# Report on BDA Assignment 3 | Simrank
# By Garv Makkar | 2021530



## Task

Using the provided data, create a graph in Neo4j where the nodes represent papers and edges represent citation relationships between them. For every row in the dataset, the value corresponding to field "paper" represents the node id of the citing paper, and the list corresponding to "reference" represents the node ids of the cited papers. So, you should draw directed edges accordingly. For rows, where reference list is empty, you cannot draw a citation link but you need to keep the citing node. The task is to run SimRank algorithm on the constructed citation graph using Apache Spark to list the most similar nodes w.r.t. to a given query node. Use three different values of C={0.7,0.8,0.9} for three different runs and report the results. Take the query node ids as {2982615777, 1556418098}.

## Solution

### Step 1: Viewing Data

This step involved viewing the dataset in JSON format to get an idea of how it looks and how to use it.
Length of dataset: 564340 entries

### Step 2: Adding Data to Neo4j database to store the graph.

Data was loaded into a list, and these queries were run by looping over it after connecting to the database.

**Query 1: Create Constraints**
CREATE CONSTRAINT IF NOT EXISTS FOR (p:Paper) REQUIRE p.id IS UNIQUE

**Query 2: Create or Match a Citing Paper Node**
MERGE (p:Paper {id: $paper_id})

**Query 3: Create Citation Relationships**
MERGE (ref:Paper {id: $ref_id})
MERGE (p:Paper {id: $paper_id})
MERGE (p)-[:CITES]->(ref)

## Step 3: Connecting DB to Spark

Connected neo4j DB to Spark using the connector jar file

## Step 4: Sampling Dataset

To avoid a huge runtime, 10 percent of the nodes were sampled, and the edges between them were added.

## Step 5: Performing Simrank

SimRank is approximated for memory efficiency by partitioning the graph into smaller subsets of nodes, reducing the memory load during computation. The process involves:

1. **Graph Partitioning:** Nodes are divided into manageable partitions based on their connectivity, ensuring each partition fits within a memory constraint.
2. **Sparse Adjacency Representation:** Adjacency lists are created to represent node relationships compactly, minimizing memory overhead.
3. **Iterative Similarity Computation:** Within each partition, pairwise similarities are calculated iteratively using only the local adjacency information and previously computed similarities, avoiding global computations.
4. **Incremental Updates:** Similarities are updated across partitions iteratively, ensuring convergence while keeping the computation focused on subsets of the graph.

This approach balances computational demands and memory usage by processing smaller graph portions incrementally, ensuring scalability for large datasets.

## Results

**For query node 2982615777:**
There were no results for this node because it had no neighborhood, so no similar nodes to it.

**For query node 1556418098:**
These results change on every run because different data is sampled each time. We can avoid sampling, but that would take too much time.

- **Output after running code for the first time:**

  **Results for C = 0.7**
  Top similar papers for query ID 1556418098:
  Paper ID: 1556418098, Similarity: 1.0000
  Paper ID: 2152222281, Similarity: 0.7000
  Paper ID: 2158479011, Similarity: 0.7000
  Paper ID: 2169776910, Similarity: 0.3500
  Paper ID: 2097575504, Similarity: 0.0272

  **Results for C = 0.8**
  Top similar papers for query ID 1556418098:
  Paper ID: 1556418098, Similarity: 1.0000
  Paper ID: 2152222281, Similarity: 0.8000
  Paper ID: 2158479011, Similarity: 0.8000
  Paper ID: 2169776910, Similarity: 0.4000
  Paper ID: 2097575504, Similarity: 0.0356

  **Results for C = 0.9**
  Top similar papers for query ID 1556418098:
  Paper ID: 1556418098, Similarity: 1.0000
  Paper ID: 2152222281, Similarity: 0.9000
  Paper ID: 2158479011, Similarity: 0.9000
  Paper ID: 2169776910, Similarity: 0.4500
  Paper ID: 2097575504, Similarity: 0.0450

- **Output after running code for the second time:**

  **Results for C = 0.7**
  Top similar papers for query ID 1556418098:
  Paper ID: 1556418098, Similarity: 1.0000
  Paper ID: 1496163732, Similarity: 0.1750
  Paper ID: 1513489099, Similarity: 0.0000
  Paper ID: 1508068333, Similarity: 0.0000
  Paper ID: 1497057245, Similarity: 0.0000

  **Results for C = 0.8**
  Top similar papers for query ID 1556418098:
  Paper ID: 1556418098, Similarity: 1.0000
  Paper ID: 1496163732, Similarity: 0.2000

Paper ID: 1513489099, Similarity: 0.0000
Paper ID: 1508068333, Similarity: 0.0000
Paper ID: 1497057245, Similarity: 0.0000

**Results for C = 0.9**
Top similar papers for query ID 1556418098:
Paper ID: 1556418098, Similarity: 1.0000
Paper ID: 1496163732, Similarity: 0.2250
Paper ID: 1513489099, Similarity: 0.0000
Paper ID: 1508068333, Similarity: 0.0000
Paper ID: 1497057245, Similarity: 0.0000

## Code and other files

**For Step 1:** 1_View_Data.ipynb
**For Step 2:** 2_Add_Data_To_Neo4j_DB.ipynb
**For Steps 3-5:** Final_V1.py
**Results:** simrank_results_V1.txt
**Log file:** V1_log.log
**Dataset:** Data/train.json (Not included in submission)