Project 2: Simplified PageRank Algorithm

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

**1. Description of the 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 for efficient lookup.
* **std::vector<std::string> id\_to\_url**: Provides the reverse mapping from integer IDs back to URL strings, useful for output.
* **std::vector<std::vector<int>> adj\_list**: The core adjacency list, where adj\_list[i] contains the IDs of pages that page [i] links to.
* **std::vector<int> out\_degrees**: Stores the count of outgoing links for each page, essential for distributing PageRank mass.

**Justification:** The Adjacency List is ideal for web graphs because they are typically sparse (few connections relative to total possible connections), saving significant memory compared to an adjacency matrix. It allows for efficient iteration over a page's neighbors during rank calculation and supports 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 Class Methods:**

* **AdjacencyList() (Constructor):**
  + **Complexity:** O(1)
  + **Explanation:** Initializes member variables.
* **get\_id(const std::string& url):**
  + **Complexity:** O(logV)
  + **Explanation:** std::map lookup/insertion is logarithmic with respect to the number of URLs.
* **add\_edge(const std::string& from\_url, const std::string& to\_url):**
  + **Complexity:** O(logV)
  + **Explanation:** Dominated by two get\_id calls; vector operations are amortized O(1).
* **PageRank(int p\_iterations):**
  + **Complexity:** O(V+p⋅E+VlogV)
  + **Explanation:**
    - **Initialization:** O(V) for rank vectors.
    - **Power Iterations:** The main loop runs p−1 times. Each iteration involves iterating through all edges to distribute rank, taking O(E) time, plus O(V) for resetting/copying rank vectors. Total: O(p⋅(V+E)), which simplifies to O(p⋅E) as E is typically ≥V.
    - **Sorting Results:** O(VlogV) for sorting the final PageRanks by URL.
    - **Overall:** The dominant term is O(p⋅E).

**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():**
  + **Complexity:** O(NlogV+p⋅E)
  + **Explanation:**
    - **Graph Construction:** Reading N input lines and calling add\_edge for each takes O(NlogV).
    - **PageRank Calculation:** Calling adjacencyList.PageRank(p) takes O(p⋅E).
    - **Overall:** The total complexity is the sum of these two phases.

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

**What I Learned:**

1. **Core PageRank Mechanics:** Gained a clear understanding of how PageRank iteratively distributes rank based on incoming links.
2. **Adjacency List Practicality:** Solidified the practical implementation of adjacency lists for graph representation in C++.
3. **Algorithm Efficiency:** Enhanced my ability to analyze the computational complexity of algorithms using Big O notation.
4. **Data Structure Choice Impact:** Reinforced that selecting the appropriate data structure is crucial for performance.

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

1. **std::unordered\_map for URL Mapping:** I would use std::unordered\_map instead of std::map for url\_to\_id. This would improve the average-case complexity of URL lookups and insertions from O(logV) to O(1), potentially speeding up graph construction.
2. **Modular Design for PageRank Variants:** For extensibility, I'd design the PageRank method with more modularity to easily incorporate features like a damping factor or explicit dangling node handling.
3. **Enhanced Input Validation:** In a production environment, I would add more robust checks for input data validity to improve program resilience.