# Graph Traversals: BFS and DFS

Mudit Rathore, Ishani Mishra, and Gugloth Lavanya

***Abstract*—In today's world, all tasks are performed by machines, computers, robots etc. All real life problems can be visualized as graphs. Thus graphs prove to be an efficacious aid in solving typical problems. Hence, generating and visualizing information as graphs is now-a-days an important step towards problem solving.**

**The first step towards solving a problem is storing of a graph in computer memory and traversing it. Graphs can be stored as adjacency and incidence- matrices or lists. There are two prevalent graph traversals namely Breadth First Search (BFS) and Depth First Search (DFS). This paper aims at analyzing both traversals on Adjacency Matrix and Incidence Matrix for simple, directed or undirected graphs.**

## I. Introduction

A graph **G(V,E)** is a collection of sets **V** and **E**, where **V** is a collection of vertices and **E** is a collection of edges. An edge is a line or an arc connecting two vertices and is denoted by a pair **(i,j)** where **i, j** belong to the set of vertices V. A graph can be of two types namely *directed graph* and *undirected graph [1]*.

A graph, which has unordered pair of vertices, is called an *undirected graph*. It has an edge between vertices u and v then it can be represented as either **(u,v)** or **(v,u)**.

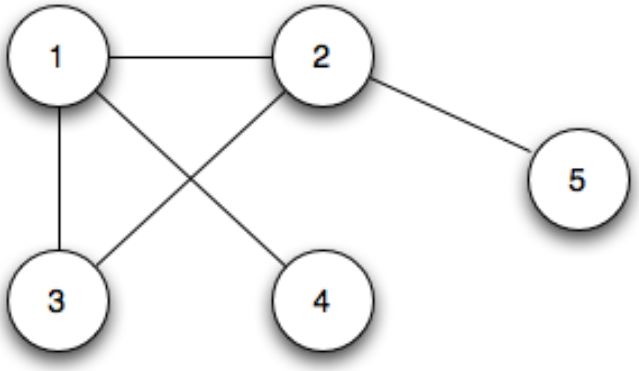



Fig. 1. A simple undirected graph

A *directed graph* or a digraph is a graph which has ordered pair of vertices **(u,v)** where **u** is the tail and **v** is the head of the edge. In this type of graph, **(u,v)** represents a direction which is not same as **(v,u)**, they represent different edges.

**Adjacency** is a relation between two vertices of a graph. A vertex **v** is adjacent to vertex **u** if there is an edge from vertex u to vertex v i.e. **edge(u,v)** $\in$ **E.**

An adjacency matrix is a matrix holding this relation. It is a square matrix **A** used to represent a finite graph. The elements of the matrix indicate whether pairs of vertices ($\mathbf{v}_i$, $\mathbf{v}_j$) are adjacent or not, in the graph. If vertex $\mathbf{v}_i$ is adjacent to vertex $\mathbf{v}_j$, then $\mathbf{a}_{ij}$ = 1 and $\mathbf{a}_{ji}$ =1, if they are not connected

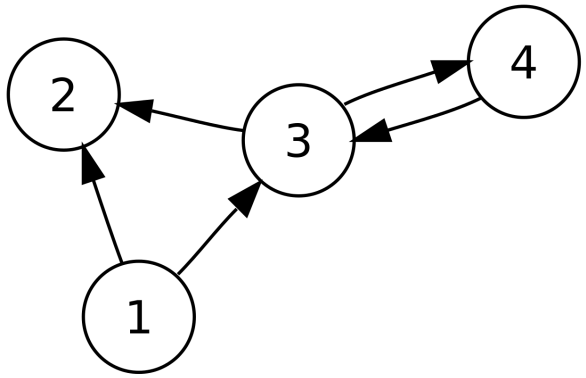



Fig. 2. A simple directed graph

then $\mathbf{a}_{ij}$ **= 0** and $\mathbf{a}_{ji}$ **= 0.** In the special case of a finite simple graph, the adjacency matrix is a (0,1)-matrix of order **|V|** with zeros on its diagonal $\mathbf{a}_{ii}$ **= 0** which represents loop condition.

