Yusrat Sadia Nailat

Graphs:

Key terminologies:

1. **Graph:** A graph represents a set of vertices V and a set of edges E that represents the relationship among the vertices. Typically, it is represented as $G := G(V, E)$. if $a, b \in V$ and $\exists E_{ab} \in E$, meaning there is an edge between vertices, then the edge is represented as (a, b)

2. **Degree of a vertex:** the number of neighboring vertices of a vertex is called the degree of a vertex
3. **Path**: a path is a between two vertices a and d of a graph can the following $(a, b), (b, c), (c, p), (k, d)$
4. **Disconnected graph:** If there exists two vertices such that they are not connected by a path, then the graph is called disconnect. The following graph is disconnected have **two components:**
5. **Cycle:** A cycle in a graph is a sequence of edges that starts and ends at the same vertex.

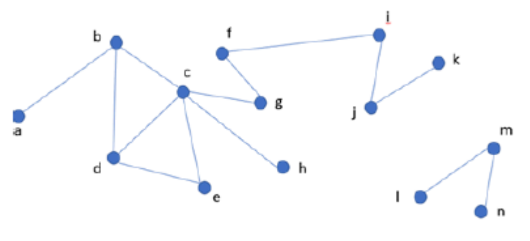


Figure 5: an undirected graph

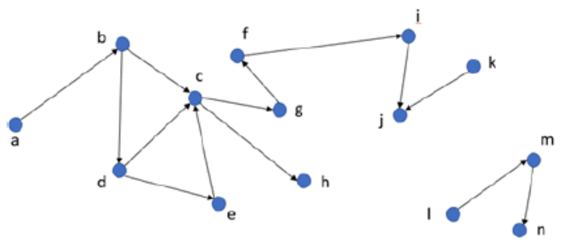


Figure 6: a directed graph

Two representations:

1. **Adjacency Matrix:** if there are N vertices in a graph, we represent their relations in a $N \times N$ matrix. The elements of the matrix represent the weight of each edge of the graph.
2. **Adjacency List**: the outgoing edges of every vertex are sorted in a linked list (For directed graph). For undirected graph, we list both vertex (in and out going)

Incoming vertices - column

Outgoing vertices - row

Space Complexity For Adjacency List:

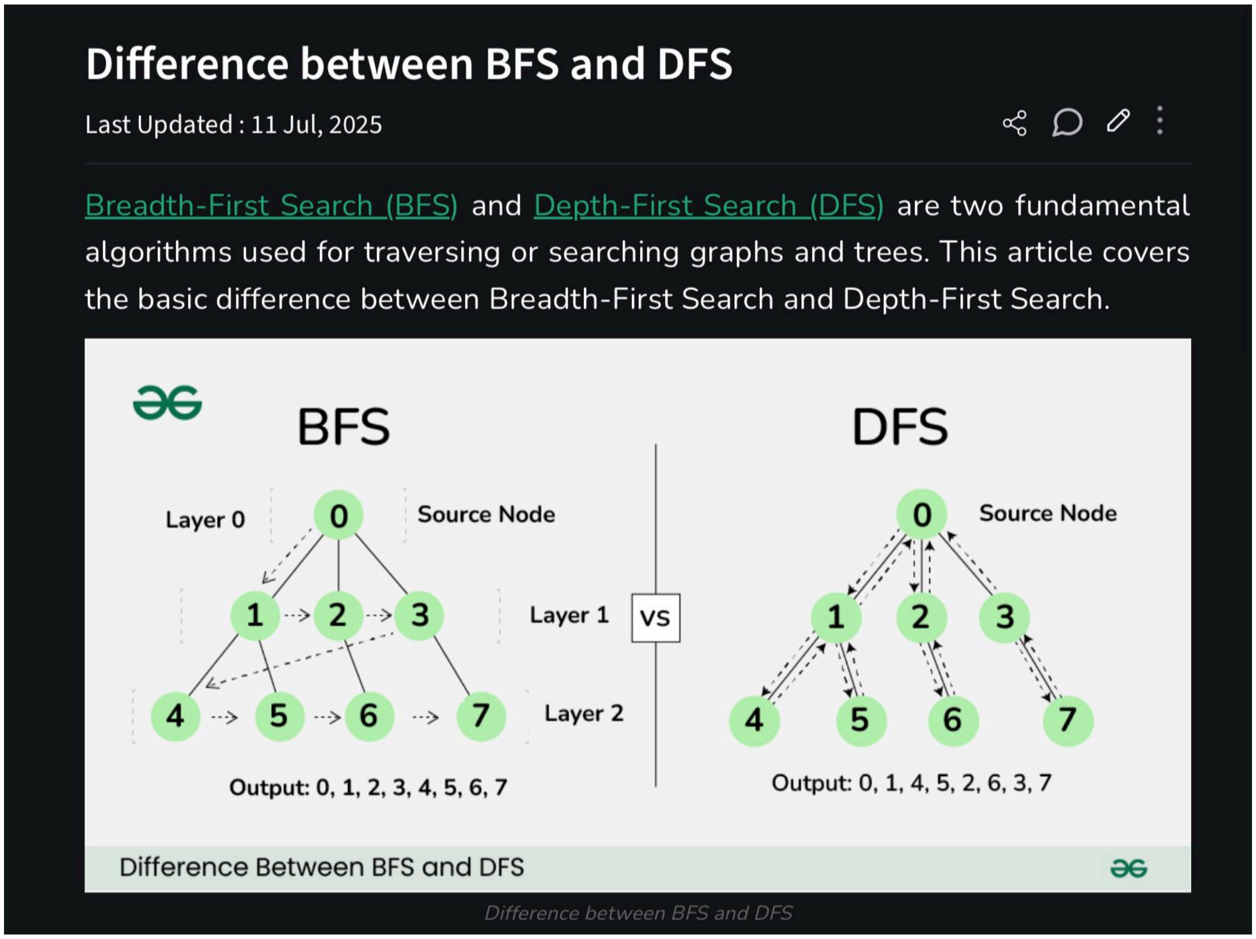
if a graph has N vertices and M edges, then the adjacency list representation needs an array of size N and the total number of nodes in various lists in different array indices will be total $2M$ (in case of undirected graph) or M (in case of a directed graph). Hence the space requirement for the adjacency list representation is proportional to $N + M$. In the asymptotic notation, we can write the space complexity as $\mathcal{O}(N + M)$.

