BFS – Breath First Search

Breadth-First Search (BFS) is a fundamental graph-traversal algorithm used to explore nodes level by level. It starts from a chosen source node and visits all its neighboring nodes first before moving on to the next level of neighbors. BFS uses a queue data structure to keep track of the order in which nodes should be visited. This ensures that the algorithm always explores the closest unvisited nodes first. It is commonly used for finding the shortest path in an unweighted graph, checking connectivity, and solving problems such as finding levels in a tree or network. Because it processes nodes in a systematic, layered manner, BFS is efficient, easy to implement, and widely applied in computer science.

Advantages of BFS

- **Finds the shortest path** in an unweighted graph because it explores nodes level by level.
- **Simple and easy to implement** using a queue.
- **Guarantees completeness**, meaning it will find a solution if one exists.
- **Works well for problems modeled as layers**, such as finding minimum steps or levels in a tree or graph.

Limitations of BFS

- **High memory usage**, because it stores all nodes of the current level in a queue.
- **Not efficient for large or infinite graphs**, as it may run out of memory quickly.
- **Can be slower** than other algorithms when the solution is far from the root.
- **Requires all edges to be equal weight** to guarantee shortest path results; otherwise, algorithms like Dijkstra's are needed.

Steps of BFS

1. **Start from a chosen source node**
   Select a starting vertex in the graph from where the traversal should begin.
2. **Mark the source node as visited**
   This prevents the algorithm from visiting the same node again.
3. **Insert the source node into a queue**
   BFS uses a **queue (FIFO)** to store nodes to be processed.
4. **Repeat the following until the queue becomes empty:**
   - **Remove (dequeue) the front node** from the queue.
   - **Visit the node** (process it or print it).
   - **Check all adjacent (neighboring) nodes** of the dequeued node.
   - For each neighbor:
     - If it is **not visited**, mark it as visited.
     - **Enqueue** the neighbor into the queue.
5. **Continue the process level by level**
   BFS systematically explores all nodes at the current level before moving to the next level.
6. **Stop when the queue is empty**
   This means all reachable nodes have been visited.

Pseudocode :

```
BFS(graph, start):
  create a queue Q
  create a visited array initialized to false

  mark visited[start] = true
  enqueue start into Q

  while Q is not empty:
    node = Q.front()
    Q.pop()

    print node   // or process node

    for each neighbor in graph[node]:
      if visited[neighbor] == false:
        visited[neighbor] = true
        Q.push(neighbor)
```

Time Complexity:

The time complexity of BFS is:

O(V+E)

where:

- V = number of vertices (nodes)
- E = number of edges

BFS CODE :

https://github.com/Hazra32/Algorithm-Problem/blob/main/BFS/Basic/basic.cpp