SNS COLLEGE OF ENGINEERING

Kurumbapalayam(Po), Coimbatore – 641 107
Accredited by NAAC-UGC with 'A' Grade
Approved by AICTE, Recognized by UGC & Affiliated to Anna University, Chennai

Department of Artificial Intelligence and Data Science

Course Name – Introduction to Artificial Intelligence

II Year / III Semester

Unit 1 Breadth and depth first

What is BFS Algorithm (Breadth-First Search)?

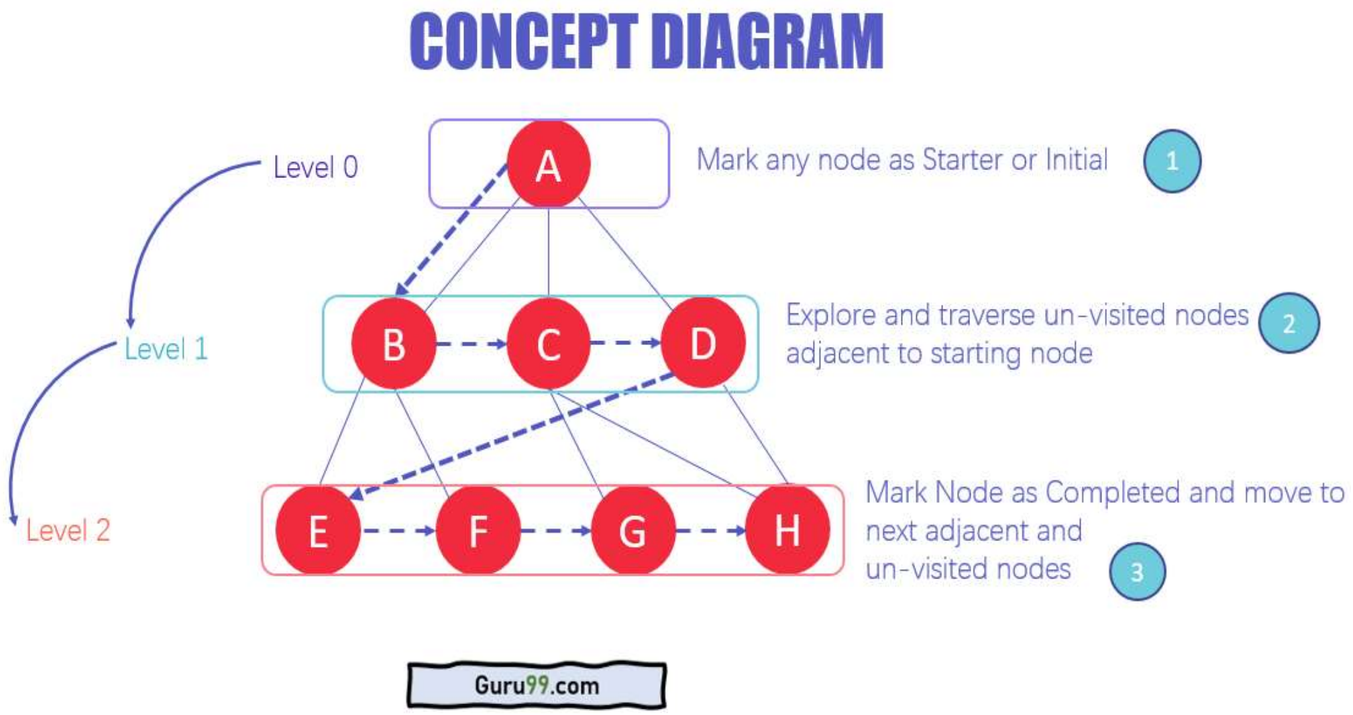
Breadth-first search (BFS) is an algorithm that is used to graph data or searching tree or traversing structures. The full form of BFS is the Breadth-first search.

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. Remember, BFS accesses these nodes one by one.

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.

What is Graph traversals?

A graph traversal is a commonly used methodology for locating the vertex position in the graph. It is an advanced search algorithm that can analyze the graph with speed and precision along with marking the sequence of the visited vertices. This process enables you to quickly visit each node in a graph without being locked in an infinite loop.



1.In the various levels of the data, you can mark any node as the starting or initial node to begin traversing. The BFS will visit the node and mark it as visited and places it in the queue.
2.Now the BFS will visit the nearest and un-visited nodes and marks them. These values are also added to the queue. The queue works on the FIFO model.
3.In a similar manner, the remaining nearest and un-visited nodes on the graph are analyzed marked and added to the queue. These items are deleted from the queue as receive and printed as the result.

Why do we need BFS Algorithm?

There are numerous reasons to utilize the BFS Algorithm to use as searching for your dataset. Some of the most vital aspects that make this algorithm your first choice are:

- BFS is useful for analyzing the nodes in a graph and constructing the shortest path of traversing through these.
- BFS can traverse through a graph in the smallest number of iterations.
- The architecture of the BFS algorithm is simple and robust.
- The result of the BFS algorithm holds a high level of accuracy in comparison to other algorithms.
- BFS iterations are seamless, and there is no possibility of this algorithm getting caught up in an infinite loop problem.

