Arion Stern - Simplified PageRank Documentation

1. Data Structure Description

Data Structures Used:

- unordered_map<string, vector<string>> outgoingEdges: Stores all outgoing links from a given webpage. This represents the directed edges in the graph.
- unordered_map<string, vector<string>> incomingEdges: Stores all incoming links to a given webpage. This structure is essential for calculating PageRank, since PageRank is derived only from incoming links.
- set<string> allPages: Keeps track of all unique webpages (nodes). It also facilitates alphabetical sorting of the final output.

Why These Were Chosen:

- **Hash maps (unordered_map)** provide O(1) average case access time, which is useful for dynamic and fast edge insertions.
- **Vectors** make edge traversal efficient and simple.
- **The set** ensures that we can iterate through the pages in alphabetical order when generating the final result, as required by the specification.

This design allows fast construction of the graph and efficient iteration during rank computation, while clearly separating incoming vs. outgoing relationships.

2. Method Time Complexities

addEdge(const string& from, const string& to)

- **Worst-case Time Complexity:** O(log V)
- **Variables Considered:**
  - V: Number of unique pages (vertices)
  - n: Number of edges added so far

Justification: This method performs the following updates:

1. Appends “to” to the “from” node's outgoing edge list and vice versa for incomingEdges. Each unordered_map insertion and vector push_back() takes **O(1)** on average.
2. Inserts both from and to into a std::set (allPages), which takes **O(log V)** time per insertion.

Since these set insertions dominate the operation, the overall worst-case complexity is **O(log V)** per call.

PageRank(int power_iterations)

- **Worst-case Time Complexity:** O(p * (V + E) + V log V)
- **Variables Considered:**
  - V: Number of unique pages (vertices)
  - E: Number of edges (links)
  - p: Number of power iterations

Detailed Breakdown:

1. **Initialization**: Setting rank of each node to 1/V takes O(V)
2. **Power Iterations**: For each of the (p - 1) iterations:
   - Iterate over all nodes: O(V)
   - For each node, loop through its incoming links (can be up to O(E) total)
   - Total per iteration: O(V + E)
   - Total for all iterations: O(p * (V + E))
3. **Final Sorting**: Alphabetically sorting the result using set or a sorted vector takes O(V log V)

Thus, the total time complexity is **O(p * (V + E) + V log V)**.

3. main() Time Complexity

main() Function:

- **Worst-case Time Complexity:** O(n log V + p * (V + E) + V log V)
- **Variables Considered:**
  - n: Number of edges provided as input
  - V: Number of unique pages
  - E: Number of edges
  - p: Number of power iterations

Justification:

1. **Input Parsing**: Reading each of the n lines and parsing with stringstream is O(1) per line
2. **Graph Construction**: addEdge() is called once per line, and is O(log V) due to std::set insertions → O(n log V) total

3. **Calling PageRank()**: This method is called at the end of main() and has time complexity **O(p * (V + E) + V log V)** as described above

Therefore, the total worst-case time complexity of main() is **O(n log V + p * (V + E) + V log V)**.

4. Reflection

What I Learned:

This project deepened my understanding of the PageRank algorithm and how graph-based ranking systems function. Implementing it from scratch taught me how link structure and node connectivity influence rank distribution across a directed graph. I also strengthened my C++ skills, particularly in using unordered_map, vector, and set effectively to represent and traverse graph structures. Additionally, I came to appreciate the value of thorough unit testing, especially in identifying edge cases involving self-loops, cycles, and sink nodes.

What I Would Do Differently:

- Start documenting my time complexity as I developed each function, rather than at the end.
- Add a toggle or flag to suppress cout output during testing — useful when PageRank() returns a string and prints to cout.
- Plan and write test cases alongside development, instead of after. Doing so would have helped me catch issues earlier and build my implementation with a clearer structure from the start.
- Structure the PageRank logic into smaller helper functions and add clarifying comments earlier in the process. This would have made the code easier to debug, maintain, and test incrementally.