EX.NO.10.B Implement DFS and BFS graph traversal

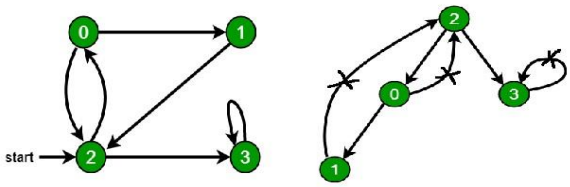
Aim:

To write a C program implement DFS and BFS graph traversal**.**

DESCRIPTION:

Depth First Traversal (or Search) for a graph is similar to Depth First Traversal of a tree. The only catch here is, unlike trees, graphs may contain cycles, so we may come to the same node again. To avoid processing a node more than once, we use a boolean visited array.

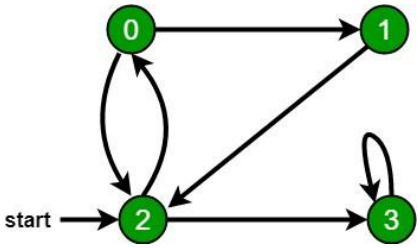
For example, in the following graph, we start traversal from vertex 2. When we come to vertex 0, we look for all adjacent vertices of it. 2 is also an adjacent vertex of 0. If we don't mark visited vertices, then 2 will be processed again and it will become a non-terminating process. A Depth First Traversal of the following graph is 2, 0, 1, 3.



Breadth First Search or BFS for a Graph

Breadth First Traversal (or Search) for a graph is similar to Breadth First Traversal of a tree (See method 2 of this post). The only catch here is, unlike trees, graphs may contain cycles, so we may come to the same node again. To avoid processing a node more than once, we use a boolean visited array. For simplicity, it is assumed that all vertices are reachable from the starting vertex.

For example, in the following graph, we start traversal from vertex 2. When we come to vertex 0, we look for all adjacent vertices of it. 2 is also an adjacent vertex of 0. If we don't mark visited vertices, then 2 will be processed again and it will become a non-terminating process. A Breadth First Traversal of the following graph is 2, 0, 3, 1.



ALGORITHM:

DFS

1. Define a Stack of size total number of vertices in the graph.

2. Select any vertex as starting point for traversal. Visit that vertex and push it on to the Stack.
3. Visit any one of the adjacent vertex of the verex which is at top of the stack which is not visited and push it on to the stack.
4. Repeat step 3 until there are no new vertex to be visit from the vertex on top of the stack.
5. When there is no new vertex to be visit then use back tracking and pop one vertex from the stack.
6. Repeat steps 3, 4 and 5 until stack becomes Empty.

7. When stack becomes Empty, then produce final spanning tree by removing unused edges from the graph

BFS

1. Define a Queue of size total number of vertices in the graph.
2. Select any vertex as starting point for traversal. Visit that vertex and insert it into the Queue.
3. Visit all the adjacent vertices of the verex which is at front of the Queue which is not visited and insert them into the Queue.
4. When there is no new vertex to be visit from the vertex at front of the Queue then delete that vertex from the Queue.
5. Repeat step 3 and 4 until queue becomes empty.
6. When queue becomes Empty, then produce final spanning tree by removing unused edges from the graph

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]);

}

}
```

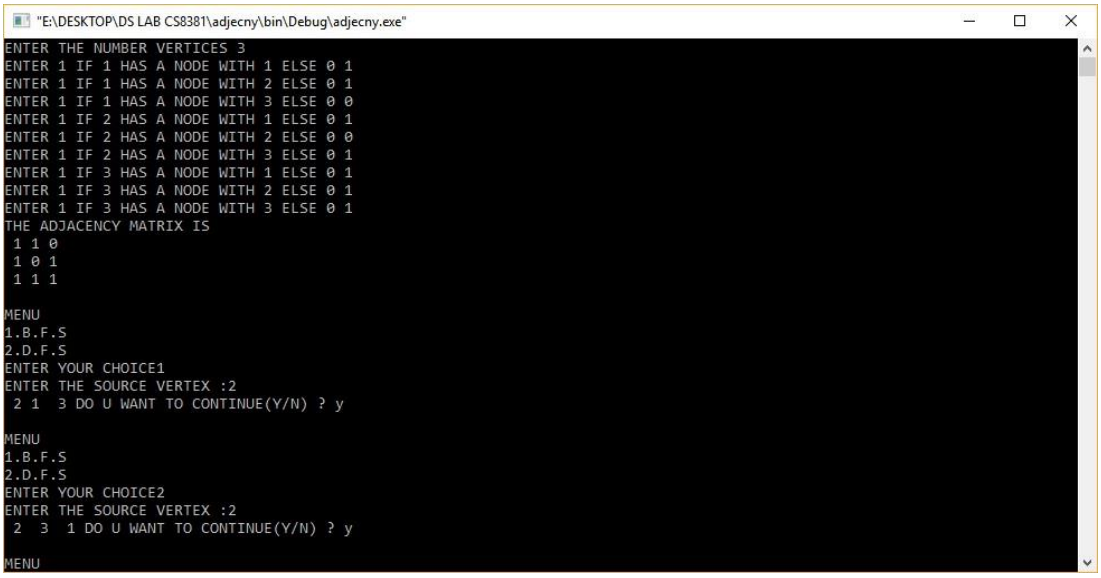
```
 printf("THE ADJACENCY MATRIX IS\n");
for(i=1;i<=n;i++) { for(j=1;j<=n;j++)
{ printf(" %d",a[i][j]);
} printf("\n");
}  do { for(i=1;i<=n;i++) vis[i]=0;
printf("\nMENU"); printf("\n1.B.F.S");
printf("\n2.D.F.S"); printf("\nENTER YOUR
CHOICE"); scanf("%d",&ch); printf("ENTER
THE SOURCE VERTEX :"); 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",&dummy); scanf("%c",&c);
}while((c=='y')||(c=='Y'));

}
//**************BFS(breadth-first search) code**************//
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((a[p][i]!=0)&&(vis[i]==0))
{ add(i);
vis[i]=1; }
p=delete();
if(p!=0) printf("
%d ",p);
} 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);
}
}
//***************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



RESULT:

Thus the C program implemented DFS and BFS graph traversal.