BFS

(Breadth-First Search)

➢ What Is BFS?

BFS is a way to explore a graph step by step, level by level. You start from one node and visit all the neighbors close to it first, then you move outward.

Think of it like dropping a pebble in water the waves move outward in circles. BFS does the same in a graph.

➢ Main Ideas of BFS

1.Uses a Queue (FIFO): BFS uses a queue, meaning the first item added is the first to come out.

2. Keeps Track of Visited Nodes: We mark nodes as "visited" so we don't process the same node again.

3. Works Level by Level: BFS always explores the closest nodes first, then the next layer of nodes.

➢ What You Need

To use or write BFS, you need:

- A graph (usually stored as an adjacency list)
- A queue
- A visited list or set
- A starting node

➢ How BFS Works

1. You start at one node.
2. You look at all the nodes directly connected to it.
3. After finishing those, you move to the nodes that are one step farther.
4. You continue moving outward, layer by layer.
5. You stop when there are no more nodes left to explore.

Where BFS Is Used

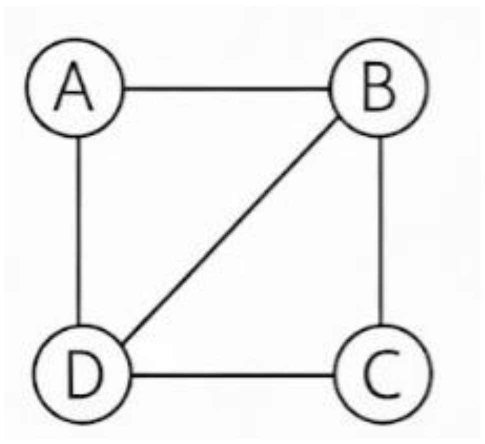
BFS is helpful in many real-life problems:

- Finding the shortest path in an unweighted graph
- Maze solving
- Social network friend suggestions
- Web crawling
- Networking and message broadcasting

Pseudocode:

```
BFS(G, s) {
    for each u in V {
        color[u] = white
        d[u] = infinity
        pred[u] = null
    }
    color[s] = gray
    d[s] = 0
    Q = {s}
    while (Q is nonempty) {
        u = Q.Dequeue()
        for each v in Adj[u] {
            if (color[v] == white) {
                color[v] = gray
                d[v] = d[u] + 1
                pred[v] = u
                Q.Enqueue(v)
            }
        }
        color[u] = black
    }
}
```

- **simple Diagrams for Understanding BFS**



BFS from node A:

Step 1: Start at A

Visited: A

Queue: [B, D]

Step 2: Visit B

Visited: A, B

Queue: [D, C]

Step 3: Visit D

Visited: A, B, D

Queue: [C]

Step 4: Visit C

Visited: A, B, D, C

Queue: []

BFS Order: A → B → D → C

- **Time Complexity**: O(V + E)
- **Space Complexity**: O(V)

- **After learning BFS:** you can traverse graphs, trees, and grids, find shortest paths in unweighted graphs, count levels or connected components, check if a graph is bipartite, solve maze or grid-based problems, and implement advanced BFS variations like multi-source BFS or BFS with path reconstruction.