PageRank Algorithm

Academic Reference:

The algorithm that we are proposing to implement is the PageRank algorithm designed by Google founders, Larry Page, and Sergey Brin (https://www.cis.upenn.edu/~mkearns/teaching/NetworkedLife/pagerank.pdf).

Other sources we used to understand the paper are Wikipedia's simplified algorithm (https://en.wikipedia.org/wiki/PageRank), as well as a handout from Stanford computer science course CN54N, *Great Ideas in Computer Science* *(https://web.stanford.edu/class/cs54n/handouts/24-GooglePageRankAlgorithm.pdf)* .

Algorithm Summary:

PageRank is an algorithm developed by Google to determine the importance of web pages based on links. Think of the web as a vast network: pages are points, and links are connections between them. Each link to a page is like a vote, suggesting that page is valuable. However, not all votes are equal; links from authoritative sites, like CNN or Mayo Clinic, count more. The more significant and numerous these "votes" a page receives from high-ranking sites, the higher its own PageRank will be. This recursive system ensures pages genuinely useful to users rank higher.

Function I/O

This implementation consists of two parts:

1. Graph Building Function:
Graph buildGraph(std::string infile)
- `@param infile` -- the absolute path to the file containing the web link structure.
- `@return` A `Graph` object that represents the link structure of the web as an adjacency matrix.

This function will parse the given input file to construct a graph where each node represents a webpage and each directed edge represents a hyperlink from one page to another. It will read the link structure from `infile` and create an adjacency matrix that is used to represent the graph.

Proposed tests for this function would involve checking that the graph structure is correctly read from a file with a known structure. For example, we could have a small-scale test with a file representing a mini-web of a few pages and manually verify that the adjacency matrix is built correctly.

PageRank Algorithm

2. PageRank Calculation Function:

PageRankVector calculatePageRank(const Graph &graph, double damping_factor, unsigned int max_iterations)
- `@param graph` -- a `Graph` object representing the web link structure.
- `@param damping_factor` -- the probability at any step that the person will continue clicking links.
- `@param max_iterations` -- the maximum number of iterations to run the algorithm for convergence.
- `@return` A `PageRankVector` where each element represents the PageRank score of the corresponding webpage.

This function takes the web graph as input and computes the PageRank for each node using the damping factor and a fixed number of iterations to approximate convergence. It implements the iterative calculation that refines the PageRank scores until they reach a steady state or until the maximum number of iterations is reached.

For testing, we would check the output PageRank scores against a graph with a known PageRank distribution. We could use a small graph where we can manually calculate the PageRank or compare it to a trusted implementation.

Note:
Our actual implementation would need additional components like a function to parse the input file into a suitable `Graph` structure, handling edge cases in the graph structure (e.g., dangling nodes), and ensuring convergence of the PageRank values to within a specified tolerance, rather than simply iterating a fixed number of times.

Data Description:

We got our data from the Stanford Large Network Dataset Collection under the Stanford Network Analysis Platform (SNAP) site. This collection contains a multitude of graphs, several of which as .txt files, that can be used for network analysis. The collection also indicates whether the data is provided in the form of a directed graph, so we specifically chose directed graphs to be used as inputs for our PageRank algorithm. Since the data we retrieved is already in the form of a directed graph as a .txt file, we would not need to process it. We also included a small dataset of 6 nodes to be used for initial testing of our algorithm.

The Stanford Large Network Dataset Collection can be found here: https://snap.stanford.edu/data/index.html.