Space complexity for Adjacency Matrix

For a graph with N vertices and M edges, the adjacency matrix representation will take $N \times N$ entries, that is, N^2 entries. Consequently, the asymptotic space complexity in this representation is $\mathcal{O}(N^2)$. Apparently, the space cost of adjacency matrix representation is significantly higher than that of adjacency list representation. Then why do we use this second representation? The answer is many computations are easier in the matrix representation than in the list representation. We trade space with time when using any representation

Graph Traversal:

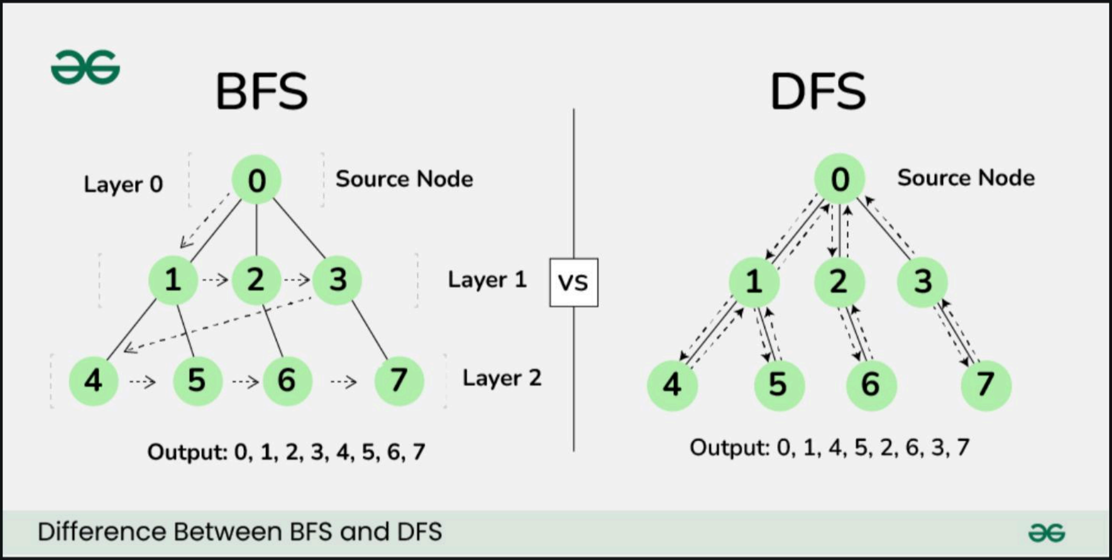
Yusrat Sadia Nailat



Difference between BFS and DFS

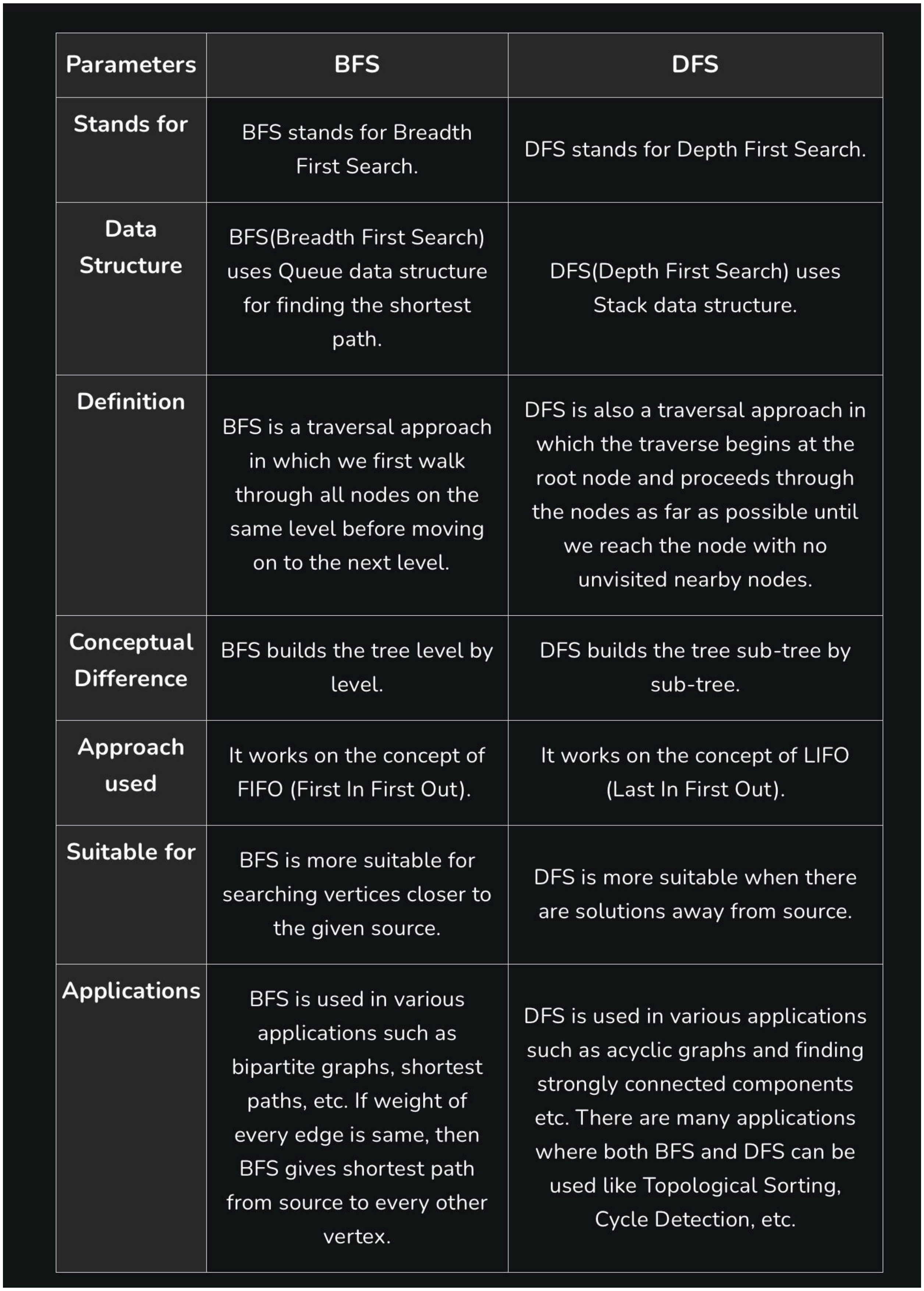
Last Updated : 11 Jul, 2025

Breadth-First Search (BFS) and Depth-First Search (DFS) are two fundamental algorithms used for traversing or searching graphs and trees. This article covers the basic difference between Breadth-First Search and Depth-First Search.


BFS
Layer 0
Source Node
Layer 1
Layer 2
Output: 0, 1, 2, 3, 4, 5, 6, 7
VS
DFS
Source Node
Output: 0, 1, 4, 5, 2, 6, 3, 7
Difference Between BFS and DFS

Difference between BFS and DFS

Yusrat Sadia Nailat



Parameters	BFS	DFS
Stands for	BFS stands for Breadth First Search.	DFS stands for Depth First Search.
Data Structure	BFS(Breadth First Search) uses Queue data structure for finding the shortest path.	DFS(Depth First Search) uses Stack data structure.
Definition	BFS is a traversal approach in which we first walk through all nodes on the same level before moving on to the next level.	DFS is also a traversal approach in which the traverse begins at the root node and proceeds through the nodes as far as possible until we reach the node with no unvisited nearby nodes.
Conceptual Difference	BFS builds the tree level by level.	DFS builds the tree sub-tree by sub-tree.
Approach used	It works on the concept of FIFO (First In First Out).	It works on the concept of LIFO (Last In First Out).
Suitable for	BFS is more suitable for searching vertices closer to the given source.	DFS is more suitable when there are solutions away from source.
Applications	BFS is used in various applications such as bipartite graphs, shortest paths, etc. If weight of every edge is same, then BFS gives shortest path from source to every other vertex.	DFS is used in various applications such as acyclic graphs and finding strongly connected components etc. There are many applications where both BFS and DFS can be used like Topological Sorting, Cycle Detection, etc.