Final Report

Summary of Final Datasets:

File Name	File Description
small.txt	Text depiction of a small directed graph with 6 nodes and 13 edges.
small-loop.txt	Text depiction of a small directed graph with 7 nodes and 15 edges, including a loop.
med-berkstan.txt	Text depiction of a medium-sized directed graph with 100 nodes and 464 edges. Modified from web-BerkStan.txt which contains 685230 nodes.
med-wiki.txt	Text depiction of a medium-sized directed graph with 95 nodes and 610 edges. Modified from wiki-Vote.txt which contains 7115 nodes.
large-berkstan.txt	Text depiction of a large-sized directed graph with 1000 nodes and 4848 edges. Modified from a web-BerkStan.txt which contained 685230 nodes.
large-wiki.txt	Text depiction of a large-sized directed graph with 911 nodes and 9760 edges. Modified from wiki-Vote.txt which contains 7115 nodes.

Proof of Correctness of Algorithm:

Since we had such large datasets, it would be quite tedious to calculate and rank all of them by hand to show that the algorithm works. In our code, we chose a convergence threshold of one ten-millionth. This would mean that we would have to go through millions of hand done calculations if we want to get the precise answers we got when running our algorithm. This is because you'd have to go through So instead, we parsed the datasets we had into a network analysis Python library called NetworkX, that had a built-in PageRank method. Here we will show one iteration of the algorithm:

<u>Nodes:</u>
1 -> 2, 1 -> 3, 1 -> 4
2 -> 1, 2 -> 3
3 -> 1, 3 -> 4, 3 -> 6
4 -> 3
5 -> 4, 5 -> 2
6 -> 3, 6 -> 4

<u>Adjacency Matrix:</u>
[0 1 1 1 0 0] Node 1 links to 2, 3, 4
[1 0 1 0 0 0] Node 2 links to 1, 3
[1 0 0 1 0 1] Node 3 links to 1, 4, 6
[0 0 1 0 0 0] Node 4 links to 3
[0 1 0 1 0 0] Node 5 links to 2, 4

Final Report

[0 0 1 1 0 0] Node 6 links to 3, 4

<u>Transition Matrix:</u>
[0 ⅓ ⅓ ⅓ 0 0] From Node 1
[½ 0 ½ 0 0 0]   From Node 2
[⅓ 0 0 ⅓ 0 ⅓] From Node 3
[0 0 1 0 0 0]   From Node 4
[0 ½ 0 ½ 0 0]   From Node 5
[0 0 ½ ½ 0 0]   From Node 6

All nodes will start with a page rank of ⅙ or 0.1666667

Node 1:
(1 - 0.85)/6 + 0.85 * (1/2*0.1667 + 0 + 0)
≈ 0.025 + 0.85 * (0.08335 + 0 + 0)
≈ 0.025 + 0.07085
≈ 0.09585

Node 2:
(1 - 0.85)/6 + 0.85 * (1/3*0.1667 + 0 + 0 + 0 + 1/2*0.1667)
≈ 0.025 + 0.85 * (0.05556 + 0 + 0 + 0 + 0.08335)
≈ 0.025 + 0.11826
≈ **0.14326**

Node 3:
(1 - 0.85)/6 + 0.85 * (1/3*0.1667 + 1/2*0.1667 + 0 + 0 + 0 + 1/2*0.1667)
≈ 0.025 + 0.85 * (0.05556 + 0.08335 + 0 + 0 + 0 + 0.08335)
≈ 0.025 + 0.18861
≈ **0.21361**

Node 4:
(1 - 0.85)/6 + 0.85 * (1/3*0.1667 + 0 + 1/3*0.1667 + 0 + 1/2*0.1667)
≈ 0.025 + 0.85 * (0.05556 + 0 + 0.05556 + 0 + 0.08335)
≈ 0.025 + 0.16731
≈ **0.19231**

Node 5:
(1 - 0.85)/6 + 0.85 * (0 + 0 + 0 + 0 + 0)
≈ 0.025 + 0
≈ **0.025**

Final Report

Node 6:
(1 - 0.85)/6 + 0.85 * (0 + 0 + 1/3*0.1667 + 1*0.1667)
≈ 0.025 + 0.85 * (0 + 0 + 0.05556 + 0.1667)
≈ 0.025 + 0.18861
≈ **0.21361**

As you can see, the PageRank algorithm demonstrates its effectiveness even after just one iteration, primarily through the initial distribution of PageRank values that reflect the underlying link structure of the graph. Nodes with more inbound links or connections from significant nodes begin to exhibit higher PageRank scores, aligning with the core principle of PageRank: a page's importance is determined by the importance of the pages linking to it. This initial differentiation of values, influenced by the damping factor (typically around 0.85), showcases the algorithm's fundamental functionality. The damping factor, representing the probability of randomly jumping to another page, ensures that no page has a zero probability of being visited, crucial for dealing with nodes without outbound links.

Moreover, the first iteration sets a trend toward the eventual convergence of PageRank values, indicative of the algorithm's robustness and mathematical soundness. The changes observed in PageRank values, although more pronounced in early iterations, begin to stabilize over time, demonstrating the algorithm's ability to balance and accurately reflect the importance of nodes within a network. This early behavior, grounded in principles of linear algebra and the treatment of dangling nodes (nodes with no outbound links), provides confidence in the algorithm's effectiveness from the outset. Overall, even a single iteration is a strong indicator that the PageRank algorithm functions correctly, laying the foundation for more precise calculations in subsequent iterations.

Time Complexity Calculation:

Analyzing the computational complexity of the implemented PageRank algorithm reveals an interesting divergence from its theoretical complexity, usually cited as O(E * K). This discrepancy is primarily rooted in the specific implementation details, particularly the approach to processing the graph's structure.

Starting with the construction of the adjacency matrix, the implementation requires iterating through each edge of the graph. This operation alone contributes a complexity of O(E), where E is the number of edges. The next step involves creating the transition matrix. While in a dense graph, this could potentially scale to O(N^2) complexity (N being the number of nodes), a more nuanced estimate for typical graphs would be O(N + E). This is because the process essentially entails traversing all edges once, thus aligning more closely with the graph's actual connectivity.

However, the crux of the complexity issue lies in the calculation of the PageRank values. This section of the code, which runs for a predetermined number of iterations (`maxIterations`), involves two key operations: computing the dangling rank (O(N)) and updating the PageRank for each node (O(N^2)), owing to the nested loop structure. As a result, the PageRank calculation part operates with a complexity of O(maxIterations * N^2), thereby dominating the overall complexity of the algorithm, which can be expressed as O(E + maxIterations * N^2).

The contrast with the theoretical complexity of O(E * K) is primarily due to the algorithm's execution manner in each iteration. The theoretical model assumes a streamlined process where each edge is efficiently processed once per iteration, directly linked to the graph's structure. In contrast, our implementation necessitates a nested loop per iteration, accounting for all nodes and their potential interconnections. This difference highlights the influence of implementation specifics, such as the choice of data structures and the method of processing node connections. It serves as a reminder that the practical complexity of algorithms like PageRank can significantly vary based on how they are actualized, especially in cases of dense graphs where nodes have numerous connections.