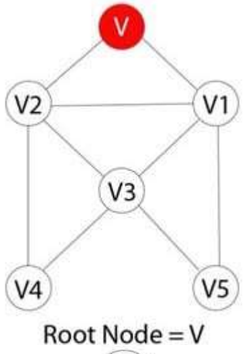
How does BFS Algorithm Work?

Graph traversal requires the algorithm to visit, check, and/or update every single un-visited node in a tree-like structure. Graph traversals are categorized by the order in which they visit the nodes on the graph.

BFS algorithm starts the operation from the first or starting node in a graph and traverses it thoroughly. Once it successfully traverses the initial node, then the next non-traversed vertex in the graph is visited and marked.

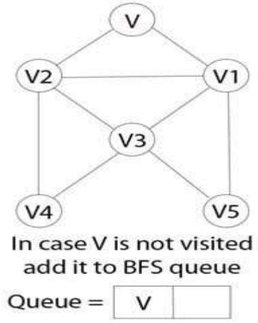
Hence, you can say that all the nodes adjacent to the current vertex are visited and traversed in the first iteration. A simple queue methodology is utilized to implement the working of a BFS algorithm, and it consists of the following steps:

Step 1)



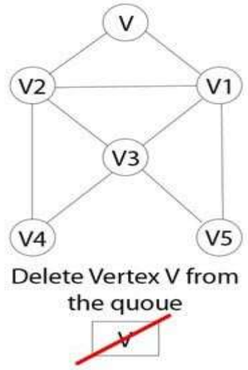
Each vertex or node in the graph is known. For instance, you can mark the node as V.

Step 2)



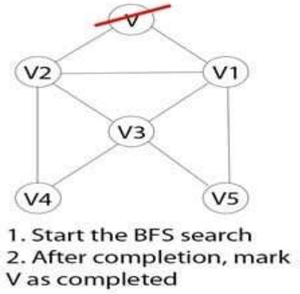
In case the vertex V is not accessed then add the vertex V into the BFS Queue.

Step 3)



Start the BFS search, and after completion, Mark vertex V as visited.

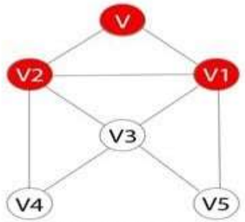
Step 4)



1. Start the BFS search
2. After completion, mark V as completed

The BFS queue is still not empty, hence remove the vertex V of the graph from the queue.

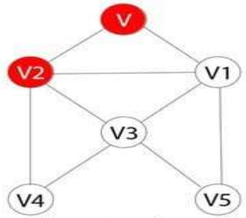
Step 5)



Visit and retrieve all the adjacent and un-visited nodes from the node V

Retrieve all the remaining vertices on the graph that are adjacent to the vertex V

Step 6)

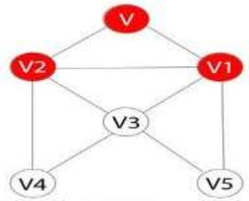


For adjacent and un-visited vertex say V1, add it to BFS queue

Queue = ~~V~~ | V1

For each adjacent vertex let's say V1, in case it is not visited yet then add V1 to the BFS queue

Step 7)



BFS will visit V1, mark it as visited and delete it from the queue

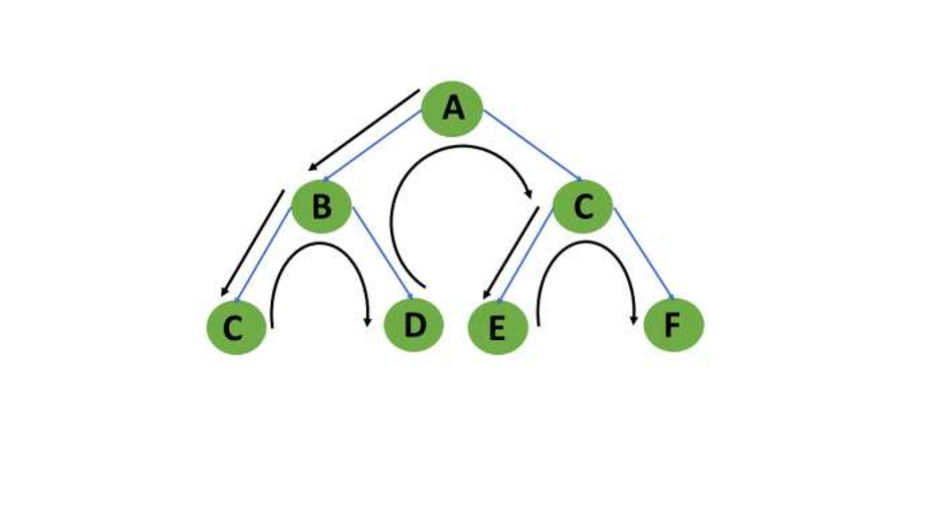
Queue = ~~V~~ | ~~V1~~

o For each adjacent vertex let's say V1, in case it is not visited yet then add V1 the BFS queue

Depth-First Search

The depth-first search or DFS algorithm raverses or explores data structures, such as trees and graphs. The algorithm starts at the root node (in the case of a graph, you can use any random node as the root node) and examines each branch as far as possible before backtracking.