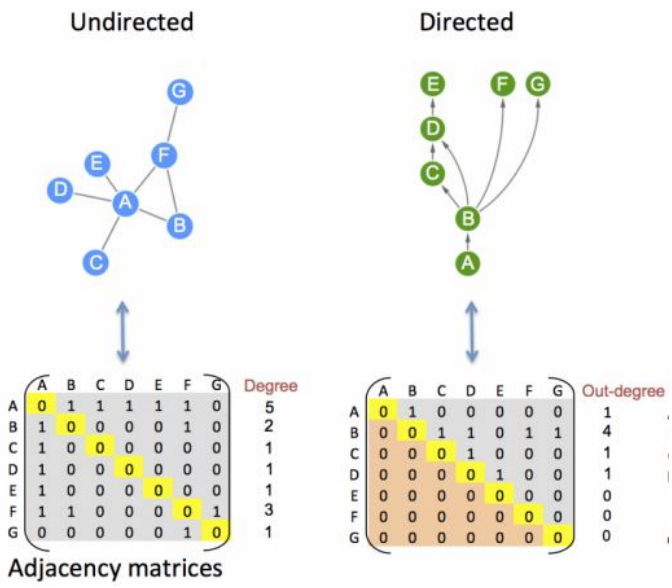
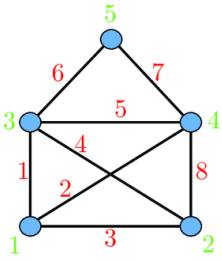



Fig. 3. Adjacency Matrices for Undirected and Directed Graphs.

The **Incidence Matrix [2]**A of a graph **G(V,E)** with **V** as a set of vertices and **E** as the set of edges is a **|V| x |E|** matrix $\mathbf{A} = [\mathbf{a}_{ij}]$**,** where the if rows correspond to the vertices and the of columns correspond to edges. For an undirected graph, if $j^{th}$ edge is incident on the $i^{th}$ vertex then $\mathbf{a}_{ij}$ **= 1** if the **0** otherwise.

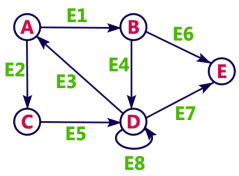
For a Directed Graph $\mathbf{a}_{ij}$ **= -1** if the tail of the $j^{th}$ edge incident on the $i^{th}$ vertex, $\mathbf{a}_{ij}$ **= 1** if the head of the $j^{th}$ edge incident on the $i^{th}$ vertex and $a_{ij}$ = 0 otherwise.

Breadth first search or BFS [3] traversal expands the frontier




$$\mathbf{A}_G = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} \\ \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix} \end{array}$$

Fig. 4. Incidence Matrix for Undirected Graph.




| | E1 | E2 | E3 | E4 | E5 | E6 | E7 | E8 |
|---|---|---|---|---|---|---|---|---|
| A | 1 | 1 | -1 | 0 | 0 | 0 | 0 | 0 |
| B | -1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| C | 0 | -1 | 0 | 0 | 1 | 0 | 0 | 0 |
| D | 0 | 0 | 1 | -1 | -1 | 0 | 1 | 1 |
| E | 0 | 0 | 0 | 0 | 0 | -1 | -1 | 0 |

Fig. 5. Incidence Matrix for a Directed Graph.

between discovered and undiscovered nodes uniformly across the breadth of the frontier, discovering all nodes at a distance k from the source node before nodes at distance k + 1. **BFS(s)** computes for every node **v** ∈ **G** the distance from **s** to **v** in **G**. This traversal is equivalent to level order traversal of trees.

Depth First Search [4] or DFS traversal algorithm we travel along a path in the graph and when we reach the dead end of the traversal, and go back and try a different one. In DFS [5] , you go as deep as possible down one path before backing up and trying a different one. This technique is named so because search proceeds deeper in the graph i.e. we traverse along a path as deep as we can. Traversal using DFS is like traveling a maze.

## II. Methodology

Input from the user is either an incidence matrix or an adjacency matrix on which traversal has to be done. The below specified DFS and BFS [6] are for adjacency matrix. If input is incidence matrix, it is first converted to an adjacency matrix and then traversal is done on the converted adjacency matrix.

### *A. Converting incidence matrix into adjacency matrix:*

1. Traverse the incidence matrix column-wise.
2. For each column, identify the two row entries for which the matrix cell entries are 1 and 1 (in case of undirected graph) or 1 and -1 (in case of directed graph). Say they are u and v.
3. Update the adjacency matrix (initially filled with zeros) of the corresponding (u,v) to 1.
4. If the graph is undirected, update (u,v) and (v,u) of the adjacency matrix both as 1.
5. Repeat 2, 3, 4 till all columns of the incidence matrix are not seen.

### *B. Breadth First Search (BFS)*

For the algorithm, a *queue* is used to store the waiting vertices.

