("%d%d",&s,&d);

a[s-1][d-1]=1;

a[d-1][s-1]=1;

printf("\n do you want to enter more no edges:[0/1] ");

scanf("%d",&s);

}while(s!=0);

printf("\nthe input graph is:\n");

for(i=0;i<n;i++)

{

printf("\n");

for(j=0;j<n;j++)

{

printf("%d",a[i][j]);

printf("\t");

}

}

do

{

printf("\n\t1.BFS");

printf("\n\t2.DFS");

printf("\n\t3.EXIT");

printf("\n\tenter your choice: ");

scanf("%d",&d);

switch(d)

{

case 1:

printf("\nplease enter the source node: ");

scanf("%d",&s);

s=s-1;

bfs(a,n,s);

break;

case 2:

printf("\nplease enter the source node: ");

scanf("%d",&s);

s=s-1;

dfs(a,n,s);

break;

case 3:

break;

default:

printf("\nenter valid choice: ");

break;

}

} while(d!=3);

free(a);

return 0;

}

void bfs(int\*\*a,int n,int s)

{

int\*d=(int\*) calloc(n,sizeof(int));

int\*parent=(int\*) calloc(n,sizeof(int));

int\*q=(int\*) calloc(n,sizeof(int));

int i,count=0,k,u,v,front=-1,rear=-1;

for(i=0;i<n;i++)

{

d[i]=-1;

parent[i]=-1;

}

d[s]=0;

q[++rear]=s;

while(front!=rear)

{

v=q[++front];

printf("\n%d",v+1);

count++;

for(u=0;u<n;u++)

{

if(a[v][u]==1)

{

if(d[u]<0)

{

d[u]=d[v]+1;

parent[u]=v;

q[++rear]=u;

}

}

}

}

printf("\nthe distance of vertices from source %d: ",(s+1));

for(i=0;i<n;i++)

{

//printf("\t%d",d[i]);

printf("\n distance from node %d to node %d->%d",(s+1),(i+1),d[i]);

}

if(count==n)

{

printf("\n graph is connected\n");

}

else

{

printf("\n graph is disconnected\n");

}

for(i=0;i<n;i++)

{

if(i!=s)

{

printf("\n%d",(i+1));

k=i;

while(parent[k]!=-1)

{

k=parent[k];

printf("->%d",(k+1));

}

printf("\n");

}

}

free(d);

free(q);

free(parent);

}

void dfs(int\*\*a,int n,int s)

{

int\*d=(int\*) calloc(n,sizeof(int));

int\*parent=(int\*) calloc(n,sizeof(int));

int\*stk=(int\*) calloc(n,sizeof(int));

int i,count=0,k,u,v,top=-1;

for(i=0;i<n;i++)

{

d[i]=-1;

parent[i]=-1;

}

d[s]=0;

stk[++top]=s;

while(top!=-1)

{

v=stk[top--];

printf("\n%d",v+1);

count++;

for(u=0;u<n;u++)

{

if(a[v][u]==1)

{

if(d[u]<0)

{

d[u]=d[v]+1;

parent[u]=v;

stk[++top]=u;

}

}

}

}

printf("\nthe distance of vertices from source %d: ",(s+1));

for(i=0;i<n;i++)

{

//printf("\t%d",d[i]);

printf("\n distance from node %d to node %d->%d",(s+1),(i+1),d[i]);

}

if(count==n)

{

printf("\n graph is connected\n");

}

else

{

printf("\n graph is disconnected\n");

}

for(i=0;i<n;i++)

{

if(i!=s)

{

printf("\n%d",(i+1));

k=i;

while(parent[k]!=-1)

{

k=parent[k];

printf("->%d",(k+1));

}

printf("\n");

}

}

free(d);

free(stk);

free(parent);

}

**Output:**

We are taking two graphs one connected and one disconnected. We will apply both of our algorithms on those two graphs for finding the shortest path from a given source and whether the input graph is connected or not

The output of the above code is as follows:  
enter the no of vertices in the graph: 6

enter the edges:

enter the vertex pair for which there is an edge: 1 2

do you want to enter more no edges:[0/1] 1

enter the vertex pair for which there is an edge: 1 4

do you want to enter more no edges:[0/1] 1

enter the vertex pair for which there is an edge: 4 5

do you want to enter more no edges:[0/1] 1

enter the vertex pair for which there is an edge: 4 6

do you want to enter more no edges:[0/1] 1

enter the vertex pair for which there is an edge: 6 5

do you want to enter more no edges:[0/1] 1

enter the vertex pair for which there is an edge: 3 2

do you want to enter more no edges:[0/1] 0

the input graph is:

0 1 0 1 0 0

1 0 1 0 0 0

0 1 0 0 0 0

1 0 0 0 1 1

0 0 0 1 0 1

0 0 0 1 1 0

1.BFS

2.DFS

3.EXIT

enter your choice: 1

please enter the source node: 1

1

2

4

3

5

6

the distance of vertices from source 1:

distance from node 1 to node 1->0

distance from node 1 to node 2->1

distance from node 1 to node 3->2

distance from node 1 to node 4->1

distance from node 1 to node 5->2

distance from node 1 to node 6->2

graph is connected

2->1

3->2->1

4->1

5->4->1

6->4->1

1.BFS

2.DFS

3.EXIT

enter your choice: 2

please enter the source node: 1

1

4

6

5

2

3

the distance of vertices from source 1:

distance from node 1 to node 1->0

distance from node 1 to node 2->1

distance from node 1 to node 3->2

distance from node 1 to node 4->1

distance from node 1 to node 5->2

distance from node 1 to node 6->2

graph is connected

2->1

3->2->1

4->1

5->4->1

6->4->1

1.BFS

2.DFS

3.EXIT

enter your choice: 3

Now we will take a disconnected graph and run our code on it

enter the no of vertices in the graph: 5

enter the edges:

enter the vertex pair for which there is an edge: 1 2

do you want to enter more no edges:[0/1] 1

enter the vertex pair for which there is an edge: 2 3

do you want to enter more no edges:[0/1] 1

enter the vertex pair for which there is an edge: 3 1

do you want to enter more no edges:[0/1] 1

enter the vertex pair for which there is an edge: 4 5

do you want to enter more no edges:[0/1] 0

the input graph is:

0 1 1 0 0

1 0 1 0 0

1 1 0 0 0

0 0 0 0 1

0 0 0 1 0

1.BFS

2.DFS

3.EXIT

enter your choice: 1

please enter the source node: 1

1

2

3

the distance of vertices from source 1:

distance from node 1 to node 1->0

distance from node 1 to node 2->1

distance from node 1 to node 3->1

distance from node 1 to node 4->-1

distance from node 1 to node 5->-1

graph is disconnected

2->1

3->1

4

5

1.BFS

2.DFS

3.EXIT

enter your choice: 2

please enter the source node: 1

1

3

2

the distance of vertices from source 1:

distance from node 1 to node 1->0

distance from node 1 to node 2->1

distance from node 1 to node 3->1

distance from node 1 to node 4->-1

distance from node 1 to node 5->-1

graph is disconnected

2->1

3->1

4

5

1.BFS

2.DFS

3.EXIT

enter your choice: 3

**Discussion:**

* In both of our algorithms time complexity remains the same. The time complexity for DFS and BFS is O(V+E) where V is the number of vertices and E is the number of edges respectively.
* Our program is only applicable for integer type vertices.
* We have used array representation instead of linked representation for queue in BFS and for stack in DFS so wastage of space can happen.