Graph Class Hierarchy:

In this assignment, we have designed a Graph class hierarchy in Python. The parent class is a generic `Graph` class that encapsulates basic graph operations. It serves as the foundation for two specialized child classes.

Child Class 1 - Traversal: This class is responsible for traversing the graph using Breadth First Search (BFS) and Depth First Search (DFS). The unique feature here is the ability to output all paths in the graph where the value of the next node is higher than the previous node. The source node is provided as input, and the traversal results are stored for later analysis.

Child Class 2 - Adjacent: This class specializes in providing information about directly connected edges for a given input edge. It takes an edge as input and outputs the directly connected edges. This can be useful for understanding the local structure of the graph.

Main File: The main file serves as the entry point for utilizing the graph classes. Users can input any graph of their choice, specifying nodes and edges. It demonstrates the capabilities of the Graph class hierarchy, showcasing BFS, DFS, and edge exploration functionalities.

Phase 2 - Database Integration:

The second phase involves setting up a PostgreSQL database and utilizing it to store and retrieve graph data. The `graph` table is created to store edge information with columns `from_node` and `to_node`. The code connects to the database using PostgreSQL and reads the graph data from the database.

The results of the graph algorithms are stored in a `graph_results` table, which has columns `id`, `algo_name`, and `results`. The `results` column is a JSON object storing the traversal outcomes. Unit tests are implemented to validate the correctness of each traversal and graph operation.

Description:

This assignment showcases a comprehensive approach to graph manipulation in Python, covering fundamental graph operations and their application to real-world scenarios. The implementation is designed to be modular, extensible, and efficient. The integration with a PostgreSQL database adds a layer of persistence, enabling the storage and retrieval of graph data, making it suitable for large-scale graph processing.

The GitHub repository contains well-documented code with detailed explanations, ensuring ease of understanding and accessibility for developers. The unit tests included in the project validate the correctness of the implemented algorithms, providing a robust foundation for further development and expansion. The project is a valuable resource for anyone interested in graph theory, database integration, and algorithmic exploration.