A state array stores the state of the vertices which are *unvisited, waiting, visited.*

*1) Algorithm:* During the algorithm, any vertex will be in one of the three states- initial, waiting, visited. At the start of

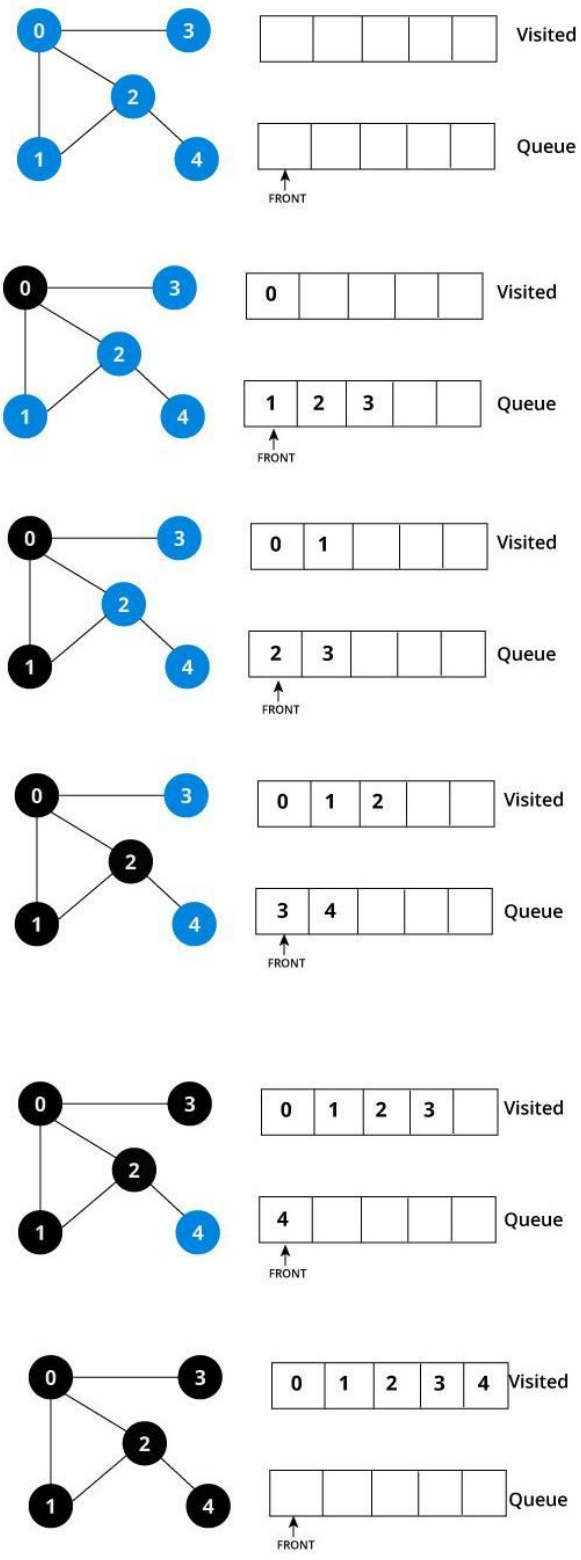



Fig. 6.

