BDA Assignment-3

Big Data Analytics – Graph Analysis with Neo4j and Apache Giraph

**Dataset

Wikipedia Vote Network**

- Download: https://snap.stanford.edu/data/wiki-Vote.html
- Format: Edge list (`from to`), each line represents a directed edge.
- Size: **7,115 nodes**, **103,689 edges**.

Ground Truth Metrics (for verification):

Metric	Ground Truth
Nodes	7,115
Edges	103,689
Nodes in largest WCC	7,066 (0.993)
Edges in largest WCC	103,663 (1.000)
Nodes in largest SCC	1,300 (0.183)
Edges in largest SCC	39,456 (0.381)
Average clustering coefficient	0.1409
Number of triangles	608,389
Fraction of closed triangles	0.04564
Diameter	7
90% effective diameter	3.8

Assignment Goals

1. Implement all graph metrics in **Neo4j** (Cypher + GDS).
2. Implement the same metrics in **Apache Giraph** (Java vertex-centric programming).
3. Compare **execution time** of both.
4. Analyze the trade-offs.

Part 1 – Installation & Free Online Tools

Neo4j Options

1. **Free Online Option**: Neo4j AuraDB Free
    - Sign up with a free account.
    - Create an empty project and database.
    - Load CSV via the "Import" tab or Cypher `LOAD CSV`.
2. **Local Option**: Neo4j Desktop
    - Runs on Windows/Mac/Linux.
    - Includes GDS library in recent versions.

Apache Giraph Options

1. **Free Online Option**:
    - Use a **Google Colab + Hadoop/Giraph Docker image** (lightweight cluster) or
      GitHub Codespaces with Hadoop pre-installed.
      Example Docker image: `giraph-docker`
2. **Local Option**:
    - Install Hadoop locally.
    - Download Apache Giraph.
    - Build with Maven:
    - `mvn clean install -DskipTests`

Part 2 – Neo4j Implementation

Data Loading (CSV)

Convert `wiki-Vote.txt` into CSV files:
`nodes.csv` (id) and `edges.csv` (src, dst).

```
// Create constraints
CREATE CONSTRAINT FOR (u:User) REQUIRE u.id IS UNIQUE;

// Load nodes
LOAD CSV WITH HEADERS FROM 'file:///nodes.csv' AS row
CREATE (:User {id: toInteger(row.id)});

// Load edges
LOAD CSV WITH HEADERS FROM 'file:///edges.csv' AS row
MATCH (u1:User {id: toInteger(row.src)}), (u2:User {id:
toInteger(row.dst)})
CREATE (u1)-[:VOTED_FOR]->(u2);
```

Starter Query Skeleton

```
// Count nodes & edges
MATCH (n:User) RETURN count(n) AS nodes;
MATCH ()-[r:VOTED_FOR]->() RETURN count(r) AS edges;

// Largest Weakly Connected Component
CALL gds.wcc.stream({
  nodeProjection: 'User',
  relationshipProjection: 'VOTED_FOR'
})
```

```
YIELD nodeId, componentId;

// Largest Strongly Connected Component
CALL gds.alpha.scc.stream({
  nodeProjection: 'User',
  relationshipProjection: 'VOTED_FOR'
})
YIELD nodeId, componentId;

// Average Clustering Coefficient
CALL gds.localClusteringCoefficient.stream('myGraph')
YIELD nodeId, coefficient;

// Triangle Count
CALL gds.triangleCount.stream('myGraph')
YIELD nodeId, triangleCount;

// Diameter (Eccentricity)
CALL gds.alpha.eccentricity.stream('myGraph')
YIELD nodeId, eccentricity;
```

Note: Replace `'myGraph'` with a named in-memory graph created via `CALL gds.graph.create(...)`.

Part 3 – Apache Giraph Implementation

Input Format

Giraph needs an adjacency list format:

```
<vertex_id> <neighbor1>:<edge_value> <neighbor2>:<edge_value> ...
```

For unweighted graphs, use `0` as edge value.

Starter Java Skeleton

```
import org.apache.giraph.graph.BasicComputation;
import org.apache.giraph.edge.Edge;
import org.apache.giraph.graph.Vertex;
import org.apache.hadoop.io.DoubleWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.NullWritable;

import java.io.IOException;

// Example: Weakly Connected Components via Label Propagation
public class WCCComputation extends BasicComputation<
        LongWritable, LongWritable, NullWritable, LongWritable> {

    @Override
    public void compute(Vertex<LongWritable, LongWritable, NullWritable>
vertex,
                        Iterable<LongWritable> messages) throws IOException
{
```

```
        long currentComponent = vertex.getValue().get();
        boolean changed = false;

        // Find minimum label among self and messages
        for (LongWritable msg : messages) {
            if (msg.get() < currentComponent) {
                currentComponent = msg.get();
                changed = true;
            }
        }

        if (changed) {
            vertex.setValue(new LongWritable(currentComponent));
            sendMessageToAllEdges(vertex, new
LongWritable(currentComponent));
        }

        vertex.voteToHalt();
    }
}
```

Other tasks:

- **SCC** – Implement two-phase algorithm (forward/backward BFS).
- **Triangle Count** – Each vertex sends neighbor list to neighbors; count intersections.
- **Clustering Coefficient** – From triangle counts and degree.
- **Diameter/Effective Diameter** – BFS from sample vertices.

Running Giraph

```
hadoop jar giraph-examples-<version>-for-hadoop.jar \
org.apache.giraph.GiraphRunner \
your.package.WCCComputation \
-vif org.apache.giraph.io.formats.LongLongNullTextInputFormat \
-vof org.apache.giraph.io.formats.IdWithValueTextOutputFormat \
-w 1
```

Part 4 – Performance Measurement

1. Record **execution time** for each metric in Neo4j (use `PROFILE`/`EXPLAIN` or GDS logs).
2. Record **execution time** for each Giraph job (check job summary logs).
3. Create a **comparison table**:

Metric	Neo4j Time (sec)	Giraph Time (sec)	Faster Tool	% Difference
WCC	...	...	...	...
SCC	...	...	...	...
...	...	...	...	...

Part 5 – Report

- **Implementation details** (both platforms).
- **Environment setup** (Neo4j Aura, Giraph on Docker/Colab).
- **Performance observations** – Neo4j faster for interactive queries; Giraph faster for large-scale batch computations.
- **Limitations** – Graph size, cluster resources, API limitations.
- **Conclusion** – Recommendations for tool choice by scenario.

If you want, I can also prepare a **Colab notebook** that sets up Giraph in a container for students so they can run it without worrying about Hadoop installation. That would make it much easier for them to test and time the jobs.

Do you want me to create that Colab-based Giraph environment setup next?