What is BFS?

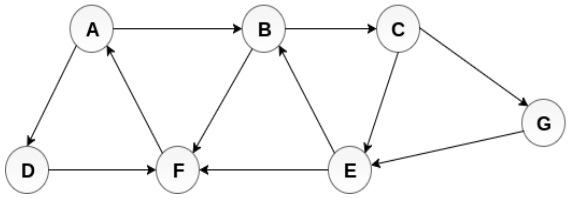
BFS is an algorithm that is used to graph data or searching tree or traversing structures. The algorithm efficiently visits and marks all the key nodes in a graph in an accurate breadthwise fashion.

This algorithm selects a single node (initial or source point) in a graph and then visits all the nodes adjacent to the selected node. Once the algorithm visits and marks the starting node, then it moves towards the nearest unvisited nodes and analyses them.

Once visited, all nodes are marked. These iterations continue until all the nodes of the graph have been successfully visited and marked. The full form of BFS is the Breadth-first search.

Rule 1 – Visit the starting vertex and mark it as visited.
Display it.
Set a pointer to the starting vertex.(Currently working Vertex.)

Rule 2 – if(Currently working Vertex has adjacent unvisited vertex)
{
Visit the adjacent unvisited vertex and Mark it as visited.
Insert it in a queue. (enqueue)
Display it.
}
else{
Update the pointer to the first element of queue.
Remove the first element from queue. (dequeue)
}
(repeat rule 2 until queue is empty)



Adjacency Lists

A : B, D
B : C, F
C : E, G
G : E
E : B, F
F : A
D : F

Solution:

Minimum Path P can be found by applying breadth first search algorithm that will begin at node A and will end at E. the algorithm uses two queues, namely **QUEUE1** and **QUEUE2**. **QUEUE1** holds all the nodes that are to be processed while **QUEUE2** holds all the nodes that are processed and deleted from **QUEUE1**.
Lets start examining the graph from Node A.

1. Add A to QUEUE1 and NULL to QUEUE2.
   QUEUE1 = {A}
   QUEUE2 = {NULL}

2. Delete the Node A from QUEUE1 and insert all its neighbours. Insert Node A into QUEUE2
   QUEUE1 = {B, D}
   QUEUE2 = {A}

3. Delete the node B from QUEUE1 and insert all its neighbours. Insert node B into QUEUE2.
QUEUE1 = {D, C, F}
QUEUE2 = {A, B}

4. Delete the node D from QUEUE1 and insert all its neighbours. Since F is the only neighbour of it which has been inserted, we will not insert it again. Insert node D into QUEUE2.
QUEUE1 = {C, F}
QUEUE2 = { A, B, D}

5. Delete the node C from QUEUE1 and insert all its neighbours. Add node C to QUEUE2.
QUEUE1 = {F, E, G}
QUEUE2 = {A, B, D, C}

6. Remove F from QUEUE1 and add all its neighbours. Since all of its neighbours has already been added, we will not add them again. Add node F to QUEUE2.
QUEUE1 = {E, G}
QUEUE2 = {A, B, D, C, F}

7. Remove E from QUEUE1, all of E's neighbours has already been added to QUEUE1 therefore we will not add them again. All the nodes are visited and the target node i.e. E is encountered into QUEUE2.
QUEUE1 = {G}
QUEUE2 = {A, B, D, C, F, E}

Now, backtrack from E to A, using the nodes available in QUEUE2.
The minimum path will be **A → B → C → E**.

Time complexity

The time complexity is *O(V+E)*, where *V* is the number of vertices and *E* is the number of edges.

In the tree structured graph it can be wrutten as $O(b^d)$ where b is the branch factor (how many children one node may have) and d is the depth each level of the tree.

Space complexity

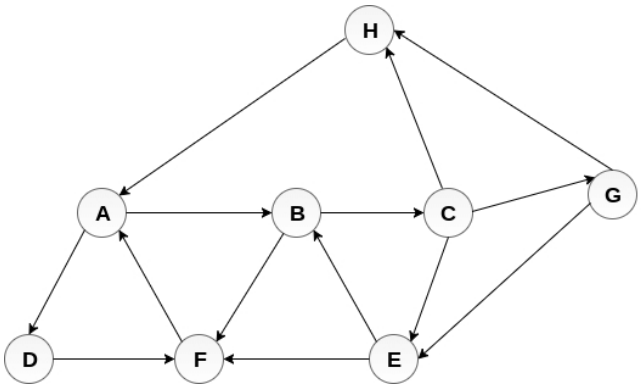
The space complexity is *O(h)*, where *h* is the maximum height of the tree.

What is DFS?

DFS is an algorithm for finding or traversing graphs or trees in depth-ward direction. The execution of the algorithm begins at the root node and explores each branch before backtracking. It uses a stack data structure to remember, to get the subsequent vertex, and to start a search, whenever a dead-end appears in any iteration. The full form of DFS is Depth-first search.

Rule 1 – Put the starting vertex into the stack.
Mark it as visited.
Display it.

