Breadth-First Search: A Decrease and Conquer Approach

Sumanth

June 18, 2025

Abstract

Breadth-First Search (BFS) is a fundamental **decrease and conquer** graph traversal algorithm that explores vertices in layers, visiting all neighbors at the current depth before proceeding to the next level. Using a queue, it is ideal for finding shortest paths in unweighted graphs and other graph-related problems. This document details the algorithm's formulation, pseudocode, complexity analysis, practical applications, and implementation considerations, emphasizing its foundational role in graph algorithms, as noted by the contributor.

1 Introduction

Breadth-First Search (BFS) is a **decrease and conquer** algorithm for traversing or searching a graph, exploring vertices in a layered manner starting from a source vertex. By using a queue to manage the exploration order, BFS ensures that all vertices at distance k from the source are visited before those at distance $k+1$. Its ability to find shortest paths in unweighted graphs and its versatility make it a cornerstone of graph algorithms, as highlighted by the contributor.

2 Problem Formulation

Given a graph $G = (V, E)$ with $|V| = n$ vertices and $|E| = m$ edges, and a source vertex s, traverse all reachable vertices in G starting from s. Optionally, compute:

- The shortest path (in terms of edges) from s to each reachable vertex.
- The distance (number of edges) from s to each vertex.

The output includes the order of visited vertices and, if required, shortest paths or distances.

2.1 Key Idea

BFS explores the graph in layers using a queue to maintain vertices to be visited. It starts with the source vertex, enqueues its unvisited neighbors, and processes vertices in the order they are dequeued, ensuring a breadth-first traversal. The algorithm marks vertices as visited to avoid cycles and can track parent vertices for path reconstruction.

3 Breadth-First Search Algorithm

The algorithm uses a queue and a visited array for traversal. Below is the pseudocode, including shortest path computation:

Algorithm 1 Breadth-First Search Algorithm

Input: Graph $G = (V, E)$, source vertex s
Output: Visited vertices, distances dist, parents parent
Initialize queue Q
Initialize visited$[v] \leftarrow$ false for all $v \in V$
Initialize dist$[v] \leftarrow \infty$ for all $v \in V$
Initialize parent$[v] \leftarrow -1$ for all $v \in V$
visited$[s] \leftarrow$ true, dist$[s] \leftarrow 0$
Q.enqueue(s)
while Q is not empty **do**
    $u \leftarrow Q$.dequeue$()$
    Output u ▷ Process vertex
    for each neighbor v of u in G **do**
        if visited$[v] =$ false **then**
            visited$[v] \leftarrow$ true
            dist$[v] \leftarrow$ dist$[u] + 1$
            parent$[v] \leftarrow u$
            Q.enqueue(v)
        end if
    end for
end while
Return dist, parent

3.1 Shortest Path Reconstruction

The shortest path from source s to vertex v is reconstructed by backtracking through the parent array from v to s, collecting vertices in reverse order.

3.2 Cycle Detection

In undirected graphs, BFS can detect cycles by checking if a visited neighbor v of vertex u is not the parent of u. A cycle exists if such a neighbor is found.

4 Complexity Analysis

- **Time Complexity**: $O(n+m)$, where $n = |V|$ and $m = |E|$, as each vertex and edge is processed at most once.
- **Space Complexity**: $O(n)$, for the queue, visited array, distance array, and parent array.

5 Practical Applications

BFS is used in:

- **Shortest Path**: Finding the minimum number of edges between vertices in unweighted graphs (e.g., maze solving).
- **Web Crawling**: Exploring web pages by following links in a layered manner.
- **Social Networks**: Suggesting friends by finding users within a certain distance.
- **Connected Components**: Identifying all reachable vertices in disconnected graphs.
- **Cycle Detection**: Detecting cycles in undirected graphs for graph analysis.

6 Example

Consider an undirected graph with vertices $\{1,2,3,4\}$ and edges $\{(1,2),(1,3),(2,4),(3,4)\}$, starting from vertex 1:

Vertex	1	2	3	4
Distance	0	1	1	2
Parent	-1	1	1	2

BFS visit order: $1,2,3,4$. The shortest path from 1 to 4 is $1 \to 2 \to 4$.

7 Limitations and Extensions

- **Limitations**:
  - Ineffective for weighted graphs (use Dijkstra's or Bellman-Ford for shortest paths).
  - Memory-intensive for very large graphs due to queue storage.
- **Extensions**:
  - **Bidirectional BFS**: Run BFS from both source and target to reduce search space.
  - **Parallel BFS**: Distribute exploration across processors for large graphs.
  - **Level Tracking**: Explicitly track layers for applications like bipartite graph detection.

8 Implementation Considerations

- **Graph Representation**: Use an adjacency list for sparse graphs ($O(n+m)$) or adjacency matrix for dense graphs ($O(n^2)$).
- **Queue Implementation**: Use a standard queue (e.g., `std::queue` in C++) for FIFO processing.
- **Visited Tracking**: Use a boolean array or set to avoid revisiting vertices, especially in cyclic graphs.

- **Path Reconstruction**: Maintain a parent array to enable shortest path recovery.
- **Input Validation**: Ensure the source vertex is valid and handle disconnected graphs appropriately.

9 References

- GeeksForGeeks: Breadth-First Search
- Wikipedia: Breadth-First Search