**Describe the data structure you used to implement the graph and why.**

- Efficiency for Sparse Graphs: The adjacency list is a memory-efficient choice for sparse graphs, where only a small subset of possible edges actually exists. It stores only existing edges, unlike an adjacency matrix, which would store all potential edges, regardless of whether they are present or not.

- Fast Access to Neighboring Nodes: The adjacency list structure makes it easy to access the neighbors of any given node, which is essential for traversal-based algorithms like PageRank that require examining each node's outgoing edges.

- Edge Weight Storage: Since PageRank requires weights for each edge (based on the out-degrees of nodes), using a pair<int, double> in the vector allows for efficient storage of both the neighbor's ID and the weight associated with the edge.

- Ease of Implementation: map and vector allow for straightforward management of graph nodes and edges. The map provides fast lookups and insertions, while vector allows easy addition of edges to each node.

**What is the computational complexity of each method in your implementation in the worst case in terms of Big O notation?**

**add**(int source, int target): O(log V)
**urls**(): O(1)
**assign_id**(const string& link): O(log V)
**show_ids**(): O(V)
**degrees**(): O(1)
**increment_degree**(const string& link): O(log V)
**set_weights**(): O(V + E)
**calculate_rank**(int num_iterations): O(num_iterations * (V + E))
**rank**(): O(1)
**display_rank**(): O(V log V)

**What is the computational complexity of your main method in your implementation in the worst case in terms of Big O notation?**

O(E * log V + iteration_count * (V + E))

**What did you learn from this assignment, and what would you do differently if you had to start over?**

This assignment provided valuable insights into graph representation with adjacency lists and practical experience with implementing PageRank. It underscored the importance of using efficient data structures to handle complex graph data, especially with nodes, edges, and weighted connections.

Things I would change / Improve

1. Increase modularity by isolating specific functionalities, such as extracting the PageRank logic into its own function or class.
2. Incorporate thorough error-checking mechanisms to validate input data properly.
3. Optimize the weight-setting step to reduce redundant lookups, improving the code's efficiency for handling larger graphs.