St. Francis Institute of Technology, Mumbai-400 103

Department of Information Technology

A.Y. 2020-2021

Class: SE-ITA/B, Semester: III

Subject: DATA STRUCTURE LAB

Experiment – 10 Directed graph traversal through BFS and DFS

1. **Aim:** Write a C program to implement traversal of a directed graph through BFS and DFS.

2. **Objectives:** After study of this experiment, the student will be able to
   - To understand graph traversal algorithms
   - To understand the working of BFS and DFS algorithm

3. **Outcomes:** After study of this experiment, the student will be able to
   - Implement Graph traversal algorithm.
   - Understand the applications of graph.

4. **Prerequisite:** Graph, BFS and DFS

5. **Requirements:** PC and Turbo C compiler version 3.0

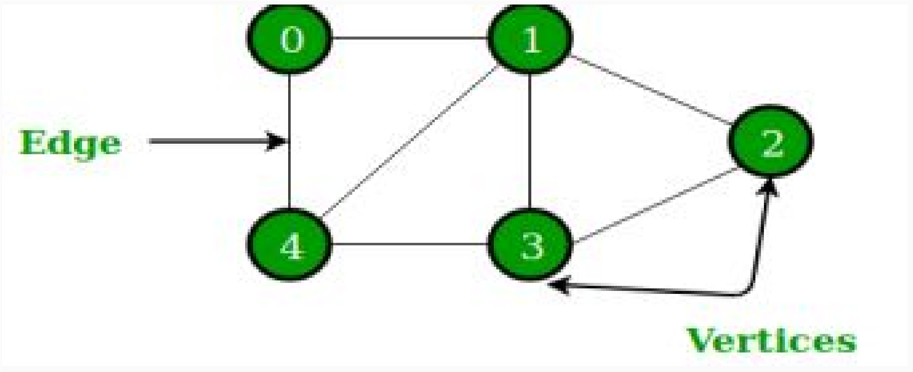
6. **Pre-Experiment Exercise:**

   Brief Theory:

A. What is Graph ?

- 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. More formally a Graph can be defined as,

- A Graph consists of a finite set of vertices(or nodes) and set of Edges which connect a pair of nodes.



B. Explain Graph terminologies with diagram

- **End-vertices** of an edge are the **endpoints** of the edge.
- Two vertices are **adjacent** if they are endpoints of the same edge.

- An edge is **incident** on a vertex if the vertex is an endpoint of the edge.
- **Outgoing edges** of a vertex are directed edges that the vertex is the origin.

Incoming edges of a vertex are directed edges that the vertex is the destination. •

Degree of a vertex, *v*, denoted *deg*(*v*) is the number of incident edges.

Out-degree, *outdeg*(*v*), is the number of outgoing edges.

In-degree, *indeg*(*v*), is the number of incoming edges.

- **Parallel edges** or multiple edges are edges of the same type and end-vertices

Self-loop is an edge with the end vertices the same vertex

Simple graphs have **no** parallel edges or self-loops

C. Explain DFS and BFS

BFS

- Breadth-first search (BFS) is a graph search algorithm that begins at the root node and explores all the neighboring nodes. Then for each of those nearest nodes, the algorithm explores their unexplored neighbor nodes, and so on, until it finds the goal.
- That is, we start examining the node A and then all the neighbors of A are examined. In the next step we examine the neighbors of neighbors of A, so on and so forth