Rule 2 – if(STACK[top] has adjacent unvisited vertex)
{
Visit the adjacent unvisited vertex and Mark it as visited.
push it into the STACK.
Display it.
}
else
{
pop top element of stack.
}
(repeat rule 2 until stack is empty)



Adjacency Lists

A : B, D
B : C, F
C : E, G, H
G : E, H
E : B, F
F : A
D : F
H : A

Solution :

Push H onto the stack

STACK : H

POP the top element of the stack i.e. H, print it and push all the neighbours of H onto the stack that are is ready state.

Print H

STACK : A

Pop the top element of the stack i.e. A, print it and push all the neighbours of A onto the stack that are in ready state.

Print A

Stack : B, D

Pop the top element of the stack i.e. D, print it and push all the neighbours of D onto the stack that are in ready state.

Print D

Stack : B, F

Pop the top element of the stack i.e. F, print it and push all the neighbours of F onto the stack that are in ready state.

Print F

Stack : B

Pop the top of the stack i.e. B and push all the neighbours

Print B
Stack : C
Pop the top of the stack i.e. C and push all the neighbours.
Print C
Stack : E, G
Pop the top of the stack i.e. G and push all its neighbours.
Print G
Stack : E
Pop the top of the stack i.e. E and push all its neighbours.
Print E
Stack :
Hence, the stack now becomes empty and all the nodes of the graph have been traversed.
The printing sequence of the graph will be :

H → A → D → F → B → C → G → E

Time complexity

The time complexity is *O(V+E)*, where *V* is the number of vertices and *E* is the number of edges.

In the tree structured graph it can be wrutten as $O(b^d)$ where b is the branch factor (how many children one node may have) and d is the depth each level of the tree.

Space complexity

The space complexity is *O(h)*, where *h* is the maximum height of the tree.

Which One Should You Choose: BFS or DFS?

The time complexity of both algorithms is the same. But in the case of space complexity, if the maximum height is less than the maximum number of nodes in a single level, then DFS will be more space optimised than BFS or vice versa.

So if the problem involves finding the nearest neighbour or the shortest path, BFS performs better — as in the case of DFS, leaf nodes are visited first.

But in the case of solving a puzzle or topological sorting (i.e., cases where the result needs to be evaluated on reaching leaf nodes), DFS performs better.

Applications of BFS

Un-weighted Graphs:

BFS algorithm can easily create the shortest path and a minimum spanning tree to visit all the vertices of the graph in the shortest time possible with high accuracy.

P2P Networks:

BFS can be implemented to locate all the nearest or neighboring nodes in a peer to peer network. This will find the required data faster.

Web Crawlers:

Search engines or web crawlers can easily build multiple levels of indexes by employing BFS. BFS implementation starts from the source, which is the web page, and then it visits all the links from that source.

Network Broadcasting:

A broadcasted packet is guided by the BFS algorithm to find and reach all the nodes it has the address for.

Applications of DFS

Weighted Graph:

In a weighted graph, DFS graph traversal generates the shortest path tree and minimum spanning tree.

Detecting a Cycle in a Graph:

A graph has a cycle if we found a back edge during DFS. Therefore, we should run DFS for the graph and verify for back edges.

Path Finding:

We can specialize in the DFS algorithm to search a path between two vertices.

Topological Sorting:

It is primarily used for scheduling jobs from the given dependencies among the group of jobs. In computer science, it is used in instruction scheduling, data serialization, logic synthesis, determining the order of compilation tasks.

Searching Strongly Connected Components of a Graph:

It used in DFS graph when there is a path from each and every vertex in the graph to other remaining vertexes.

Solving Puzzles with Only One Solution:

DFS algorithm can be easily adapted to search all solutions to a maze by including nodes on the existing path in the visited set.

KEY DIFFERNCES:

- BFS finds the shortest path to the destination whereas DFS goes to the bottom of a subtree, then backtracks.
- The full form of BFS is Breadth-First Search while the full form of DFS is Depth First Search.
- BFS uses a queue to keep track of the next location to visit. whereas DFS uses a stack to keep track of the next location to visit.
- BFS traverses according to tree level while DFS traverses according to tree depth.
- BFS is implemented using FIFO list on the other hand DFS is implemented using LIFO list.
- In BFS, you can never be trapped into finite loops whereas in DFS you can be trapped into infinite loops.

Difference between BFS and DFS Binary Tree

BFS	DFS
BFS finds the shortest path to the destination.	DFS goes to the bottom of a subtree, then backtracks.
The full form of BFS is Breadth-First Search.	The full form of DFS is Depth First Search.
It uses a queue to keep track of the next location to visit.	It uses a stack to keep track of the next location to visit.
BFS traverses according to tree level.	DFS traverses according to tree depth.
It is implemented using FIFO list.	It is implemented using LIFO list.
It requires more memory as compare to DFS.	It requires less memory as compare to BFS.
This algorithm gives the shallowest path solution.	This algorithm doesn't guarantee the shallowest path solution.
There is no need of backtracking in BFS.	There is a need of backtracking in DFS.
You can never be trapped into finite loops.	You can be trapped into infinite loops.
If you do not find any goal, you may need to expand many nodes before the solution is found.	If you do not find any goal, the leaf node backtracking may occur.