Project 2: Publications Knowledge Graph (KG) Construction, querying, and analytics

Data set

We will use the same data set from project 1. The source of the data is https://www.aminer.org/citation, version 13 as it is the most detailed one in JSON format. You can also check the schema of the respective data set on the same page under the "Description" link.

Operations

You can reuse the same pre-processing from the ETL project. Or, you can develop more pre-processing tailored towards:

- The construction of a knowledge graph for the publications data set.
- Running queries against the graph using Spark Graph Frames
- Graph analytics on the constructed knowledge graph

Use Graph frames to build a knowledge graph that represents different entities and their relationships

- Examples of entities
  - Authors
  - Institutions
  - Scientific domains
  - Publishing venues
  - publication
- Examples of relationships
  - Authorship (Author, Publication)
  - Co-authorship (Author, Author)
  - Works for (Author, Institution)
  - Specializations(Author, scientific domain)
  - Cites(Publication, Publication)
- Graph querying: You can use built-in querying features (Motif finding) in Graph frames (Cypher-like) to issue queries like
  - Co-authorship network of a given author, (up to n hops)
  - Publications that are co-authored by authors from different domains
  - Visualization of query results. You can use any visualization library of your choice to visualize the results of the queries, in a similar fashion of how Neo4J visualizes graphs.
  - **Advanced feature**: simple motif patterns are assumed to be queried against a single graph frame that represents a single relationship. Additionally, the language provided for querying looks similar to Cypher of Neo4J. However, as Neo4J is native for storing graphs, one can specify the node type in the query as well as the relationship. For example, [u:Person]-(:Owns)-> [c:Car], looks for person u and the corresponding owned cars c. In Graph frames, you would assume that there is a graph frames that holds the "owns" relationship against which you issue a motif query [u]-()->[c]. In this advanced step, you need to add a translation layer that receives a query like **[u:Person]-(:Owns)-> [c:Car]** and using meta data that tracks where each relationship

is stored, e.g., in Parquet files, see example to save and load GraphFrames here, with this metadata, your code can generate the corresponding query in PySpark that makes the query against the respective GraphFrame.

- Graph analytics:
  - Influential papers: you can use Page rank for this using the citation relation
  - Communities: you can find them by finding strongly connected components in relationships like co-authorship or being member of the same scientific domain.

Design Options and Limitations

As Spark does not support graph-native storage, we will end up with frames for vertices and frames for relationships. This is closely similar to storing the graph data in relational tables. There are two options: single table and separate relationship tables. In the former, you have one table on the form **(src, dst, relation, att1, att2, …, attn)**. You store all types of relationships in one single graph frame. Attributes att1 to attn. will be derived from the actual attributes you want to store for each relationship. For example, for authorship relationship, you can store the rank of the author in this publication, i.e., the first, second, etc. For the works-for relationship, you need to store, for example, from and to dates of this relationship. You can notice that overall, the attributes will be sparse. Moreover, you must store all the vertices in the same data frame. For example, you must put the authors and publications in the same data frame. Yet, you need to distinguish them for querying and filtering later.

The other option is to keep a separate graph frame for each relationship. This will simplify the storage and the preparation, but vertices will be duplicated if you want to keep a separate nodes-frame for each relationship. You can also keep one single nodes frame but separate relationship frames.