

K. J. Somaiya College of Engineering, Mumbai
(A Constituent College of Somaiya Vidyavihar University)
Department of Computer Engineering

Batch: B4	Roll. No.: 16010122828
Experiment:6	
Grade: AA / AB / BB / BC / CC / CD /DD	

Title: Implementation of Graph traversal menu driven program (DFS and BFS).
--

Objective: To understand graph as data structure and methods of traversing Graph

Expected Outcome of Experiment:

CO	Outcome
CO3	Demonstrate sorting and searching methods.

Websites/books referred:

<https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/>

https://www.tutorialspoint.com/data_structures_algorithms/breadth_first_traversal.htm

Abstract: - (Definition of Graph, types of graphs, and difference and similarity between graph & tree)

Graph:

A graph is a data structure that consists of the following two components a finite set of vertices also called as nodes and a finite set of ordered pair of the form (u, v) called as edge.

Types of Graph:

1. Null Graph

A graph is known as null graph if there are no edges in the graph.

2. Trivial Graph

Graph having only a single vertex, it is the smallest graph possible.

3. Undirected Graph

A graph in which edges do not have any direction. That is the nodes are unordered pairs in the definition of every edge.

4. Directed Graph

A graph in which edge has direction. That is the nodes are ordered pairs in the definition of every edge.

5. Connected Graph

The graph in which from one node we can visit any other node in the graph is known as a connected graph.

6. Disconnected Graph

The graph in which at least one node is not reachable from a node is known as a disconnected graph.

7. Regular Graph

The graph in which the degree of every vertex is equal to the other vertices of the graph.

Let the degree of each vertex be **K** then the graph is called **K-regular**.

8. Complete Graph

The graph in which from each node there is an edge to each other node.

9. Cycle Graph

The graph in which the graph is a cycle in itself, the degree of each vertex is 2.

10. Cyclic Graph

A graph containing at least one cycle is known as a Cyclic graph.

11. Directed Acyclic Graph

A Directed Graph that does not contain any cycle.

12. Bipartite graph

A graph in which vertex can be divided into two sets such that vertex in each set does not contain any edge between them.

Difference between graph and tree

No.	Graph	Tree
1	Each node can have any number of edges.	General trees consist of the nodes having any number of child nodes. But in case of binary trees every node can have at the most two child nodes.
2	There is no unique node called root in graph.	There is a unique node called root in trees.
3	A cycle can be formed.	There will not be any cycle.
4	Applications: For finding shortest path in networking graph is used.	Applications: For game trees, decision trees, the tree is use

Similarity between graph and tree:

Both graph are non linear data structures consisting of nodes and vertices/edges.

Every node in both graph and tree has at least one data variable and a pointer.

Algorithm for DFS/BFS:

DFS

- Mark all the visited array elements as false.
- Call DFS for first node.
- Mark the node as visited and print it
- Iterate through adjacency list (node) and if they are not visited then recursively call dfs for them.
- End

BFS

- Mark all the visited array elements as false.
- Call BFS for first node.
- Mark the node as visited and print it
- Iterate through adjacency list (node), if then they are not visited then mark them true and add then to the queue.
- If queue has elements the remove the top one and call BFS for that node.
- End.

Implementation Details:

Code and output screenshots:

CODE:

```
#include <stdio.h>
#include <stdlib.h>
struct node { int vertex;
struct node* next;
};
struct Graph {
int numVertices; int* visited;
};
struct node* createNode(int value) {
struct node* newNode = malloc(sizeof(struct node)); newNode->vertex = value;
newNode->next = NULL; return newNode;
}
struct Graph* createGraph(int vertices) {
struct Graph* graph = malloc(sizeof(struct Graph)); graph->numVertices = vertices;
graph->adjLists = malloc(vertices * sizeof(struct node*));
graph->visited = malloc(vertices * sizeof(int));
int i;
for (i = 0; i < vertices; i++) { graph->adjLists[i] = NULL; graph->visited[i] = 0;
}
return graph;
}
void dfs(struct Graph* graph, int vertex) { struct node* adjList = graph->adjLists[vertex]; struct node* temp = adjList;
graph->visited[vertex] = 1; printf("Visited %d \n", vertex);
while (temp != NULL) {
int connectedVertex = temp->vertex;
if (graph->visited[connectedVertex] == 0) { DFS(graph, connectedVertex);
}
temp = temp->next;
}
}
struct node* newNode = createNode(dest); newNode->next = graph->adjLists[src];
graph->adjLists[src] = newNode;
newNode->next = graph->adjLists[dest]; graph->adjLists[dest] = newNode;
void printG(struct Graph* graph) {
int v;
```

