Report on Implementing Text Preprocessing, Unigram Inverted Index, Positional Index, and Boolean & Phrase Query Processing

By Janesh Kapoor, 2021466

Text Preprocessing

The first step involved preprocessing text files in a dataset. The preprocessing steps included:

Lowercasing: Convert all text to lowercase to ensure uniformity and prevent case sensitivity issues during search queries.
Tokenization: Split the text into individual words or tokens to facilitate the removal of stopwords and punctuation.
Stopwords Removal: Filter out common words (stopwords) that are usually irrelevant to the meaning of the texts for search purposes.
Punctuation Removal: Eliminate punctuation marks, as they are often not necessary for text analysis and can interfere with search queries.
Blank Space Tokens Removal: Ensure that tokens consisting solely of whitespace are removed since they do not contribute any meaningful information.

Unigram Inverted Index

A unigram inverted index was created to efficiently support boolean search queries. This index maps each unique word to a list of documents that contain the word. Implementation details included:

- **Index Creation**: For each preprocessed document, every unique word was added to the index along with the document identifier if the word was not already present.
- **Boolean Query Support**: Functions were implemented to perform 'AND', 'OR', 'AND NOT', and 'OR NOT' operations on document sets returned by querying the index with search terms.
- **Serialization**: The index was serialized using Python's pickle module for persistence, allowing it to be saved to disk and loaded back into memory as needed.

Positional Index

The positional index extends the unigram inverted index by also recording the positions of each word occurrence within documents. This enables the support of phrase queries, where the order of words matters. Key aspects included:

- **Index Structure**: Each entry in the index consists of a word mapped to a dictionary, where keys are document identifiers and values are lists of positions where the word occurs within the document.
- **Phrase Query Processing**: To process a phrase query, the index is queried for each word in the phrase, and document lists are intersected in a way that considers word positions to ensure the words appear in the correct order and proximity.

Query Processing

The implementation supports both boolean and phrase queries:

- Boolean Queries: Queries combining terms with boolean operators ('AND', 'OR', 'AND NOT', 'OR NOT') were supported. A generalized approach allowed for the execution of complex queries involving multiple boolean operations.
- Phrase Queries: Queries for specific phrases were supported by utilizing the positional index to find documents where the queried words occurred in the specified order and proximity.

Input/Output Handling

- For boolean queries, the input format included the number of queries followed by the queries themselves, each split into terms and operations. The output format detailed the query number, the number of documents retrieved, and their names.
- For phrase queries, a similar input format was used, with each query representing a phrase to search for. The output format again included the number of documents retrieved and their names, but tailored to phrase query processing.

Conclusion

This project demonstrated the essential components of a text search system, including text preprocessing, index creation, and query processing. Through the implementation of unigram inverted and positional indexes, it supported efficient boolean and phrase queries over a preprocessed dataset. The use of Python's pickle module for index serialization facilitated the persistence of these data structures, enhancing the system's usability.