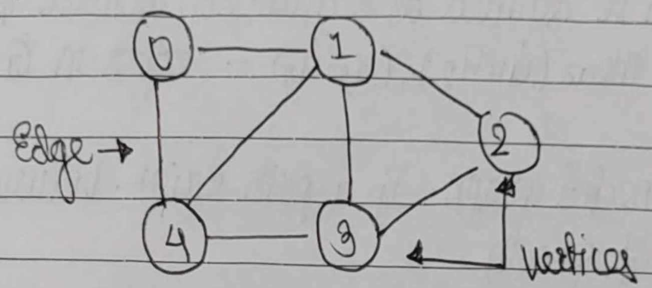
Harsh Kasliwal
2003085
C21

Experiment 9

Aim: Implement Graph traversal methods.

Theory:

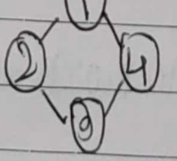
A Graph is a non linear data structure consisting of nodes and edges. The nodes are sometimes also referred to as vertices and the edges are lines or arcs that connect any two nodes in the graph.



Vertices: {0,1,2,3,4}.
Edges: {01, 12, 23, 34, 04, 14, 13}.

- Two nodes are adjacent if they are connected by an edge.
- Graphs can be directed or undirected.

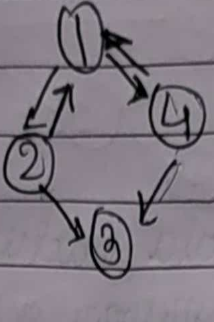
In undirected graph, edges do not have any direction associated with them.
i.e pair (u, v) & (v, u) represents the same edge.
eg.



{12, 23, 34, 41} ✓
{21, 32, 43, 14} ✗ only 4.

Harsh Nagliwal
2003089

In directed graph, edges forms an ordered pair i.e (u,v) and (v,u) are two distinct edges.



Terminologies:

1> Path: can be defined as sequence of vertices ($u_1, u_2 \cdots v$) in such a way that there are (u_1, u_2), (u_2, u_3) ··· edges in G(E).

2> Strongly connected graph: If a path exist. between each pair of vertices of graph.

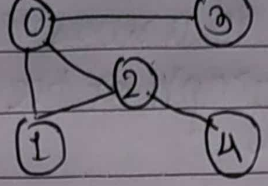
3> Indegree: of a node (u) is no. of edges that terminates at u.
4) Outdegree: of a node (u) is no. of edges that originates at u.
5) Degree: of a node (u), is sum of indegree & outdegree of that nod
6) Length: No. of edges in the path.
7) Cycle: A path that starts and ends at same nodes.

1. BFS (Breadth First Search) : A standar BFS implementation puts each vertex of the graph into one of two categories.
   - Visited
   - Not Visited.

The purpose of the algorithm is to mark each vertex as visited while avoiding cycles.

Harsh Rasliwal.
2003085

Example:



0					Visited

1	2	3			Queue

↑ Front.

start from vertex 0, BFS algorithm starts by putting in visited list & putting all its adjacent vertices in the stack.

Next, we visit the element at the front of queue i.e 1 & go to its adjacent nodes, ~~next we visit 2 instead of 0~~.

0	1				V

2	3				Q

V(2) has an unvisited adjacent vertex in 4, so we add that the back of the queue & visit 3

0	1	2			V

3	4				Q

Similarly.

0	1	2	3	4	V

0					Q

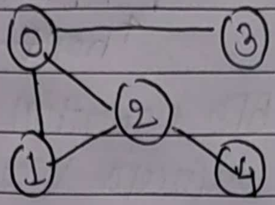
0 is alread visited so queue is empty. BFS completed.

Harsh Kashiwal
2003085
C21

DATE

DFS (Depth First Search) : DFS is a recursive algorithm for searching all the vertices of a graph or tree data structure.

Example :



1> Initially push 0 onto stack as follows

stack : 0.