K. J. Somaiya College of Engineering, Mumbai

(A Constituent College of Somaiya Vidyavihar University)

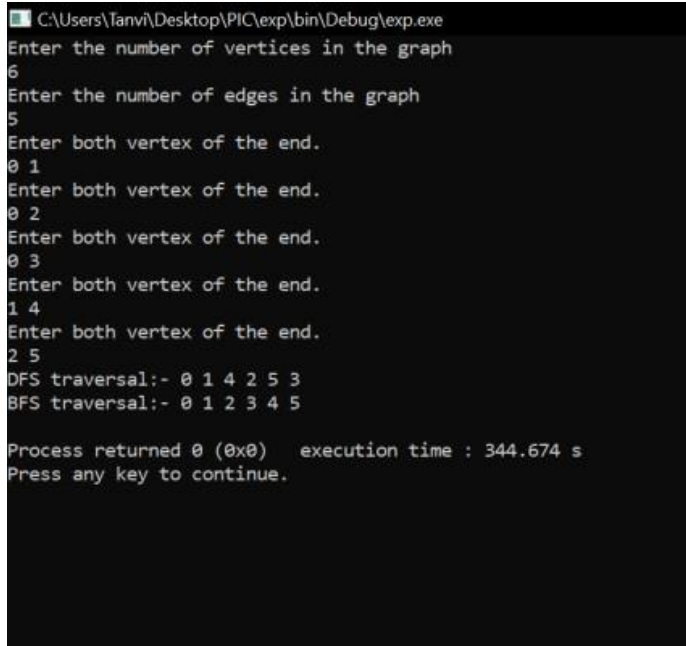
Department of Computer Engineering

```
for (v = 0; v < graph->numVertices; v++) { struct node* temp = graph->adjLists[v];
printf("\n Adjacency list of vertex %d\n ", v); while (temp) {
printf("%d -> ", temp->vertex); temp = temp->next;
}
printf("\n");
}
}

int main() {
int num, source, destination;
printf("Enter the number of nodes in the graph "); scanf("%d" ,&num);
struct Graph* graph = createGraph(num); int i;
for(i=0; i<num; i++)
{
printf("Enter the source "); scanf("%d" , &source); printf("enter the destination
"); scanf("%d" ,&destination);
addEdge(graph, source, destination);
}

printG(graph);
dfs(graph, 2);
return 0;
}
```

Output



```
C:\Users\Tanvi\Desktop\PIC\exp\bin\Debug\exp.exe
Enter the number of vertices in the graph
6
Enter the number of edges in the graph
5
Enter both vertex of the end.
0 1
Enter both vertex of the end.
0 2
Enter both vertex of the end.
0 3
Enter both vertex of the end.
1 4
Enter both vertex of the end.
2 5
DFS traversal:- 0 1 4 2 5 3
BFS traversal:- 0 1 2 3 4 5

Process returned 0 (0x0)   execution time : 344.674 s
Press any key to continue.
```

1. Enlist all the Steps followed and various options explored, explain your program logic, classes and methods used.

- I made a class called Graph with a 2d vector (adjacency list), visited vector and a queue;
- Addedge, bfs, dfs and reset functions were declared in the class.
- When a class object is made, the constructor resizes the adjacency list and visited list.
- Then add edge function is called to insert the vertices in source and destination lists.
- After that DFS function is called to obtain the dfs traversal for that vertex.
- Then reset function is called to reassign false to all the elements of visited list.
- Lastly the BFS function is called to get the bfs traversal for that vertex.

2. Explain the Importance of the approach followed by you

Using a adjacency list instead of array makes the graph traversal $O(V+E)$ where V is number of vertices and E is number of edges instead of $O(V^2)$.