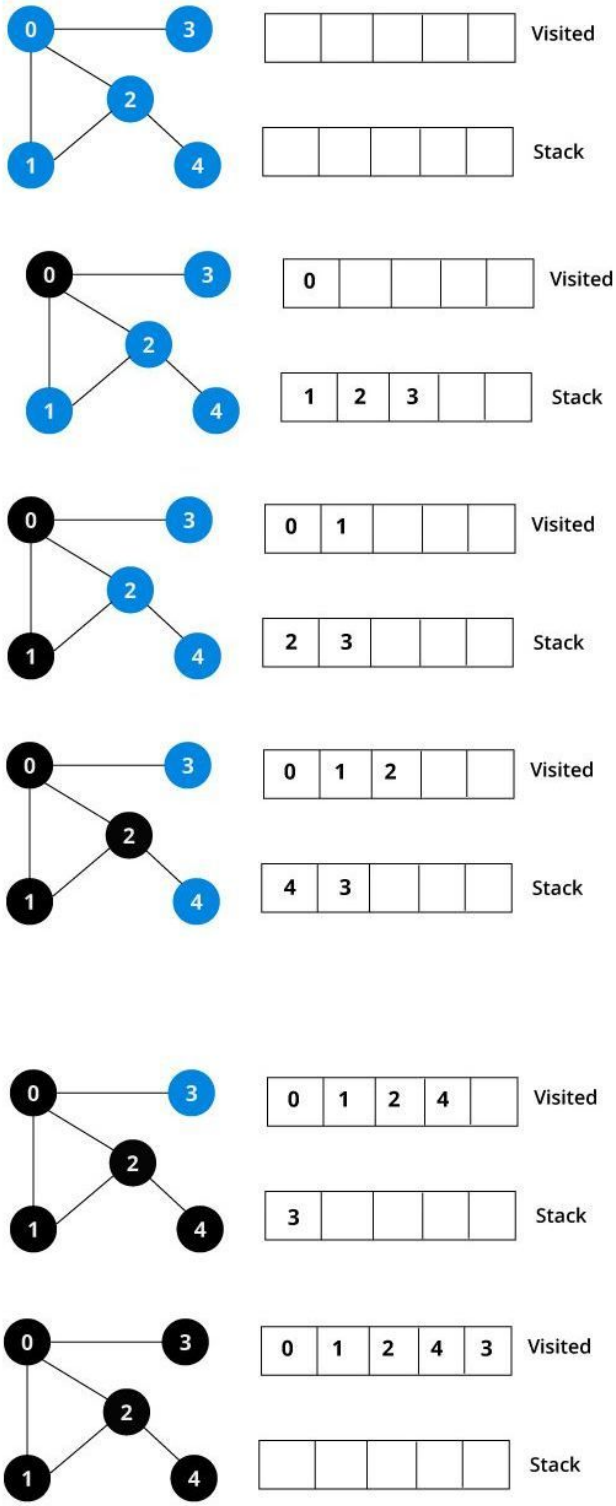



Fig. 7. Depth First Travesal of the graph, starting from vertex 0: 0 1 2 4 3

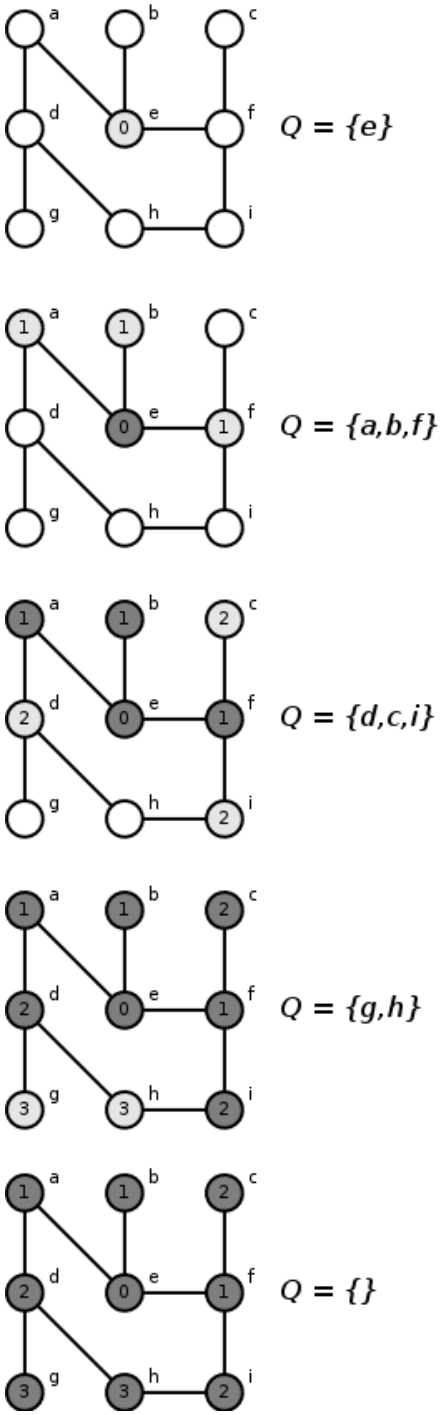



Fig. 8. Breadth First Traversals of the graph, starting from vertex e: e a b f d c i g h

the algorithm all vertices are in the initial state, when a vertex is inserted in the queue, its state will change from initial to waiting. When a vertex will be deleted, its state will change from waiting to visited.

Initially the queue is empty and all vertices are in the initial state.

1. Insert the starting vertex to the queue.
2. Delete front element from the queue and visit it, change its state to visited.
3. Look for adjacent vertices of the deleted element, and from these insert only those vertices to the queue which are in the initial state. Change the state of these inserted vertices from initial to waiting.
4. Repeat steps 2, 3 until queue is empty.

### C. Depth First Search (DFS)

For the algorithm, a *stack* is used to store the unvisited adjacent vertices.

A state array stores the state of the vertices which are *unvisited, visited*.

*1) Algorithm:* During the algorithm any vertex will be in one of the two status- initial and visited. At the start of the algorithm, all vertices will be in the initial state, and when a vertex is popped from the stack its state will change to visited.