2> Pop top elemene 0 from stack & push onto stack all adjacent nodes of 0 which is not visited.

stack : 1 2 3

3> Pop top element ① from stack & push all adjacent nodes of ① which is not visited.
stack : 2 3

4> Pop top element ② from stack & push all adjacent nodes which is not visited

stack : 4 3
Similarly :
5> Pop top element ④

stack : 3
6> Pop top element ③

stack : Empty!
visited : 0 1 2 4 3

Harsh Kasliwal
2003085

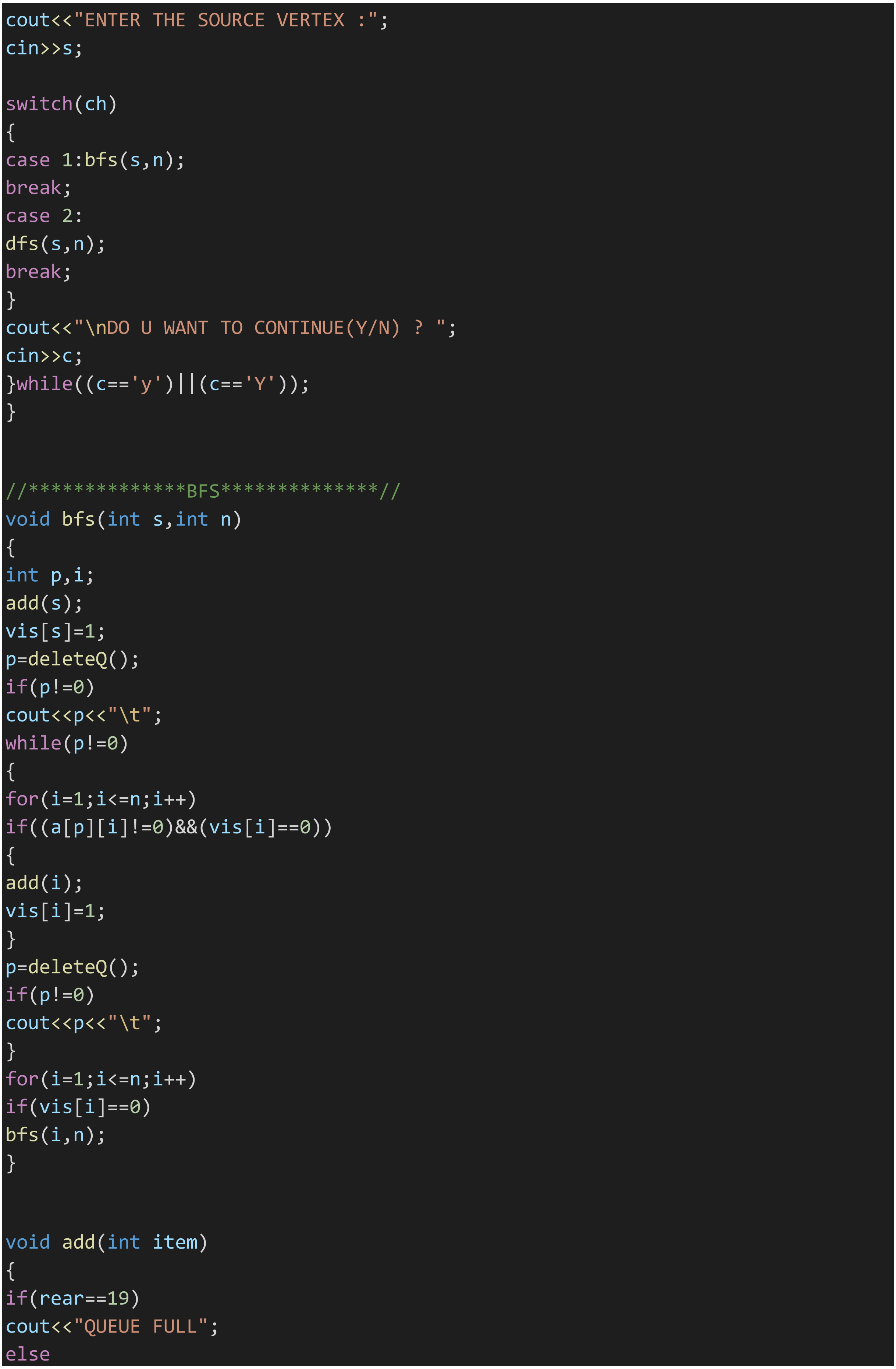
Conclusion: BFS is more suitable for searching vertices which are closer to the given source while DFS is more suitable when there are solutions away from source.

