



# Vidyavardhini's College of Engineering and Technology

## Department of Artificial Intelligence & Data Science

### Experiment No. 9: Depth First Search and Breath First Search

**Aim : Implementation of DFS and BFS traversal of graph.**

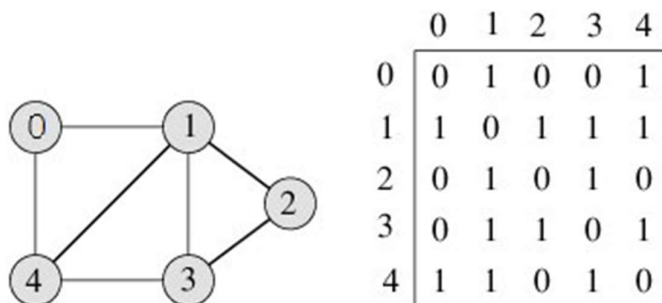
**Objective:**

1. Understand the Graph data structure and its basic operations.
2. Understand the method of representing a graph.
3. Understand the method of constructing the Graph ADT and defining its

operations **Theory:**

A graph is a collection of nodes or vertices, connected in pairs by lines referred to as edges. A graph can be directed or undirected.

One method of traversing through nodes is depth first search. Here we traverse from the starting node and proceed from top to bottom. At a moment we reach a dead end from where the further movement is not possible and we backtrack and then proceed according to left right order. A stack is used to keep track of a visited node which helps in backtracking.



**DFS Traversal –0 1 2 3 4**

**Algorithm**

Algorithm: DFS\_LL(V)

Input: V is a starting vertex

Output : A list VISIT giving order of visited vertices during traversal.

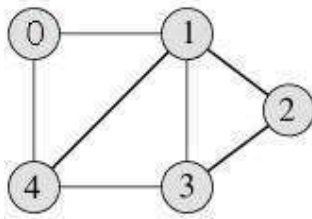
Description: linked structure of graph with gptr as pointer

1. if gptr = NULL then      print “Graph is empty” exit
2. u=v
3. OPEN.PUSH(u)
4. while OPEN.TOP !=NULL do  
    u=OPEN.POP()  
    if search(VISIT,u) = FALSE then  
        INSERT\_END(VISIT,u)



```
Ptr = gptr(u)
While ptr.LINK != NULL do
    Vptr = ptr.LINK
    OPEN.PUSH(vptr.LABEL)
End while
End if
End while
5. Return VISIT
6. Stop
```

## BFS Traversal



	0	1	2	3	4
0	0	1	0	0	1
1	1	0	1	1	1
2	0	1	0	1	0
3	0	1	1	0	1
4	1	1	0	1	0

## BFS Traversal – 0 1 4 2 3 Algorithm

```
Algorithm:   DFS()
i=0         count=1
visited[i]=1
print("Visited vertex
i")
repeat this till queue is empty or all nodes visited
repeat this for all nodes from first till last
if(g[i][j]!=0&&visited[j]!=1)
{ push(j) } i=pop()
print("Visited vertex
i")      visited[i]=1
```



## Vidyavardhini's College of Engineering and Technology

### Department of Artificial Intelligence & Data Science

count++ Algorithm:

BFS() i=0 count=1

visited[i]=1

print("Visited vertex

i")

repeat this till queue is empty or all nodes visited

repeat this for all nodes from first till last

if(g[i][j]!=0&&visited[j]!=1) { enqueue(j)

}

i=dequeue()

