Graphs – Breadth First Search (BFS)

1. Introduction

Breadth First Search (BFS) is a **graph traversal algorithm** used to visit all vertices of a graph **level by level**.

It explores all neighboring vertices first before moving deeper into the graph.

BFS is especially useful when we want to find the **shortest path** or explore nodes in **layers**.

---

2. What is BFS?

BFS starts from a **given source vertex** and:

- Visits all its immediate neighbors
- Then visits neighbors of neighbors
- Continues level by level until all reachable vertices are visited

It uses a **queue** data structure to maintain traversal order.

---

3. Key Characteristics of BFS

- Traverses graph **breadth-wise**
- Uses **queue (FIFO)**
- Visits nodes **level by level**
- Guarantees shortest path in **unweighted graphs**
- Can be applied to **graphs and trees**

---

4. BFS Working Principle (Concept)

1. Start from a source node

2. Mark it as visited
3. Add it to a queue
4. While queue is not empty:
    - Remove front node
    - Visit all unvisited adjacent nodes
    - Mark them visited and add to queue

---

5. BFS Example

Graph:

```
A —— B —— D
|    |
C    E
```

Start BFS from **A**

Traversal order:

```
A → B → C → D → E
```

Explanation:

- Visit A
- Visit neighbors B and C
- Visit neighbors of B (D, E)

---

6. Visualization of BFS Traversal

```
Level 0: A
Level 1: B, C
Level 2: D, E
```

BFS processes one level completely before moving to the next.

7. BFS Using Queue (Conceptual)

Queue state during traversal:

```
Start: [A]
After A: [B, C]
After B: [C, D, E]
After C: [D, E]
```

8. BFS Algorithm (Plain English)

1. Create a queue
2. Mark all vertices as unvisited
3. Enqueue the starting vertex
4. While queue is not empty:
    - Dequeue a vertex
    - Process it
    - Enqueue all its unvisited neighbors

9. Time and Space Complexity

Aspect	Complexity
Time Complexity	O(V + E)
Space Complexity	O(V)

Where:

- V = number of vertices
- E = number of edges

10. BFS vs DFS

Feature	BFS	DFS
Data Structure	Queue	Stack / Recursion
Traversal	Level-wise	Depth-wise
Shortest Path	Yes (unweighted)	No
Memory Usage	More	Less

11. Applications of BFS

- Shortest path in unweighted graphs
- Social networking (friend suggestions)
- Web crawling
- Network broadcasting
- Level order traversal in trees
- Finding connected components

12. Advantages of BFS

- Finds shortest path
- Simple logic
- Level-based exploration
- Useful in many real-world problems

13. Limitations of BFS

- Uses more memory
- Not efficient for deep graphs
- Queue size can grow large
- Slower than DFS in some cases

14. BFS in Trees vs Graphs

- In trees, BFS is called **Level Order Traversal**
- In graphs, BFS needs a **visited array** to avoid cycles

---

15. Summary

- BFS explores graph level by level
- Uses queue data structure
- Guarantees shortest path in unweighted graphs
- Time complexity is O(V + E)
- Widely used traversal algorithm