PROGRAM:

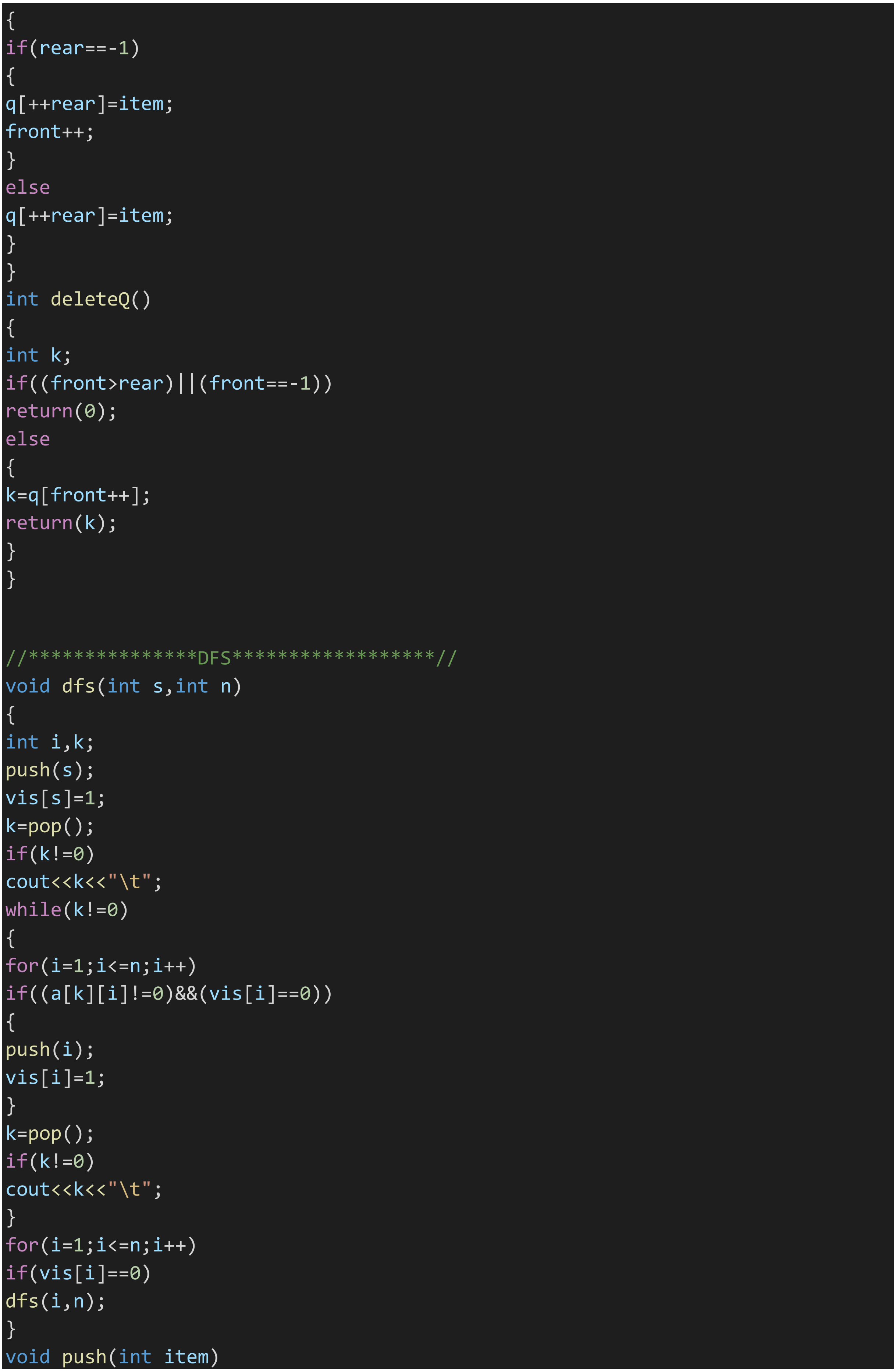
Write a program to implement infix to postfix conversion using stack.

Code:

