Graphs

This C++ file is a foundational demonstration of the two most important graph traversal algorithms: **Breadth-First Search (BFS)** and **Depth-First Search (DFS)**. The graph itself is represented using a classic **Adjacency Matrix**, a 2D array where the presence of an edge between two nodes is marked with a 1. 🗺️

The file perfectly contrasts the two exploration strategies:

* **BFS** uses a **Queue** data structure to explore the graph level by level. It visits all of a node's immediate neighbors before moving on to the next level of neighbors. Think of it like exploring all your immediate friends on a social network before moving on to *their* friends.
* **DFS**, in contrast, uses **recursion** (which implicitly uses the program's call stack) to explore as deeply as possible down one path before it backtracks to explore another. It's like navigating a maze by always taking the first available turn until you hit a dead end, then backtracking to try the next turn.

**BFS (Breadth-First Search)**

This function performs a **Breadth-First Search**, a "level-by-level" exploration of the graph. It's guaranteed to find the shortest path (in terms of the number of edges) from the start node to any other node.

* int visited[7] = {0}; This array acts as our **checklist** or "guest book." It keeps track of which nodes have already been visited to ensure we don't process the same node twice or get stuck in an infinite loop in graphs with cycles.
* cout << i << " "; visited[i] = 1; enqueue(i); The process kicks off by visiting the start node. It is immediately printed, marked as visited, and then added to the end of a **queue**. The queue holds the nodes that are "waiting" to have their neighbors explored.
* while (!isEmpty()) { i = dequeue(); ... } This is the main engine of BFS. The loop continues as long as there are nodes in the queue waiting to be processed. In each iteration, we **dequeue** a node i (the one that has been waiting the longest) to explore its neighbors.
* if (G[i][j] == 1 && visited[j] == 0) { ... } For the dequeued node i, this inner loop checks all other nodes j to find its **adjacent neighbors**. If there's an edge from i to j (G[i][j] == 1) and that neighbor j hasn't been visited yet, it is immediately visited (printed, marked) and **enqueued**, so its own neighbors can be explored in a future iteration.

**DFS (Depth-First Search)**

This function performs a **Depth-First Search**, a "dive-deep" exploration strategy. It goes as far down one path as it can before backtracking. This is naturally implemented using recursion.

* static int visited[7] = {0}; The static keyword is key here. It ensures that the visited array is initialized only **once** across all the recursive calls of the function. Without static, the visited array would be reset to zeros with every recursive call, breaking the algorithm.
* if (visited[start] == 0) { ... } This is the **base case** for a given node. The code only processes a node if it hasn't been visited before. This is what prevents the algorithm from getting stuck in infinite loops.
* for (j = 1; j < n; j++) { if (G[start][j] == 1 && visited[j] == 0) DFS(G, j, n); } This is the core of the recursive logic. After visiting the start node, it loops through all its potential neighbors. For the **first unvisited neighbor** it finds, it immediately makes a **recursive call** DFS(G, j, n). This is the "deep dive" — the algorithm will fully explore this new path starting from j before it ever returns to finish the rest of this for loop and check the other neighbors of start.

**Main Function**

This is the control center that defines the specific graph we are working with and initiates the traversals from a designated starting point.

* int G[7][7] = { ... }; This is the **Adjacency Matrix** representation of the graph. A 1 at G[i][j] signifies a direct, unweighted edge from node i to node j. A 0 means there is no direct edge. For example, G[1][2] = 1 means there's an edge from node 1 to node 2.
* DFS(G, 4, 7); and BFS(G, 4, 7); These lines kick off the traversals. Both algorithms are told to begin their exploration of the graph G starting from **node 4**. The 7 indicates the size of the matrix (number of nodes + 1, since it's 1-indexed).