### Programming Guide for “Graphs”

#### 1. Graph Class Implementation

**Overview:** A graph is a data structure that consists of a set of nodes (vertices) and a set of edges that connect pairs of nodes.

**Pseudo Code for Graph class:**

CLASS Graph  
 FUNCTION \_\_init\_\_(vertices)  
 INITIALIZE vertices  
 INITIALIZE graph as a list of empty lists for adjacency  
  
 FUNCTION add\_edge(u, v)  
 APPEND v to graph[u] and u to graph[v]  
  
 FUNCTION bfs(start\_vertex)  
 INITIALIZE visited array  
 CREATE queue and enqueue start\_vertex  
 WHILE queue is not empty  
 DEQUEUE vertex from queue  
 FOR each neighbor of vertex  
 IF neighbor is not visited  
 MARK as visited and ENQUEUE neighbor  
 RETURN traversal order  
  
 FUNCTION dfs(start\_vertex)  
 INITIALIZE visited array  
 CALL recursive helper function dfs\_util  
 RETURN traversal order  
END CLASS

#### 2. Breadth-First Search (BFS)

**Overview:** BFS is a traversing algorithm that starts traversing from a selected node and explores all the neighboring nodes at the present depth before moving on to the nodes at the next depth level.

**Pseudo Code for bfs:**

FUNCTION bfs(start\_vertex)  
 CREATE queue and enqueue start\_vertex  
 MARK start\_vertex as visited  
 WHILE queue is not empty  
 DEQUEUE vertex from queue  
 PROCESS vertex  
 FOR each neighbor of vertex  
 IF neighbor is not visited  
 MARK as visited and ENQUEUE neighbor  
END FUNCTION

#### 3. Depth-First Search (DFS)

**Overview:** DFS is an algorithm for traversing or searching tree or graph data structures. It starts at the root and explores as far as possible along each branch before backtracking.

**Pseudo Code for dfs:**

FUNCTION dfs(start\_vertex)  
 CREATE a visited array  
 DEFINE a recursive helper function dfs\_util  
 CALL dfs\_util for start\_vertex  
 RETURN traversal order  
  
FUNCTION dfs\_util(vertex)  
 MARK vertex as visited  
 PROCESS vertex  
 FOR each neighbor of vertex  
 IF neighbor is not visited  
 CALL dfs\_util(neighbor)

#### 4. Shortest Path Algorithm

**Overview:** The shortest path algorithm finds the shortest path from a start vertex to an end vertex in a graph.

**Pseudo Code for shortest\_path:**

FUNCTION shortest\_path(graph, start\_vertex, end\_vertex)  
 INITIALIZE visited array and prev array  
 CREATE queue and enqueue start\_vertex  
 MARK start\_vertex as visited  
 WHILE queue is not empty  
 DEQUEUE vertex from queue  
 FOR each neighbor of vertex  
 IF neighbor is not visited  
 MARK as visited  
 UPDATE prev array  
 ENQUEUE neighbor  
 RECONSTRUCT path from end\_vertex to start\_vertex using prev array  
 RETURN path  
END FUNCTION

**Implementation Tips:** - In BFS and DFS, use a list to track visited vertices. - For BFS, use a queue to explore vertices in order of their discovery. - For DFS, use recursion or a stack to explore as far as possible along a branch before backtracking. - For finding the shortest path, BFS works well for unweighted graphs. For weighted graphs, algorithms like Dijkstra’s or A\* would be more suitable.