**Documentation: Breadth-First Search (BFS) Traversal**

This document explains two implementations of the **Breadth-First Search (BFS)** algorithm for graph traversal.  
BFS is used to explore nodes in a graph level by level, starting from a given source node. It visits all the immediate neighbors of a node before moving to the next level of nodes.

Both versions in this program perform the same task — visiting all reachable vertices from the starting node — but they differ slightly in how they manage the list of nodes to visit next.

**1. BFS Without Queue**

**Description**

In the first version, the algorithm performs BFS **without using an explicit queue data structure**.  
Instead, it uses a simple list named container to store nodes that need to be visited. The first element of the list always represents the next node to process.

**Working Process**

1. Start by inserting the starting node into the container.
2. While the container is not empty:

Take the first element of the container (similar to dequeue operation).

If it has not been visited yet:

Print or process it.

Mark it as visited.

Add all unvisited neighboring nodes to the end of the container.

Repeat until the container becomes empty.

**Example Execution**

For the given graph:

A → B, C

B → D, E

C → F

Starting from **A**, the traversal order will be:  
**A B C D E F**

**Key Points**

* The algorithm works like normal BFS but uses **list slicing** instead of queue.pop(0) for removing the first element.
* Although it performs correctly, slicing a list creates a **new list each time**, which is less efficient for large graphs.
* Time complexity remains **O(V + E)**, but the slicing operation adds overhead.

**2. BFS Using Queue**

**Description**

The second version implements BFS using a **queue**, which is the standard and more efficient approach.  
The queue follows the **First-In, First-Out (FIFO)** principle, ensuring nodes are processed in the order they were discovered.

**Working Process**

1. Start by placing the starting node into the queue.
2. While the queue is not empty:
   * Remove the front node (dequeue operation).
   * If it is not already visited:
     + Print or process it.
     + Mark it as visited.
     + Enqueue all its unvisited neighbors at the end of the queue.
3. Continue until the queue is empty.

**Example Execution**

Using the same graph and starting from **A**, the traversal order remains:  
**A B C D E F**

**Key Points**

* This is the **standard BFS algorithm** and performs efficiently for large graphs.
* Time complexity: **O(V + E)**
* Space complexity: **O(V)** due to the queue and visited set.

**Conclusion :**

Both programs successfully demonstrate how **Breadth-First Search (BFS)** explores a graph level by level starting from a chosen vertex.

* The first approach (without queue) is simpler but less efficient.
* The second approach (with queue) is the standard BFS algorithm used in most graph problems.

This program illustrates the key concepts of graph traversal, including how to maintain a visited set, handle adjacency lists, and process nodes in a systematic order.