HW-1: Big Data Analysis FALL 2024-2025

# Spark and Information Retrieval

The main goal of this assignment is to build a small-scale spark-based search engine that enables users to search for a list of relevant documents as answers to a user’s query. To implement this index, you need to accomplish the following:

## Inverted Index Construction:

Assume having a list of N documents: doc1, doc2, doc3, …, docN, each of which contains a sequence of tokes (words).

You have to read the content of the files into a single RDD object, and then perform the required mappings and/or other operations in order to end up creating a new file (wholeInvertedIndex.txt) having the following format:

**Word, Count(Word), Document\_lst\_wth\_positions:**

**Word**: one single word from this collection of files

**Count(Word)**: the number of documents containing “Word”

**Document\_lst\_wth\_positions**: a list of document objects. Each document Object

(doc, [pos1, pos2, pos3, …])

includes the key (doc) that refers to the document containing the word “Word” and a list of positions sorted in ascending order, where each position refers to the locations of “Word” within the document. Assume that each document is tokenized based on spaces and the position of the first token is 0.

For example, some random rows from the generated inverted index file can be

meet, 2, [(doc3, [30, 43]), (doc14,[100])]

play, 4, [(doc3, [4, 14]), (doc6, [15, 10]), (doc120, [12, 45, 60]), (doc133, [34, 100, 150])]

soccer, [(doc6,[16, 32]), (doc12, [22, 77])]

.

.

Note the following:

* The number of rows in this file equals to the number of unique words in the collection.
* this step is done once.
* The words are sorted alphabetically
* The document list associated with each token is sorted in an ascending order and the including positions as well.

## MongoDB:

Here you need to store your generated index into MongoDB collection, where each document in the collection corresponds to one term.

## Query Processing:

* In this step, we need to read wholeInvertedIndex.txt into a new RDD. Then, the application should search this RDD to answer a user’s query. For example, if user searches for “play”, then the system should print: doc3, doc6, doc120, doc133. If user searches for “play soccer”, then the system prints: doc6.
* Your search should support “phrase queries”, so that when a user enters a query with multiple tokens, then the system should able to retrieve that words contain the phrase as is.
* Implement your query processing engines using two modes:
  + MongoDB APIs
  + Spark RDD APIs
* Use a large set of queries to compare between the two query processing modes in terms of the required running time.