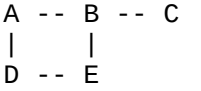
BFS Algorithm:

It acts as the foundation for many graph-based problems. It is mainly used to traverse or search through a graph or tree level by level — making it very efficient for finding the shortest path in unweighted graphs, solving maze problems, and even in social network analysis.

Core Idea of our Algorithm:

“Explores all neighbors of a node before moving to the next level of neighbors.”

Let’s understand with an example:



If we start BFS from node A, the traversal order will be:
A → B → D → C → E

It explores each node level by level, ensuring that all nodes at a certain distance are visited before moving further.

Data Structures Used:

Graph → To store nodes and edges (Adjacency List or Adjacency Matrix)

Queue → For storing the next node to visit (First-In-First-Out order)

Visited List / Set → To keep track of already visited nodes

Pseudocode:

```
BFS(graph, start):
    create a queue Q
    mark start as visited and enqueue it into Q

    while Q is not empty:
        node = Q.dequeue()
        process(node)
        for each neighbor in graph[node]:
            if neighbor not visited:
                mark neighbor as visited
                enqueue(neighbor)
```

Advantages of using this Algorithm:

Finds shortest path in unweighted graphs

Useful in social network analysis

Used in web crawlers

Helps in solving puzzles or mazes

Detects connected components in a graph