PageRank Implementation Documentation

1. Introduction

This document provides a detailed overview of the PageRank algorithm implementation in Java, covering the Graph structure, computation method, and integration with the database.

2. Graph Representation

The Graph class encapsulates the link structure and related metadata:

- **outgoingLinkCount** (Map<Integer, Integer>): Stores the number of outgoing links (out-degree) for each document.
- **incomingLinks** (Map<Integer, List<Integer>>): Maps each document to the list of documents that link to it.
- **danglingNodes** (Set<Integer>): Contains documents with no outgoing links (out-degree = 0).

3. Building the Graph

The buildGraph method initializes the data structures and processes a list of links:

1. Key steps:

- Initialize incomingLinks and outgoingLinkCount for each document ID.
- Iterate through all Link objects:
- - Increment the out-degree for the source document.
- - Add the source document to the incoming links list of the target document.
- Identify dangling nodes by collecting documents with zero out-degree.

4. PageRank Computation

The computePageRank method performs the iterative calculation of PageRank scores:

2. Algorithm details:

- Initialize all pages with a uniform rank of 1/N, where N is the total number of documents.
- For each iteration up to MAX_ITERATIONS:
- 1. Compute the total rank of dangling nodes and distribute it uniformly (dangling contribution).
- 2. For each document, sum the contributions from incoming links (each source page divided by its out-degree).
- 3. Apply the damping factor formula:

- `newRank = (1 - d)/N + d * (incomingSum + danglingContribution)`
- 4. Check for convergence: if the maximum change across all pages is below `CONVERGENCE_THRESHOLD`, stop early.

Key parameters: `DAMPING_FACTOR_PAGE_RANK` (typically 0.85), `CONVERGENCE_THRESHOLD` for early stopping, and `MAX_ITERATIONS` as a safety bound.

5. Handling Dangling Nodes

Dangling nodes (pages with no outgoing links) can distort rank distribution. In this implementation, their rank is summed and redistributed equally among all pages in each iteration.

6. Convergence and Performance

Convergence is detected when the maximum rank change drops below the threshold, reducing unnecessary iterations and improving performance.

Time complexity per iteration: O(E + N), where E is the number of links and N is the number of pages. Total complexity depends on the number of iterations until convergence.

7. Database Integration

After computing the PageRank scores, the `rankPagesByPopularity` method retrieves all document IDs and links from the database, runs the computation, and then updates the ranks in batch via `databaseHelper.batchUpdatePageRank(pageRankScores)`.

8. Conclusion

This implementation offers a clear and modular approach to computing PageRank, handling graph construction, dangling nodes, damping factor application, and convergence detection, with seamless integration into a database-backed system.