Project 2: Simplified PageRank Algorithm

This document outlines the design, computational complexity, and reflections on the implementation of the Simplified PageRank algorithm.

**1. Graph Data Structure and Justification**

The web graph is implemented using an Adjacency List data structure, comprising:

* **std::map<std::string, int> url\_to\_id**: Maps URLs to unique integer IDs.
* **std::vector<std::string> id\_to\_url**: Maps integer IDs back to URL strings.
* **std::vector<std::vector<int>> adj**: The core adjacency list, where adj[i] contains IDs of pages that page i links to.
* **std::vector<int> out\_degree**: Stores the count of outgoing links for each page.

Justification: Adjacency Lists are ideal for sparse web graphs, saving significant memory compared to adjacent matrices. They enable efficient iteration over a page's outgoing links during rank calculation and support dynamic sizing as new URLs are discovered.

**2. Computational Complexity of Each Method**

Let V be the number of unique webpages (vertices), E be the number of directed links (edges), and p be the number of power iterations.

* **AdjacencyList() (Constructor):** O(1). Initializes member variables.
* **getOrCreateId(const std::string& url):** O(logV). Dominated by std::map lookup/insertion. std::vector::push\_back is amortized O(1), and std::vector::resize (if triggered) is amortized O(1) over the total V insertions.
* **addEdge(const std::string& from\_url, const std::string& to\_url):** O(logV). Dominated by two getOrCreateId calls.
* **PageRank(int powerIterations):** O(p⋅E+VlogV).
  + PageRank Iterations: The main loop runs p times. Each iteration involves traversing all edges and vertices to distribute ranks based on incoming links according to the simplified formula. This results in O(V+E), simplifying to O(E). Total: O(p⋅E).
  + Preparing Results: Populating the std::map for final sorted output involves V insertions, each O(logV). Total: O(VlogV).
  + Overall: The sum of these dominant phases.

**3. Computational Complexity of the Main Method (runProgram)**

Let N be the number of lines of input (equivalent to E, the number of edges).

* **runProgram():** O(NlogV+p⋅E+VlogV).
  + Graph Construction: Reading N input lines and calling addEdge for each takes O(NlogV).
  + PageRank Calculation: Calling AdjacencyList::PageRank(p) takes O(p⋅E+VlogV).
  + Overall: The total complexity is the sum of these two phases.

**4. What I Learned and What I Would Do Differently**

**What I Learned:**

* **Core PageRank Mechanics:** Gained a clear understanding of iterative rank distribution, including the nuances of the *simplified* formula and how it differs from standard PageRank.
* **Adjacency List Practicality:** Solidified practical graph representation in C++ using adjacency lists.
* **Algorithm Efficiency & Data Structure Impact:** Enhanced my ability to analyze computational complexity and reinforced the importance of appropriate data structure selection for performance.
* **Output Management:** Learned the criticality of careful output aggregation (e.g., using std::map for unique, sorted results) to meet specific formatting requirements.

**What I Would Do Differently if I Had to Start Over:**

* **std::unordered\_map for URL Mapping:** Use std::unordered\_map for url\_to\_id to improve average-case lookup/insertion to O(1), potentially speeding up graph construction.
* **Modular Design for PageRank Variants:** Design the PageRank method with more modularity (e.g., helper functions for iteration steps) for better readability and extensibility.
* **Enhanced Input Validation:** Add more robust checks for input data validity to improve program resilience in a production environment.
* **Precise Algorithm Implementation:** Emphasize upfront understanding of the exact mathematical PageRank formulation (as provided in the assignment) to avoid misinterpretations and ensure correct numerical results.