Breadth-First Search (BFS) Algorithm with Time Complexity Analysis

Algorithm 1 Breadth-First Search(G, s)

```
 1: Input: Graph G = (V, E), source vertex s
 2: Output: Shortest path distances d[v] from s to all v ∈ V, BFS tree
 3: function BFS(G, s)
 4:     Initialize:
 5:     for all v ∈ V do                                  ▷ Initialization of all vertices
 6:         d[v] ← ∞                                      ▷ Set distance to infinity
 7:         π[v] ← NIL                                    ▷ No predecessor
 8:         visited[v] ← FALSE                            ▷ Vertex not visited
 9:     end for
10:     d[s] ← 0                                          ▷ Distance to the source is zero
11:     visited[s] ← TRUE                                 ▷ Mark source vertex as visited
12:     Q ← Empty Queue
13:     Enqueue(Q, s)                                     ▷ Start BFS from the source
14:     while Q ≠ Empty do                                ▷ Process all vertices in the queue
15:         u ← Dequeue(Q)                                ▷ Remove and process the front vertex
16:         for all v ∈ G.Adj[u] do                       ▷ Iterate over all neighbors of u
17:             if visited[v] = FALSE then                ▷ If vertex v is not visited
18:                 visited[v] ← TRUE                     ▷ Mark v as visited
19:                 d[v] ← d[u] + 1                       ▷ Update distance to v
20:                 π[v] ← u                              ▷ Set u as predecessor of v
21:                 Enqueue(Q, v)                         ▷ Add v to the queue for processing
22:             end if
23:         end for
24:     end while
25: end function
```

Time Complexity Analysis

Initialization (Lines 2–10):

- Each vertex $v \in V$ is initialized with $\mathcal{O}(1)$ operations (distance, predecessor, and visited status).
- Total cost: $\mathcal{O}(V)$.

Main BFS Loop (Lines 12–21):

- The while-loop runs once for each vertex in the queue.
- Enqueuing and dequeuing operations each take $\mathcal{O}(1)$ for a single vertex.
- For each vertex u, the inner for-loop iterates over all adjacent vertices $v \in G.Adj[u]$.
- The sum of all adjacency list iterations across all vertices is $\mathcal{O}(E)$, where $|E|$ is the number of edges.
- Total cost of the main loop: $\mathcal{O}(V + E)$.

Overall Time Complexity:

$$\mathcal{O}(V + E)$$

Explanation

- The algorithm processes each vertex once, leading to a cost of $\mathcal{O}(V)$ for vertices.
- It also processes each edge once during adjacency list traversal, leading to a cost of $\mathcal{O}(E)$ for edges.
- BFS is efficient for sparse graphs, where $E \approx V$, as the complexity simplifies to $\mathcal{O}(V)$.

Notes on BFS

- BFS finds the shortest path in an unweighted graph.
- The BFS tree is formed using the predecessor array $\pi[v]$, representing the parent-child relationships in the traversal.
- The ‘visited‘ array ensures that each vertex is processed only once, avoiding redundant visits.