DFS

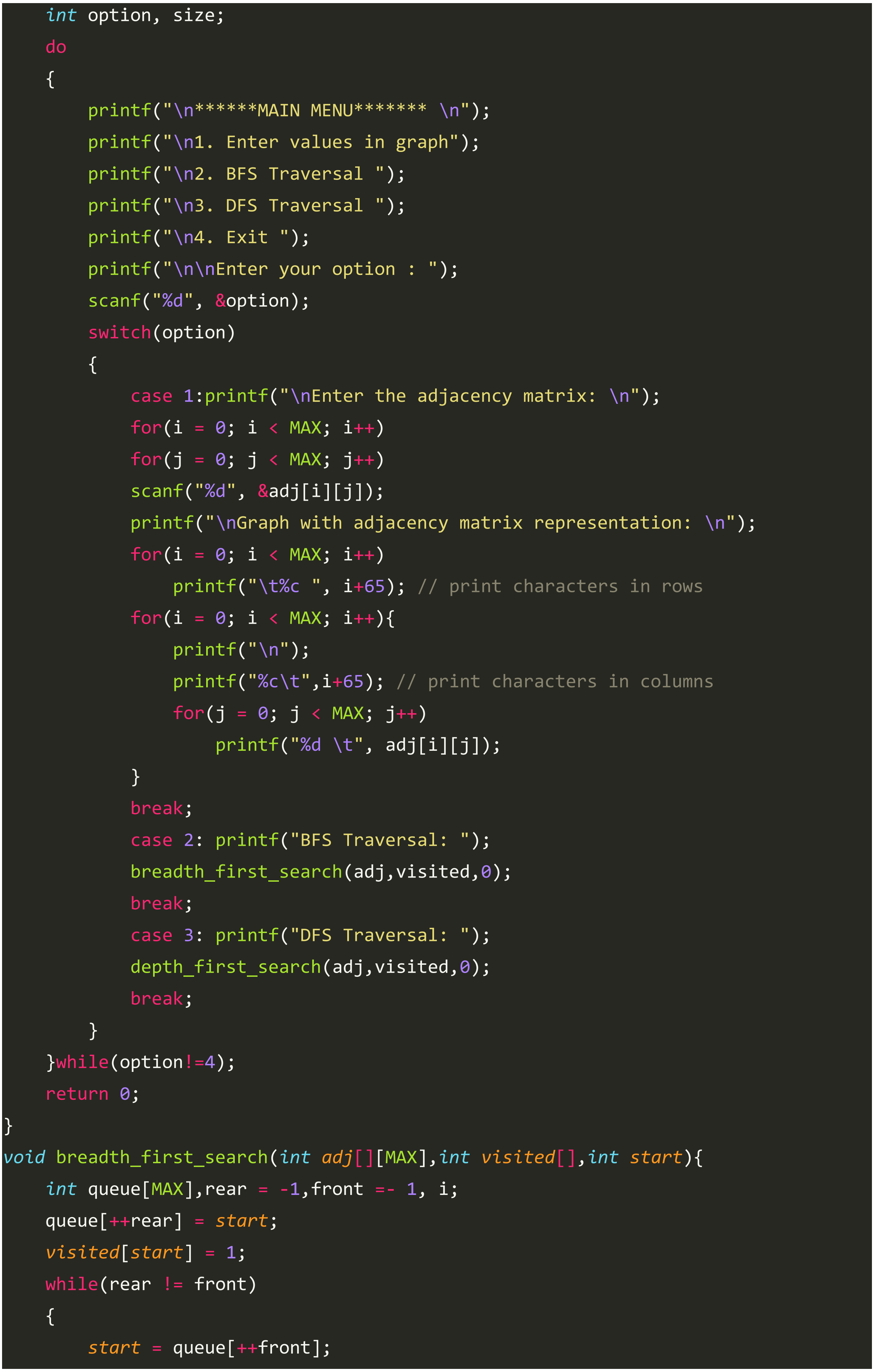
- The Depth First Search algorithm progresses by expanding the starting node of G and thus going deeper and deeper until a goal node is found, or until a node that has no children is encountered. When a dead- end is reached, the algorithm backtracks, returning to the most recent node that has not been completely explored.
- In other words, the Depth- First Search algorithm begins at a starting node A which becomes the current node. Then it examines each node N along a path P which begins at A. That is, we process a neighbor of A, then a neighbor of neighbor of A and so on. During the execution of the algorithm, if we reach a path that has a node N that has already been processed, then we backtrack to the current node. Otherwise, the un-visited (un-processed node) becomes the current node.

