BFS

Breadth-First Search (BFS) is a graph traversal algorithm that visits nodes level by level. It starts from a source node, then first explores all the nodes directly connected to it, before moving on to the next level of nodes. To do this, BFS uses a queue data structure: the starting node is added to the queue, then one by one nodes are removed from the front, their unvisited neighbors are added to the back, and the process continues. BFS ensures that the shortest path (in terms of number of edges) from the source to every other reachable node is found. It is commonly used in shortest path problems for unweighted graphs, finding connected components, and solving puzzles or games where the goal is to reach a target in minimum steps.

Advantages of BFS

- **Finds the shortest path** in an unweighted graph.
- **Guarantees level-by-level traversal**, making it easy to understand the structure of the graph.
- **Useful for solving puzzles** (like 8-puzzle, shortest moves problems).
- **Detects connected components** in an undirected graph.
- **Works well for broadcast-type problems**, where you need to reach all nodes step by step.

Limitations of BFS

- **Consumes a lot of memory**, because it stores all nodes in the queue at each level.
- **Slow for very large graphs**, since it may explore many nodes unnecessarily.
- **Doesn’t work for weighted graphs** to find shortest paths (Dijkstra is needed instead).
- **Can be inefficient** when the target node is very deep in the graph.

Steps of BFS (Breadth-First Search)

1. **Choose a starting node**
   Select the node from where you want to begin the traversal.

2. **Mark the starting node as visited**
   This prevents visiting the same node again.

3. **Insert the starting node into a queue**
   BFS uses a **queue** (FIFO) to store nodes in the order they should be explored.

4. **Repeat until the queue becomes empty:**

   a. Remove a node from the front of the queue
   This is the current node you will process.

   b. Visit all unvisited neighbors of the current node
   Look at every node directly connected to the current node.

   c. Mark each unvisited neighbor as visited
   This ensures no repetition.

   d. Insert each visited neighbor into the queue
   They will be explored in upcoming steps.

5. **Continue the loop**
   Keep removing a node from the queue and exploring its neighbors until no nodes remain.

6. **Traversal complete**
   When the queue is empty, BFS has visited all reachable nodes in **level order**.

BFS pseudocode:

```
BFS(Graph, start):
 create an empty queue Q
 mark start as visited
 enqueue start into Q

while Q is not empty:
  current = dequeue Q
  output current

  for each neighbor of current:
    if neighbor is not visited:
      mark neighbor as visited
      enqueue neighbor into Q
```

Time Complexity: O(V + E)

Where:

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

BFS code:

https://github.com/shanto470/algorithm_plm/blob/main/BFS/basic/basic.cpp