Initially stack is empty, and all vertices are in unvisited state.

1. Push the starting vertex on the stack.
2. Pop a vertex from the stack.
3. If popped vertex is in the unvisited state, visit it and change the state to visited. Push all unvisited vertices adjacent to the popped vertex.
4. Repeat steps 2 and 3 until stack is empty.

## III. Result

### A. Complexity analysis:

Given a graph with V vertices and **E** edges.

*1) BFS and DFS:* Visiting each node within the while loop takes O(V) time in total. Inside the for loop edges are scanned to check for adjacent vertices of $v_i$, hence it takes O(E). For each vertex, edges are scanned, hence overall, we see that our run-time is O(#nodes visited * #edges scanned) = **O(V * E)**.

*2) Converting incidence matrix to adjacency matrix:* E columns are traversed and for each column V vertices are checked for occurrence of 1 or -1 or both. Hence, its complexity is **O(E * V)**.

### B. Outputs

1. *Input: Directed graph adjacency matrix*

Enter the (n x n) Adjacency Matrix:
0 0 1 0 0
1 0 0 0 0
0 0 0 1 1
0 1 0 0 0
0 0 0 1 0
The DFS traversal of the given matrix is: 0 2 3 1 4
The BFS traversal of the given matrix is: 0 2 3 4 1

2. *Input: Undirected graph adjacency matrix*

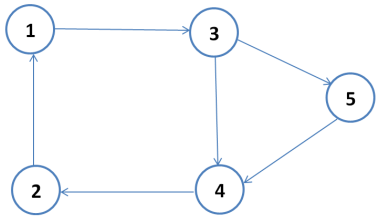



Fig. 9. Input- Incidence and adjacency matrix of this Directed Graph.

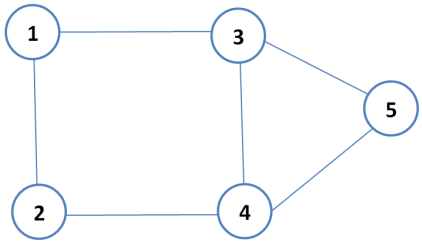



Fig. 10. Input: Incidence and adjacency matrix of this Undirected Graph

Enter the (n x n) 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
The DFS traversal of the given matrix is: 0 1 3 2 4
The BFS traversal of the given matrix is: 0 1 2 3 4

3. *Input: Directed graph incidence matrix*

Enter the (v x e) incidence matrix:
1 0 0 -1 0 0
0 0 -1 1 0 0
-1 1 0 0 1 0
0 -1 1 0 0 -1
0 0 0 0 -1 1
The DFS traversal of the given matrix is: 0 2 3 1 4
The BFS traversal of the given matrix is: 0 2 3 4 1

4. *Input: Undirected graph incidence matrix*

Enter the (v x e) incidence matrix:
1 1 0 0 0 0
1 0 1 0 0 0
0 1 0 1 1 0
0 0 1 1 0 1
0 0 0 0 1 1
The DFS traversal of the given matrix is: 0 1 3 2 4
The BFS traversal of the given matrix is: 0 1 2 3 4

## IV. Discussion

In case of directed graphs, there may arise a case in which a vertex $\mathbf{v}_i$ is not reachable from the starting vertex **s**. The

output traversal in these cases will be incomplete as it will not contain all vertices. Only the ones reachable from the starting vertex will be displayed. If we want to visit all the vertices, then we can take an unvisited vertex as the starting vertex and again start the traversal from there. This process will continue until all vertices are visited.

Another approach could be that we ask the user to input another starting vertex s and run the traversal again till we find a traversal which visits all the vertices of the graph.

## REFERENCES


[1] J. Cook, M. Wootters, V. Williams, R. Verma, S. Hildick-Smith, and G. Valiant, "Graphs." [Online]. Available: https://web.stanford.edu/class/cs161/Lectures/Lecture9/CS161Lecture09.pdf

[2] "Incidence Matrices." [Online]. Available: http://mathonline.wikidot.com/incidence-matrices

[3] "Breadth First Search or BFS for a Graph." [Online]. Available: https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/

[4] N. Kaur and D. Garg, "Analysis Of The Depth First Search Algorithms."

[5] "Depth First Search or DFS for a Graph." [Online]. Available: https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/

[6] S. Mukhopadhyay, R. Agarwal, D. Patel, K. Som, and A. Sarkar, "Complexity Based on Traversal of Graphs," *International Journal of Current Trends in Engineering & Research (IJCTER)*, vol. 2, no. 12, pp. 46–51, 2016.