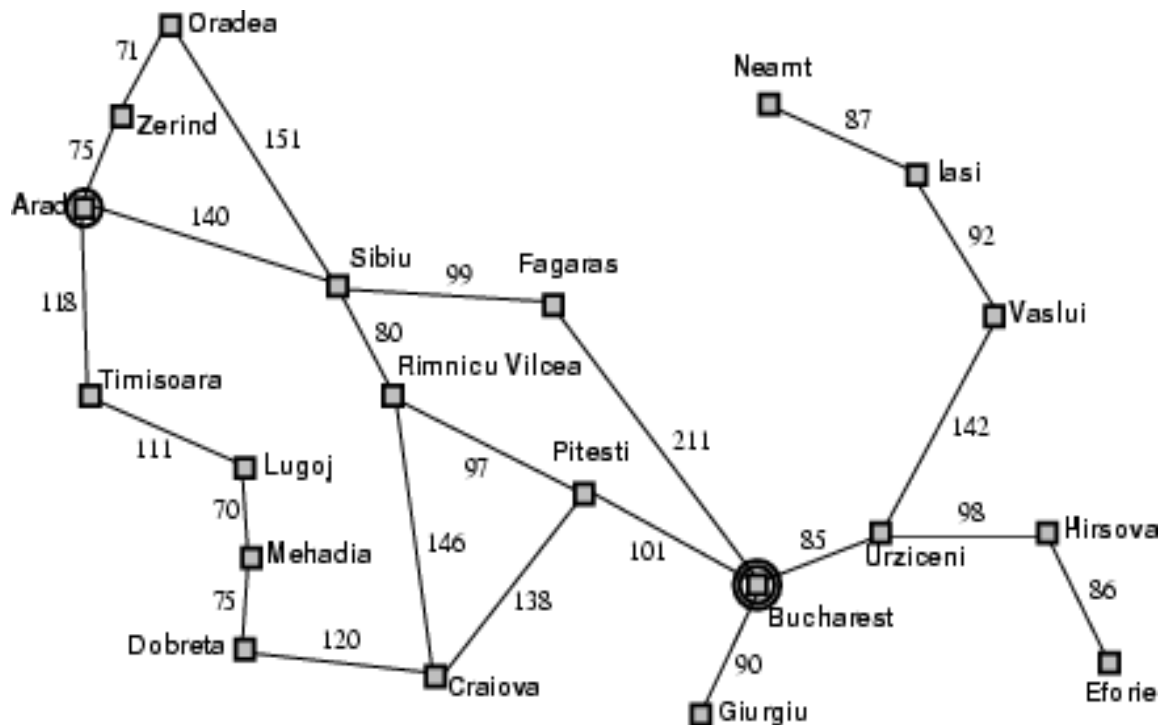
K. J. Somaiya College of Engineering, Mumbai
(A Constituent College of Somaiya Vidyavihar University)
Department of Computer Engineering

Post lab questions-

a. Differentiate between BFS and DFS.

	BFS	DFS
1.	BFS stands for Breadth First Search.	DFS stands for Depth First Search.
2.	BFS(Breadth First Search) uses Queue data structure for finding the shortest path.	DFS(Depth First Search) uses Stack data structure.
3.	BFS can be used to find single source shortest path in an unweighted graph, because in BFS, we reach a vertex with minimum number of edges from a source vertex.	In DFS, we might traverse through more edges to reach a destination vertex from a source.
3.	BFS is more suitable for searching vertices which are closer to the given source.	DFS is more suitable when there are solutions away from source.
4.	BFS considers all neighbors first and therefore not suitable for decision making trees used in games or puzzles.	DFS is more suitable for game or puzzle problems. We make a decision, then explore all paths through this decision. And if this decision leads to win situation, we stop.
5.	The Time complexity of BFS is $O(V + E)$ when Adjacency List is used and $O(V^2)$ when Adjacency Matrix is used, where V stands for vertices and E stands for edges.	The Time complexity of DFS is also $O(V + E)$ when Adjacency List is used and $O(V^2)$ when Adjacency Matrix is used, where V stands for vertices and E stands for edges.
6.	Here, siblings are visited before the children	Here, children are visited before the siblings

- b. Give sequence of the nodes visited as per BFS and DFS strategy for following example. Source- Arad, Destination- Bucharest (Traversal would stop after destination is reached)



BFS – **Arad**->Zerind -> Sibiu -> Timisoara -> Oradea ->Fagaras -> Rimnicu Vilcea -> Lugoj -> **Bucharest**

DFS - **Arad**->Zerind->Oradea->Sibiu->Fagaras->**Bucharest**

Conclusion: -

In this experiment we learnt about two types of traversals in a graph and implemented them.