7. Laboratory Exercise

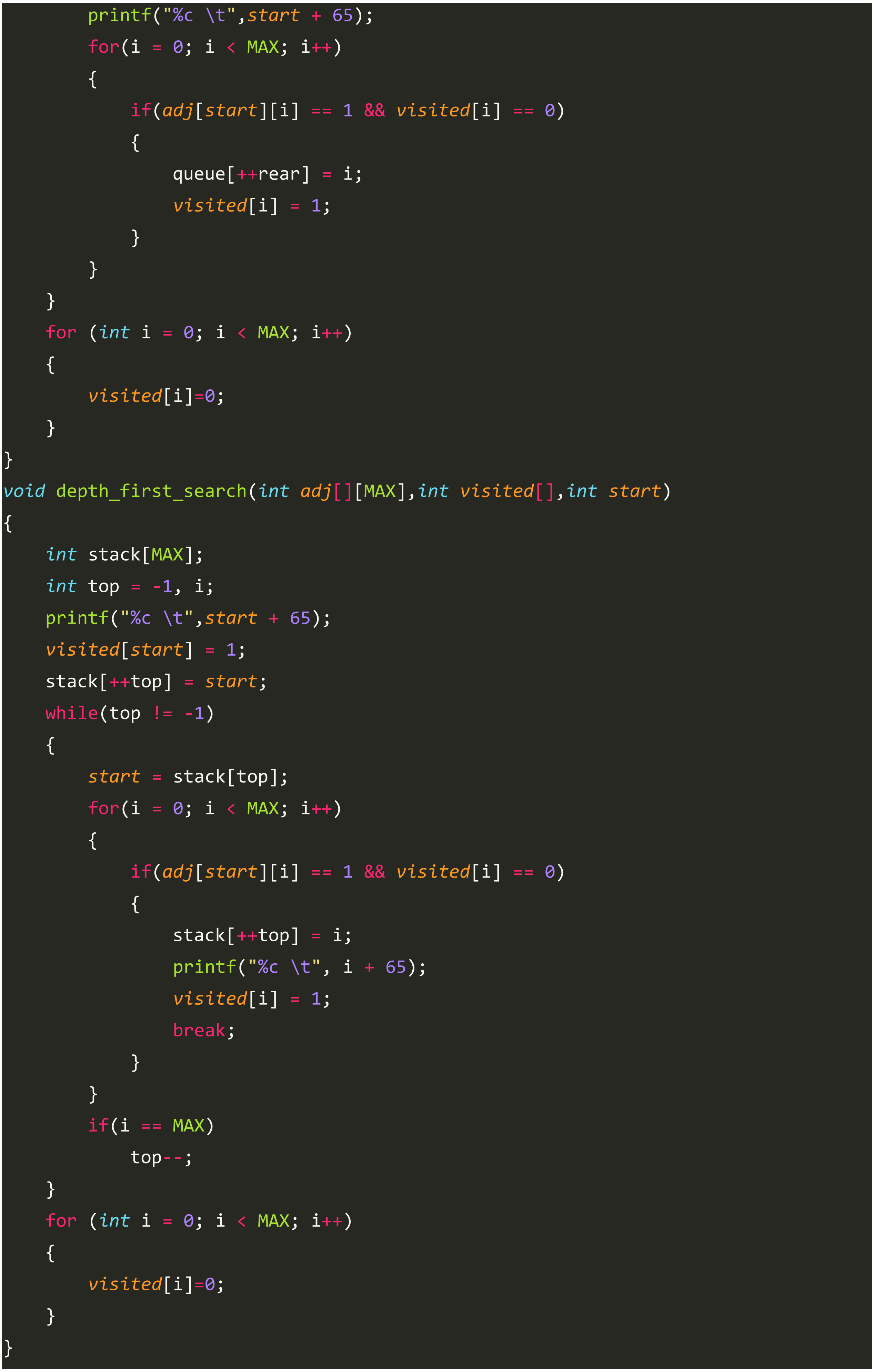
A. Procedure

Write a C program to implement traversal of a directed graph through BFS and DFS.

```c
#include <stdio.h>
#define MAX 5
void breadth_first_search(int adj[][MAX],int visited[],int start);
void depth_first_search(int adj[][MAX],int visited[],int start);
int main()
{
    int visited[MAX] = {0};
    int adj[MAX][MAX], i, j;
```



