Breadth-First Search (BFS)

Breadth-First Search (BFS) is one of the most fundamental algorithms in computer science, especially in graph theory. This guide is written in a human, easy-to-understand style to help learners grasp the concept thoroughly.

What Is BFS?

Breadth-First Search is a graph traversal algorithm. It explores nodes level by level — starting from a source node, it visits all neighbors, then all neighbors' neighbors, and so on.

Think of BFS like ripples in water: starting from one point and expanding outward evenly.

Where BFS Is Used

BFS is incredibly useful and appears in many real-world applications:

- Finding the **shortest path** in an unweighted graph
- Solving puzzles like the **Rubik's Cube** or **word ladder problems**
- Web crawlers exploring pages level-by-level
- Social network friend recommendations (e.g., friends-of-friends)
- AI and game pathfinding (when costs are equal)

Key Concepts You Need to Know

Before jumping into BFS, here are the basics:

1. Graph

A graph is made of:

- **Nodes (or vertices)** — points
- **Edges** — lines connecting nodes

Graphs can be:

- **Directed or Undirected**
- **Weighted or Unweighted** (BFS assumes unweighted)

2. Queue

BFS uses a **queue (FIFO — First In, First Out)** to keep track of which node to visit next.

3. Visited Set

To avoid revisiting nodes (and looping forever), BFS keeps a record of visited nodes.

How BFS Works (Step-by-Step)

1. Start from a chosen **source node**.
2. Mark the source as **visited**.
3. Push (enqueue) it into a **queue**.
4. While the queue is not empty:
    - Dequeue the front node.
    - Visit all its **unvisited neighbors**.
    - Mark neighbors visited and enqueue them.

This continues until the queue becomes empty.

BFS Pseudocode

Below is a clean and simple pseudocode version:

```
BFS(Graph, start):
  create a queue Q
  create a set Visited

  add start to Q
  add start to Visited

  while Q is not empty:
    node = Q.dequeue()
    print(node)    // process the node

    for each neighbor in Graph[node]:
      if neighbor not in Visited:
        add neighbor to Visited
        Q.enqueue(neighbor)
```

What You Need to Implement BFS

To implement BFS in any programming language, you need:

A graph representation

Common ways:

- **Adjacency list** (best for BFS)
- Adjacency matrix

A queue

- Built-in queue (Python `deque`, Java `Queue`, C++ `queue`)
- Or you can implement your own

A visited structure

- A set (Python `set`, Java `HashSet`)
- A boolean array (common in competitive programming)

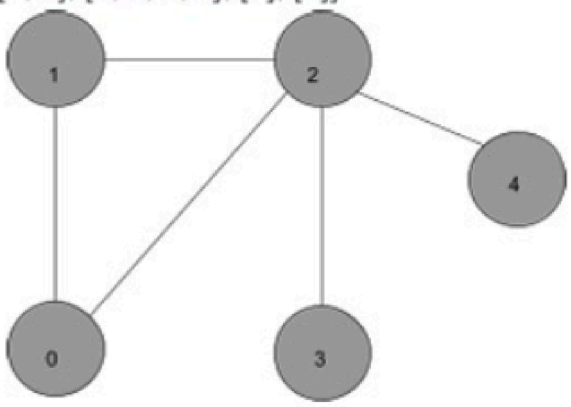
A starting node

Where the BFS begins.

Example (Simple Understanding)

Suppose we have this graph:

Input: *adj[][] = [[1, 2], [0, 2], [0, 1, 3, 4], [2], [2]]*



Output: *[0, 1, 2, 3, 4]*
Explanation: *Starting from 0, the BFS traversal proceeds as follows:*
Visit 0 - Print: 0
Visit 1 (neighbor of 0) - Print: 1
Visit 2 (next neighbor of 0) - Print: 2
Visit 3 (first neighbor of 2 that hasn't been visited yet) - Print: 3
Visit 4 (next neighbor of 2) - Print: 4

Final Thoughts

BFS is simple yet powerful. Mastering it helps you understand graph theory, shortest paths, and many real-world algorithms.

Once you're comfortable with BFS, you can move on to:

- Depth-First Search (DFS)
- Dijkstra’s Algorithm
- A* Search