**Difference between BFS and DFS**

**Breadth-First Search:**  
**BFS, Breadth-First Search,** is a vertex-based technique for finding the shortest path in the graph. It uses a [Queue data structure](https://www.geeksforgeeks.org/queue-data-structure/) that follows first in first out. In BFS, one vertex is selected at a time when it is visited and marked then its adjacent are visited and stored in the queue. It is slower than DFS.   
**Example**:

**Input:**  
 A  
 / \  
 B C  
 / / \  
 D E F

**Output:**

A, B, C, D, E, F

**Depth First Search:**  
**DFS,** [**Depth First Search**](https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/), is an edge-based technique. It uses the [Stack data structure](https://www.geeksforgeeks.org/stack-data-structure/) and performs two stages, first visited vertices are pushed into the stack, and second if there are no vertices then visited vertices are popped.   
**Example:**

**Input:**  
 A  
 / \  
 B D  
 / / \  
 C E F

**Output:**

A, B, C, D, E, F

**BFS vs DFS**

| **S.No.** | **Parameters** | **BFS** | **DFS** |
| --- | --- | --- | --- |
| **1.** | **Stands for** | BFS stands for Breadth First Search. | DFS stands for Depth First Search. |
| **2.** | **Data Structure** | BFS(Breadth First Search) uses Queue data structure for finding the shortest path. | DFS(Depth First Search) uses Stack data structure. |
| **3.** | **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. |
| **4.** | **Technique** | BFS can be used to find a single source shortest path in an unweighted graph because, in BFS, we reach a vertex with a minimum number of edges from a source vertex. | In DFS, we might traverse through more edges to reach a destination vertex from a source. |
| **5.** | **Conceptual Difference** | BFS builds the tree level by level. | DFS builds the tree sub-tree by sub-tree. |
| **6.** | **Approach used** | It works on the concept of FIFO (First In First Out). | It works on the concept of LIFO (Last In First Out). |
| **7.** | **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. |
| **8.** | **Suitability for Decision-Trees** | 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, and the then explore all paths through this decision. And if this decision leads to win situation, we stop. |
| **9.** | **Time Complexity** | 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. |
| **10.** | **Visiting of Siblings/ Children** | Here, siblings are visited before the children. | Here, children are visited before the siblings. |
| **11.** | **Removal of Traversed Nodes** | Nodes that are traversed several times are deleted from the queue. | The visited nodes are added to the stack and then removed when there are no more nodes to visit. |
| **12.** | **Backtracking** | In BFS there is no concept of backtracking. | DFS algorithm is a recursive algorithm that uses the idea of backtracking |
| **13.** | **Applications** | BFS is used in various applications such as bipartite graphs, shortest paths, etc. | DFS is used in various applications such as acyclic graphs and topological order etc. |
| **14.** | **Memory** | BFS requires more memory. | DFS requires less memory. |
| **15.** | **Optimality** | BFS is optimal for finding the shortest path. | DFS is not optimal for finding the shortest path. |
| **16.** | **Space complexity** | In BFS, the space complexity is more critical as compared to time complexity. | DFS has lesser space complexity because at a time it needs to store only a single path from the root to the leaf node. |
| **17.** | **Speed** | BFS is slow as compared to DFS. | DFS is fast as compared to BFS. |
| **18,** | **Tapping in loops** | In BFS, there is no problem of trapping into infinite loops. | In DFS, we may be trapped in infinite loops. |
| **19.** | **When to use?** | When the target is close to the source, BFS performs better. | When the target is far from the source, DFS is preferable. |