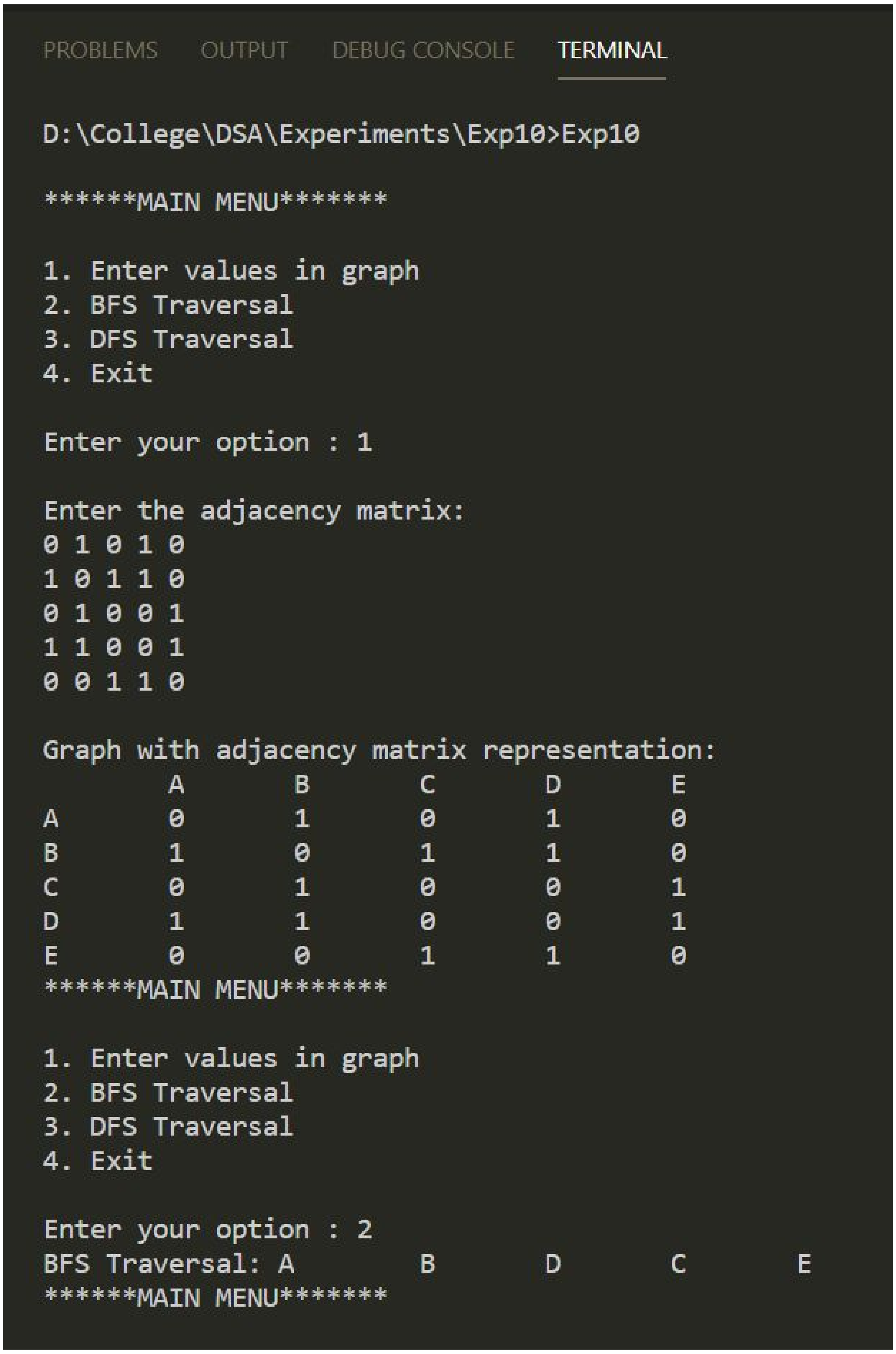
```
    int option, size;
    do
    {
        printf("\n******MAIN MENU******* \n");
        printf("\n1. Enter values in graph");
        printf("\n2. BFS Traversal ");
        printf("\n3. DFS Traversal ");
        printf("\n4. Exit ");
        printf("\n\nEnter your option : ");
        scanf("%d", &option);
        switch(option)
        {
            case 1:printf("\nEnter the adjacency matrix: \n");
            for(i = 0; i < MAX; i++)
            for(j = 0; j < MAX; j++)
            scanf("%d", &adj[i][j]);
            printf("\nGraph with adjacency matrix representation: \n");
            for(i = 0; i < MAX; i++)
                printf("\t%c ", i+65); // print characters in rows
            for(i = 0; i < MAX; i++){
                printf("\n");
                printf("%c\t",i+65); // print characters in columns
                for(j = 0; j < MAX; j++)
                    printf("%d \t", adj[i][j]);
            }
            break;
            case 2: printf("BFS Traversal: ");
            breadth_first_search(adj,visited,0);
            break;
            case 3: printf("DFS Traversal: ");
            depth_first_search(adj,visited,0);
            break;
        }
    }while(option!=4);
    return 0;
}
void breadth_first_search(int adj[][MAX],int visited[],int start){
    int queue[MAX],rear = -1,front =- 1, i;
    queue[++rear] = start;
    visited[start] = 1;
    while(rear != front)
    {
        start = queue[++front];
```