When a dead-end occurs in any iteration, the Depth First Search (DFS) method traverses a network in a deathward motion and uses a stack data structure to remember to acquire the next vertex to start a search.



Example of Depth-First Search Algorithm

The outcome of a DFS traversal of a graph is a spanning tree. A spanning tree is a graph that is devoid of loops. To implement DFS traversal, you need to utilize a stack data structure with a maximum size equal to the total number of vertices in the graph.

To implement DFS traversal, you need to take the following stages.

Step 1: Create a stack with the total number of vertices in the graph as the size.

Step 2: Choose any vertex as the traversal's beginning point. Push a visit to that vertex and add it to the stack.

Step 3 - Push any non-visited adjacent vertices of a vertex at the top of the stack to the top of the stack.

Step 4 - Repeat steps 3 and 4 until there are no more vertices to visit from the vertex at the top of the stack.

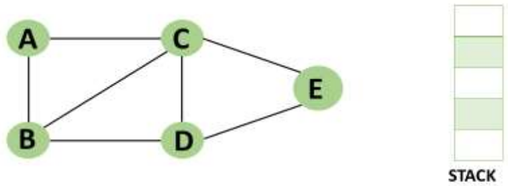
Step 5 - If there are no new vertices to visit, go back and pop one from the stack using backtracking.

Step 6 - Continue using steps 3, 4, and 5 until the stack is empty.

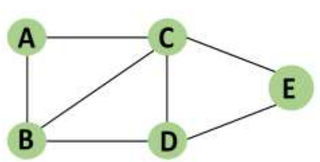
Step 7 - When the stack is entirely unoccupied, create the final spanning tree by deleting the graph's unused edges.

Consider the following graph as an example of how to use the dfs algorithm.

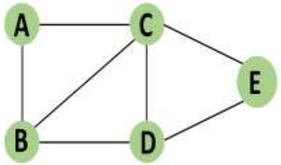
Step 1: Mark vertex A as a visited source node by selecting it as a source node.



You should push vertex A to the top of the stack.



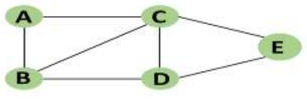
Step 2: Any nearby unvisited vertex of vertex A, say B, should be visited.

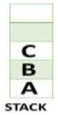


•You should push vertex B to the top of the stack.

Step 3: From vertex C and D, visit any adjacent unvisited vertices of vertex B. Imagine you have chosen vertex C, and you want to make C a visited vertex.

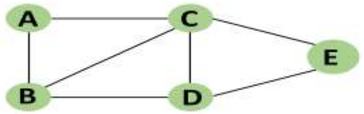
•Vertex C is pushed to the top of the stack.

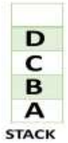


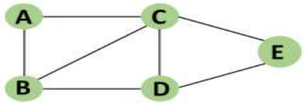


Step 4: You can visit any nearby unvisited vertices of vertex C, you need to select vertex D and designate it as a visited vertex.

•Vertex D is pushed to the top of the stack.

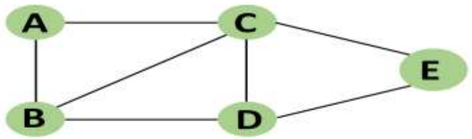




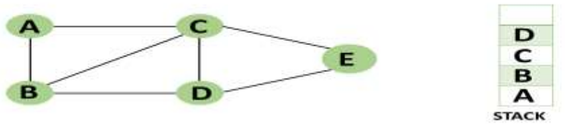


Step 5: Vertex E is the lone unvisited adjacent vertex of vertex D, thus marking it as visited.

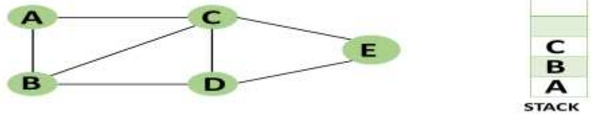
•Vertex E should be pushed to the top of the stack.



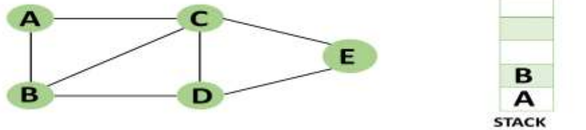
Step 6: Vertex E's nearby vertices, namely vertex C and D have been visited, pop vertex E from the stack.



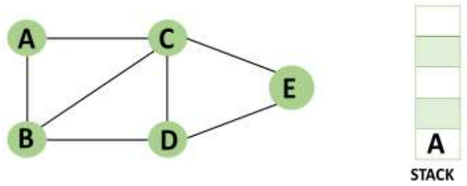
Step 7: Now that all of vertex D's nearby vertices, namely vertex B and C, have been visited, pop vertex D from the stack.



Step 8: Similarly, vertex C's adjacent vertices have already been visited; therefore, pop it from the stack.



Step 9: There is no more unvisited adjacent vertex of b, thus pop it from the stack.



Step 10: All of the nearby vertices of Vertex A, B, and C, have already been visited, so pop vertex A from the stack as well.