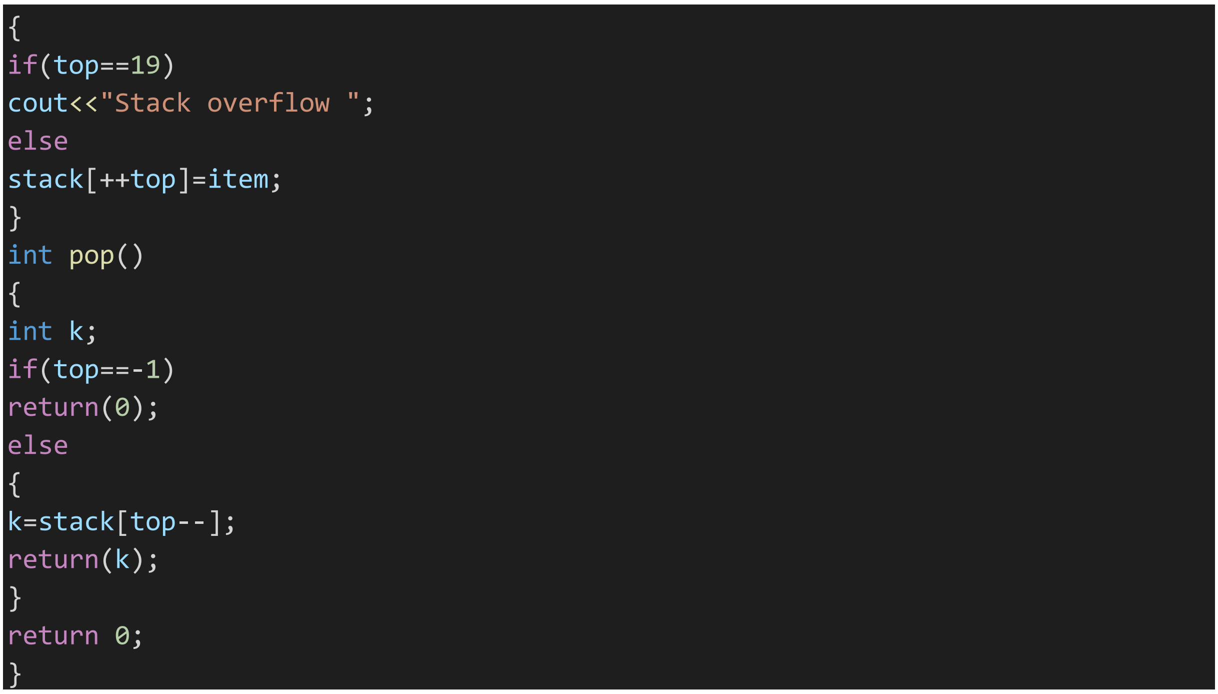
```
#include <iostream>
#include <conio.h>
using namespace std;

int q[20],top=-1,front=-1,rear=-1,a[20][20],vis[20],stack[20];
int deleteQ();
void add(int item);
void bfs(int s,int n);
void dfs(int s,int n);
void push(int item);
int pop();
int main()
{
int n,i,s,ch,j;
char c,;
cout<<"ENTER THE NUMBER VERTICES ";
cin>>n;
for(i=1;i<=n;i++)
{
for(j=1;j<=n;j++)
{
cout<<"ENTER 1 IF "<<i<<" HAS A NODE WITH "<<j<<" ELSE 0 ";
cin>>a[i][j];
}
}
cout<<"THE ADJACENCY MATRIX IS\n";
for(i=1;i<=n;i++)
{
for(j=1;j<=n;j++)
{
cout<<"\t"<<a[i][j];
}
cout<<"\n";
}

do
{
for(i=1;i<=n;i++)
vis[i]=0;
cout<<"\nENTER YOUR CHOICE";
cout<<"\n1.BFS Traversal";
cout<<"\n2.DFS Traversal\n";
cin>>ch;
```



```
cout<<"ENTER THE SOURCE VERTEX :";
cin>>s;

switch(ch)
{
case 1:bfs(s,n);
break;
case 2:
dfs(s,n);
break;
}
cout<<"\nDO U WANT TO CONTINUE(Y/N) ? ";
cin>>c;
}while((c=='y')||(c=='Y'));
}


//**************BFS**************//
void bfs(int s,int n)
{
int p,i;
add(s);
vis[s]=1;
p=deleteQ();
if(p!=0)
cout<<p<<"\t";
while(p!=0)
{
for(i=1;i<=n;i++)
if((a[p][i]!=0)&&(vis[i]==0))
{
add(i);
vis[i]=1;
}
p=deleteQ();
if(p!=0)
cout<<p<<"\t";
}
for(i=1;i<=n;i++)
if(vis[i]==0)
bfs(i,n);
}


void add(int item)
{
if(rear==19)
cout<<"QUEUE FULL";
else
```