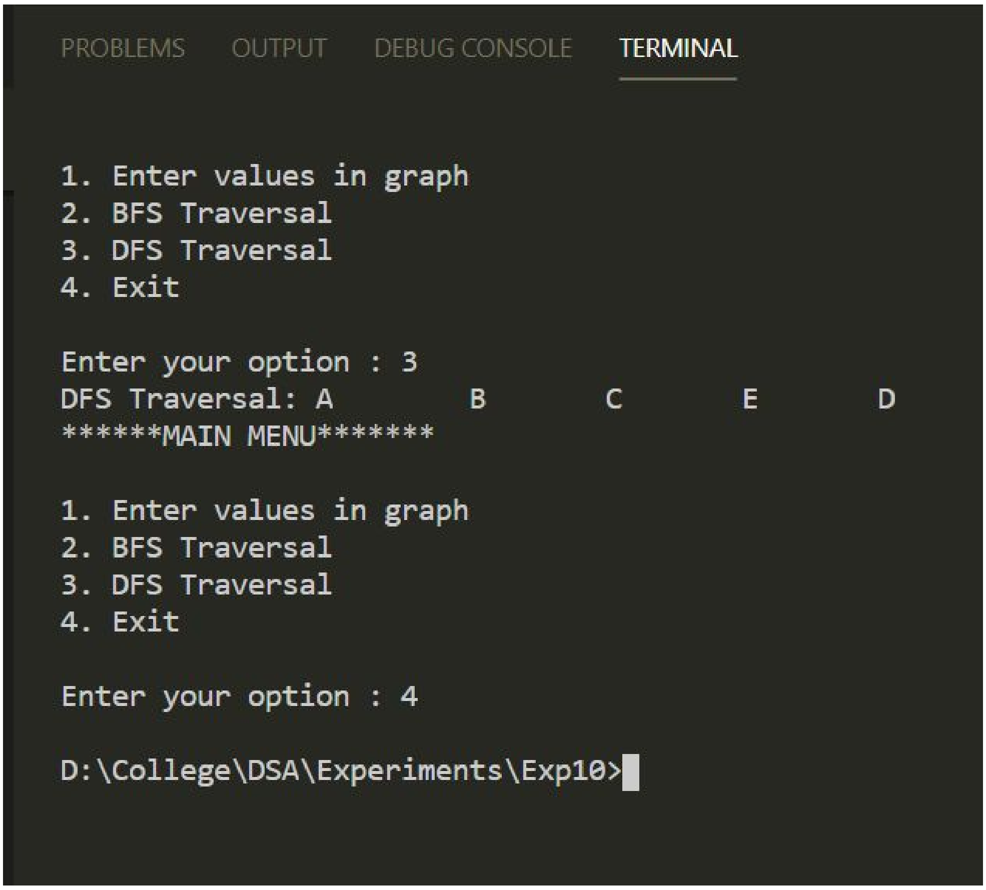
```
        printf("%c \t",start + 65);
        for(i = 0; i < MAX; i++)
        {
            if(adj[start][i] == 1 && visited[i] == 0)
            {
                queue[++rear] = i;
                visited[i] = 1;
            }
        }
    }
    for (int i = 0; i < MAX; i++)
    {
        visited[i]=0;
    }
}
void depth_first_search(int adj[][MAX],int visited[],int start)
{
    int stack[MAX];
    int top = -1, i;
    printf("%c \t",start + 65);
    visited[start] = 1;
    stack[++top] = start;
    while(top != -1)
    {
        start = stack[top];
        for(i = 0; i < MAX; i++)
        {
            if(adj[start][i] == 1 && visited[i] == 0)
            {
                stack[++top] = i;
                printf("%c \t", i + 65);
                visited[i] = 1;
                break;
            }
        }
        if(i == MAX)
            top--;
    }
    for (int i = 0; i < MAX; i++)
    {
        visited[i]=0;
    }
}
```

