Algorithms, Data Structures, and Optimizations

Tokenizer.py

1. Token Creation Function:

The tokenization function uses (`re.findall`) to split a document into individual words, then matches it to the regular expression `r'\b\w+\b'` to create boundaries and extract continuous sequences of alphanumeric characters as tokens.

1. Stopword Removal Function:

Stopword removal filters out common English stopwords from the list of tokens. The list of English stopwords used is obtained from NLTK's `stopwords` corpus, which contains commonly occurring words such as "the", "is", "and", etc.

1. Stemming Function:

The stemming function uses Porter stemming to reduce words to their root or base form. The `PorterStemmer` class from NLTK is used for the stemming process.

1. Preprocessing Optimization:

The preprocessing function combines tokenization, stopword removal, and stemming into a single step.

InvertedIndex.py

1. Inverted Index Data Structure:

The `InvertedIndex` class represents an inverted index data structure, which consists of three main components:

* + 1. `self.index`: A dictionary storing the inverted index, where each token is mapped to a tuple containing its document frequency (df) and a list of postings (document ID, term frequency).
    2. `self.IDF\_index`: A dictionary storing the inverse document frequency (IDF) values for each token.
    3. `self.tf\_idf`: A dictionary storing the TF-IDF weight vectors for each document.

1. Algorithms:
   1. `build\_index`: Constructs the inverted index, calculates document frequencies (df), and populates the index dictionary.
   2. `build\_idf`: Computes the IDF values for each token based on the document frequency (df) obtained from the inverted index.
   3. `build\_tf\_idf`: Constructs the TF-IDF weight vectors for each document using the TF and IDF values obtained from the inverted index.
   4. `get\_postings`: Retrieves the list of postings (document ID, term frequency) for a given token from the inverted index.
   5. `get\_df`: Retrieves the document frequency (df) for a given token from the inverted index.
   6. `get\_tf`: Retrieves the term frequency (TF) for a given document ID and token from the inverted index.
   7. `get\_idf`: Retrieves the IDF value for a given token from the IDF index.
   8. `get\_vector`: Retrieves the TF-IDF weight vector for a given document from the `tf\_idf` dictionary.
   9. `get\_vector\_length`: Computes the Euclidean norm (length) of the TF-IDF weight vector for a given document.
2. Optimizations:
   1. Uses dictionaries (`index`, `IDF\_index`, `tf\_idf`) to store information efficiently, allowing for fast lookup and retrieval of data.
   2. The `parse\_file` function efficiently parses XML files into individual documents using BeautifulSoup, avoiding unnecessary overhead and ensuring accurate extraction of document contents.
   3. The `get\_vector\_length` function optimizes the computation of the Euclidean norm by avoiding unnecessary iterations and utilizing the `math.pow` function.

RetrieveRank.py

1. Algorithms:
   1. Document Retrieval: The retreive method retrieves document IDs that match at least one query token by accessing the inverted index.
   2. Vector Comparison: The cosSim method calculates the cosine similarity between two vectors.
   3. Ranking: The rank method ranks document IDs based on their cosine similarity to the query vector.
   4. Query Processing: The query method processes a query by calculating its query vector, retrieving relevant documents, and ranking them.
2. Data Structures:
   1. Inverted Index: The inverted index, implemented in the InvertedIndex module, stores document IDs and their corresponding TF-IDF weights.
   2. Vectors: Document vectors are represented as dictionaries mapping tokens to their TF-IDF weights.
   3. Lists: Document IDs and their corresponding cosine similarity scores are stored in lists.
3. Optimizations:
   1. Document Retrieval: The retreive method efficiently retrieves relevant document IDs by accessing postings lists from the inverted index.
   2. I/O Optimization: Results are written to a file (Results.txt) in append mode to minimize I/O overhead.