

## Problem Analysis (BFS)

### BFS:

Breadth First Search (BFS) algorithm traverses a graph in a breadthward motion to search a graph data structure for a node that meets a set of criteria. It uses a queue to remember the next vertex to start a search, when a dead end occurs in any iteration.

Breadth First Search (BFS) algorithm starts at the tree root and explores all nodes at the present depth prior to moving on to the nodes at the next depth level.

### Process:

- **Rule 1** – Visit the adjacent unvisited vertex. Mark it as visited. Display it. Insert it in a queue.
- **Rule 2** – If no adjacent vertex is found, remove the first vertex from the queue.
- **Rule 3** – Repeat Rule 1 and Rule 2 until the queue is empty.

### BFS Algorithm Pseudocode

```
procedure BFS(G,s)
  for each vertex  $v \in V[G]$  do
     $explored[v] \leftarrow \text{false}$ 
     $d[v] \leftarrow \infty$ 
  end for
   $explored[s] \leftarrow \text{true}$ 
   $d[s] \leftarrow 0$ 
   $Q :=$  a queue data structure, initialized with  $s$ 
  while  $Q \neq \phi$  do
     $u \leftarrow$  remove vertex from the front of  $Q$ 
    for each  $v$  adjacent to  $u$  do
      if not  $explored[v]$  then
         $explored[v] \leftarrow \text{true}$ 
         $d[v] \leftarrow d[u] + 1$ 
        insert  $v$  to the end of  $Q$ 
      end if
    end for
  end while
end procedure
```

### BFS – Solution Approach (Compact Version)

1. **Build structure**  
Use adjacency list (graph) or matrix (grid) to access neighbors.

## 2. Initialize BFS

- Create a **queue**
- Mark **start** as visited
- Push start into queue

## 3. Process queue

While queue is not empty:

- Pop current node
- Visit all its neighbors
- If neighbor is **not visited** → mark visited and enqueue

## 4. Shortest path case

Use:

- distance array
  - or parent array
  - or store path with each queue entry
- BFS guarantees shortest path in **unweighted** graphs.

## 5. Stop early (optional)

If target is found, return answer immediately.

## 6. Output

Return traversal, reachable check, distance, or path depending on problem.

## Time Complexity

Scenario	Time Complexity	Space Complexity	Reason
Graph (V vertices, E edges)	$O(V + E)$	$O(V)$	Each vertex visited once; each edge checked once
Tree (N nodes)	$O(N)$	$O(N)$	Tree has $N-1$ <u>edges</u> → linear traversal
Grid $N \times M$	$O(N \times M)$	$O(N \times M)$	Every cell processed once; 4-direction check is constant
Shortest Path (Unweighted Graph)	$O(V + E)$	$O(V)$	BFS finds shortest path in linear time
Adjacency Matrix Graph	$O(V^2)$	$O(V)$	You must scan all $V$ <u>neighbors</u> for each node