B. Result/Observation/Program code:

Observe the output for the above code and print it.



```
D:\College\DSA\Experiments\Exp10>Exp10

******MAIN MENU*******

1. Enter values in graph
2. BFS Traversal
3. DFS Traversal
4. Exit

Enter your option : 1

Enter the adjacency matrix:
0 1 0 1 0
1 0 1 1 0
0 1 0 0 1
1 1 0 0 1
0 0 1 1 0

Graph with adjacency matrix representation:
        A       B       C       D       E
A       0       1       0       1       0
B       1       0       1       1       0
C       0       1       0       0       1
D       1       1       0       0       1
E       0       0       1       1       0
******MAIN MENU*******

1. Enter values in graph
2. BFS Traversal
3. DFS Traversal
4. Exit

Enter your option : 2
BFS Traversal: A        B       D       C       E
******MAIN MENU*******
```

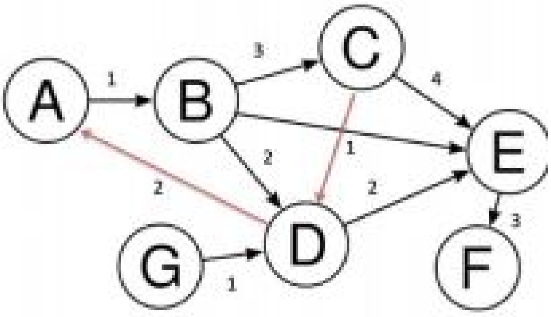


```
1. Enter values in graph
2. BFS Traversal
3. DFS Traversal
4. Exit

Enter your option : 3
DFS Traversal: A       B       C       E       D
******MAIN MENU*******

1. Enter values in graph
2. BFS Traversal
3. DFS Traversal
4. Exit

Enter your option : 4

D:\College\DSA\Experiments\Exp10>
```