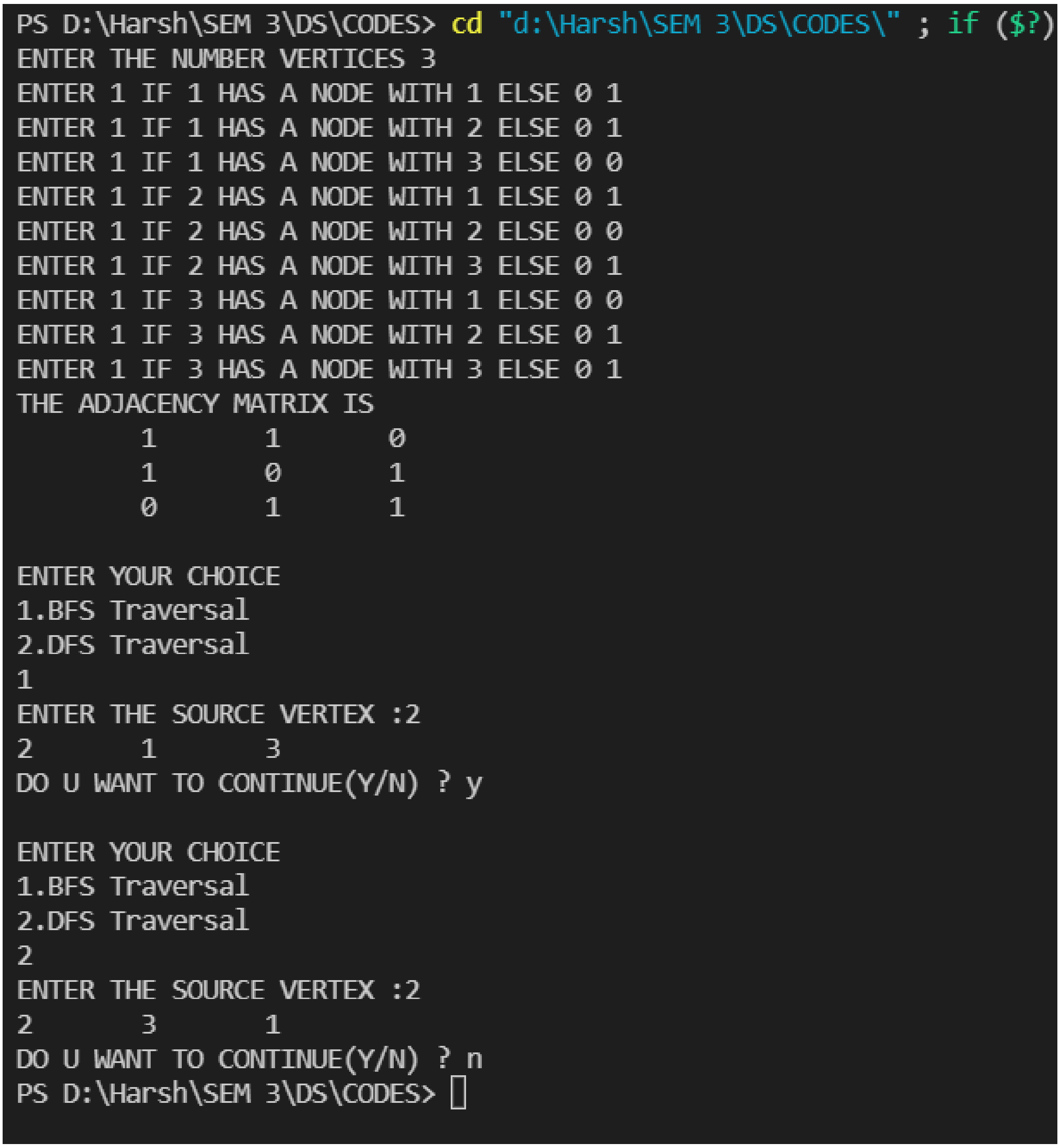


```
{
if(rear==-1)
{
q[++rear]=item;
front++;
}
else
q[++rear]=item;
}
}
int deleteQ()
{
int k;
if((front>rear)||(front==-1))
return(0);
else
{
k=q[front++];
return(k);
}
}


//***************DFS******************//
void dfs(int s,int n)
{
int i,k;
push(s);
vis[s]=1;
k=pop();
if(k!=0)
cout<<k<<"\t";
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)
cout<<k<<"\t";
}
for(i=1;i<=n;i++)
if(vis[i]==0)
dfs(i,n);
}
void push(int item)
```



```
{
if(top==19)
cout<<"Stack overflow ";
else
stack[++top]=item;
}
int pop()
{
int k;
if(top==-1)
return(0);
else
{
k=stack[top--];
return(k);
}
return 0;
}
```

OUTPUT:



```
PS D:\Harsh\SEM 3\DS\CODES> cd "d:\Harsh\SEM 3\DS\CODES\" ; if ($?)
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

ENTER YOUR CHOICE
1.BFS Traversal
2.DFS Traversal
1
ENTER THE SOURCE VERTEX :2
2       1       3
DO U WANT TO CONTINUE(Y/N) ? y

ENTER YOUR CHOICE
1.BFS Traversal
2.DFS Traversal
2
ENTER THE SOURCE VERTEX :2
2       3       1
DO U WANT TO CONTINUE(Y/N) ? n
PS D:\Harsh\SEM 3\DS\CODES> 
```