print("Visited vertex

i") visited[i]=1

count++

**Code: Dfs**

```
#include <stdio.h> #define MAX 5 void
```

```
depth_first_search(int adj[][MAX],int visited[],int start)
```

```
{
```

```
    int stack[MAX];
```

```
    int top = - 1, i; printf("%c-
```

```
",start + 65); visited[start]
```

```
= 1; stack[++top] = start;
```

```
while(top!= -1) {
```

```
    start = stack[top];
```

```
    for(i = 0; i < MAX; i++) {
```

```
        if(adj[start][i] && visited[i] == 0) {
```

```
            stack[++top] =
```

```
            i; printf("%c-", i
```

```
+ 65); visited[i]
```

```
= 1; break;
```



```
} }
```

```
    if(i == MAX)
```

```
top--;
```

```
}}
```

```
int main() {
```

```
    int adj[MAX][MAX];
```

```
    int visited[MAX] = {0}, i, j;
```

```
    printf("\n Enter the adjacency
```

```
matrix: ");    for(i = 0; i < MAX;
```

```
i++)          for(j = 0; j < MAX;
```

```
j++) scanf("%d", &adj[i][j]);
```

```
    printf("DFS Traversal: ");
```

```
    depth_first_search(adj,visited,0);
```

```
    printf("\n");
```

```
        return 0;
```

```
}
```

### **Bfs**

```
#include <stdio.h> #define MAX 10 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];
```

```
    if(start == 4)
```

```
    printf("5\t"); else
```



# Vidyavardhini's College of Engineering and Technology

## Department of Artificial Intelligence & Data Science

```
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;

        } } }

int main() {

    int visited[MAX] = {0};

    int adj[MAX][MAX], i,j;

    printf(" Enter the adjacencymatrix
:")

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

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

        scanf("%d", &adj[i][j]);

    breadth_first_search(adj,visited,0);

    return 0; }
```

### Output:

#### DFS:

```
Enter the adjacency matrix: 0 1 1 0 0
1 0 0 1 0
1 0 0 1 1
0 1 1 0 1
0 0 1 1 0
DFS Traversal: A-B-D-C-E-
-
```

#### BFS :



# Vidyavardhini's College of Engineering and Technology

## Department of Artificial Intelligence & Data Science

```
Enter the adjacency matrix: 0 1 0 1 0 0 0 0 0 0
1 0 1 0 1 0 0 0 0 0
0 1 0 0 0 1 0 0 0 0
1 0 0 0 0 0 1 0 0 0
0 1 0 0 0 0 0 1 0 0
0 0 1 0 0 0 0 0 1 0
0 0 0 1 0 0 0 0 0 1
0 0 0 0 1 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0
0 0 0 0 0 0 1 0 0 0
A      B      D      C      S      G      F      H      J      I
```

### Conclusion:

1) Write the graph representation used by your program and explain why you choose that.

- The program uses an adjacency matrix to represent the graph. An adjacency matrix is a 2D array where each cell  $adj[i][j]$  represents an edge between vertex  $i$  and vertex  $j$ . The value in the cell indicates the presence (usually 1) or absence (usually 0) of an edge. This representation was chosen because it provides a simple and straightforward way to implement both depth-first search (DFS) and breadth-first search (BFS) algorithms. It allows for easy traversal and checking of connections between nodes. However, it may not be the most memory-efficient representation for sparse graphs.

2) Write the applications of BFS and DFS other than finding connected nodes and explain how it is attained?

➤ Applications of BFS and DFS:

- Shortest Path Finding: Both BFS and DFS can be used to find the shortest path between two nodes in a graph. BFS typically finds the shortest path in an unweighted graph, while DFS can find paths in weighted graphs using backtracking.
  - Web Crawling: Search engines use BFS to crawl and index web pages efficiently. Starting from a seed webpage, they explore links level by level.
  - Solving Puzzles: DFS is commonly used to solve puzzles like mazes, Sudoku, and the NQueens problem. It explores possible solutions by trying various paths.
  - Network Broadcasting: BFS is used for broadcasting messages or information in computer networks. It ensures that messages reach all connected nodes without unnecessary duplication.
  - Topological Sorting: DFS can be used to find the topological ordering of vertices in a directed acyclic graph, which is essential in scheduling and task management.
  - Spanning Trees: Both algorithms are used to find minimum spanning trees in graphs, which have applications in network design and clustering.
  - Social Networking: BFS can help find the degree of separation between two individuals in a social network, such as the "Six Degrees of Kevin Bacon" game.
  - Garbage Collection: DFS is used in garbage collection algorithms to reclaim memory from objects that are no longer reachable in programming languages like Java.
- These algorithms can be adapted and applied in various domains to solve a wide range of problems beyond simply finding connected nodes in a graph.