8. Post-Experiments Exercise

A. Questions:

1. Find the BFS and DFS traversal of given graph and show all the steps starting with node A.



B. Conclusion:

1. Summary of Experiment
2. Importance of Experiment

C. References:

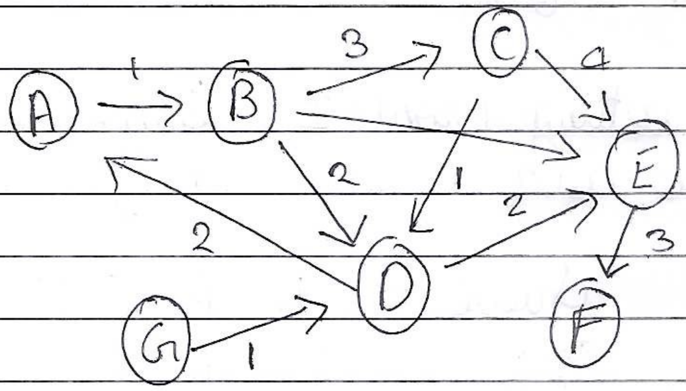
1. S. K Srivastava, Deepali Srivastava; Data Structures through C in Depth; BPB Publications; 2011.
2. Reema Thareja; Data Structures using C; Oxford.
3. Data Structures A Pseudocode Approach with C, Richard F. Gilberg & Behrouz A. Forouzan, second edition, CENGAGE Learning.

Yash Mahajan SE IT B 04

8. Post Experiment Excercise :-

A. Questions :-

1. Find the BFS and DFS traversal of given graph starting from node A.



Adjacency Matrix

$$\begin{array}{c|ccccccc} & A & B & C & D & E & F & G \\ \hline A & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ B & 0 & 0 & 3 & 2 & 1 & 0 & 0 \\ C & 0 & 0 & 0 & 1 & 4 & 0 & 0 \\ D & 2 & 0 & 0 & 0 & 2 & 0 & 0 \\ E & 0 & 0 & 0 & 0 & 0 & 3 & 0 \\ F & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ G & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{array}$$

Initially add ~~ato~~ A to Queue and null to Org,

FRONT = 0	Queue = A
REAR = 0	Orig = \0

Page :
Date : / /

Yash Mahajan SE IT B 04

Dequeue A by seting Front = 1, and enque neighbours of A. Also add A to orig of its neighbours.

Front = 1 Queue = A B
Rear = 1 orig = \0 A

Deque B by setting Front = 2, enque neighbours of B. Also add B to orig of its neighbours

Front = 2 Queue = A B C D E
Rear = 4 Orig = \0 A B B B

Dequeue C by setting Front = 3, enque neighbours of C. But neighbours of C are already visited, so we will not enque them.

Front = 3 Queue = A B C D E
Rear = 4 Orig = \0 A B B B

Dequeue D by setting Front = 4, enque neighbours of D, But they are already visited so we will not enquee them.

Front = 4 Queue = A B C D E
Rear = 4 Orig \0 A B B B

Deque E by setting front = 5, enque neighbours of E. Also add E to orig of its neighbours

Yash Mahajan SE IT B 04

Front = 5 Queue = A B C D E F
Rear = 5 Orig = \0 A B B B E

Since F has no neighbours we stop the traversal.

The Breadth first search of ~~go~~ given graph is : A B C D E F.

(ii) - DFS

~~Step1~~ Push A to the stack

stack : A

Pop A and print it, push all neighbours of A

~~St~~ Print : A stack : B

Pop the topmost element of stack and print it and push all its neighbours.

Print : B stack : C D E

Pop the topmost element of stack and ~~print~~ print. push all its neighbours.

Yash Mahajan SE IT B 04

Print E Stack : C D F

Pop the topmost element of stack and print. Since F does not have any unvisited neighbours, ~~stack remains~~ nothing is pushed in the stack.

Print F Stack : C D

Pop the topmost element of stack and print. since D doesnot have any unvisited neighbours nothing is pushed in the stack.

Print: D Stack : C

Pop the topmost element of stack and print, since C has no unvisited neighbours, nothing is pushed to the stack.

Print : C stack :

Since stack is empty. stops DFS.

DFS traversal of given graph is :-

A B E F D C

Yash Mahajan SE IT B 04

B) Conclusion :-

In this experiment we have written C programs to implement traversal of a directed graph using Breadth first search (BFS) and Depth ~~searc~~ First search (DFS) algorithms

BFS is used in networking to find all the connected nodes. It is also used to find shortest path and minimum spanning tree in an unweighted graph.

DFS is used to find a cycle in a graph; It is also used for topological sorting which is used for scheduling jobs from the given dependencies among jobs.