Documentation for the simple search engine implementation by Daniil Gurgurov

This program is a search engine that searches for relevant documents in a given collection based on a query. The search engine uses the vector space retrieval model for specifying how to rank the documents. The collection of documents is stored in an XML file, which is preprocessed and indexed using TF-IDF scores. The core of the search engine is an index that weights terms according to the tf.idf weighting scheme.
The program consists of a single class, SearchEngine, that comprises the following methods:

1. *__init__(self, collectionName, create):*
   This method (constructor) creates the SearchEngine object with the name of the document collection and a boolean parameter "create" that determines whether to create the search index or read from files. If create=True, the method calls the compute_idf() and compute_tf() methods to calculate the IDF and TF scores for each token in the document collection and write them to files. If create=False, the method reads the IDF and TF scores from files.
2. *preprocess(self, collectionName):*
   This method reads in an XML file containing the document collection, parses the file using BeautifulSoup, extracts the text from the HEADLINE and TEXT tags, removes any punctuation characters from the tokens, stems the tokens, and returns a list of dictionaries where each dictionary contains the document ID, the headline, and a list of stemmed tokens from the text and headline. The method is used as a shortcut for pre-processing the data in the compute_idf and compute_tf methods.
3. *compute_idf(self, collectionName):*

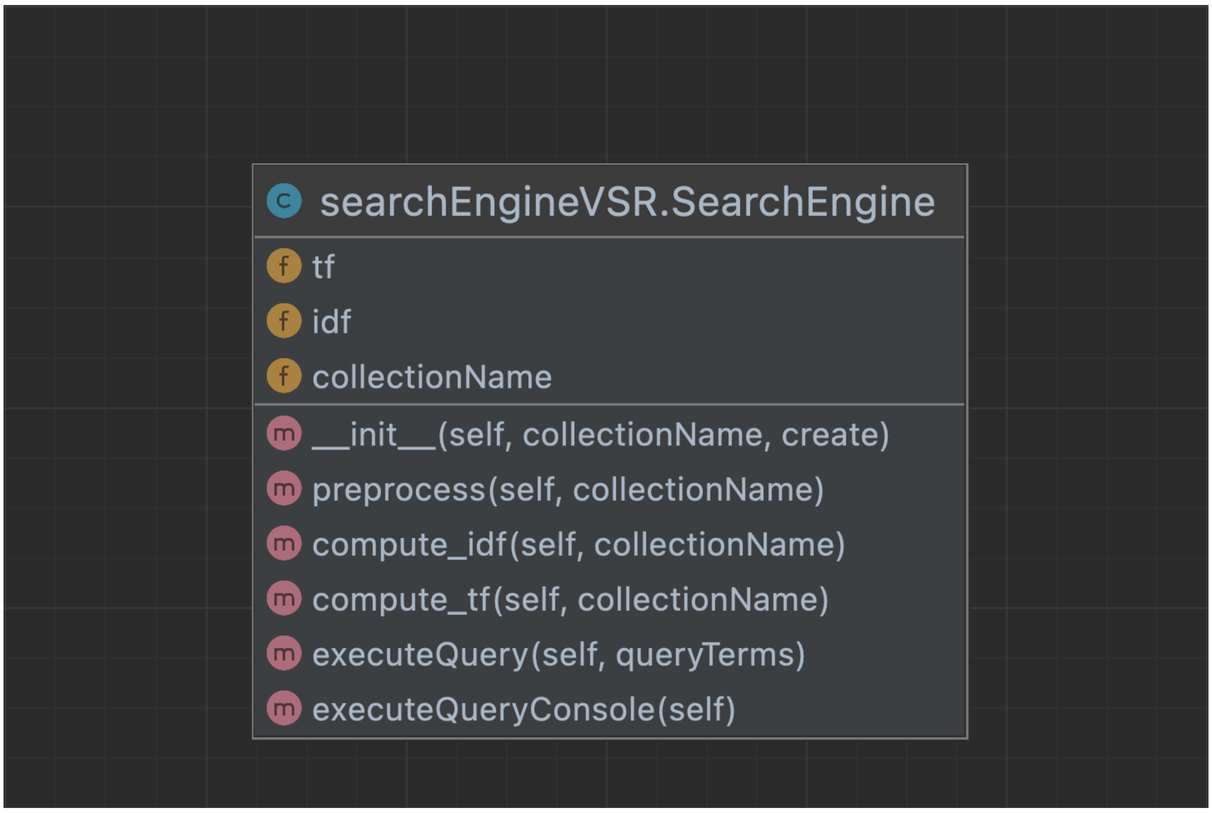
This method extracts and pre-processes the document collection with the help of the preprocess method, counts the number of documents containing each term and the number of all documents, and calculates the IDF (Inverse Document Frequency) values for each term. The IDF values are then saved in a file named collectionName.idf. Inverse document frequencies show how informative a term is in general.

4. *compute_tf(self, collectionName):*
   This method extracts and pre-processes the document collection, counts the number of occurrences of each token in each document, finds the maximum count of any token in each document, and calculates the TF (Term Frequency) value for each token in each document. Term frequencies are adjusted with regard to document length. The TF values are then written in a file named collectionName.tf. Term frequencies represent how important a term is within a document.
5. *executeQuery(self, queryTerms):*
   This method takes a query string as input, preprocesses the query by removing punctuation and stemming the tokens, calculates the TF-IDF score for each document containing at least one query term, and returns a list of dictionaries where each dictionary contains the document ID and the TF-IDF score. The documents are ranked by their TF-IDF scores in descending order. If no documents contain any query terms, an empty list is returned.
6. *executeQueryConsole(self):*
   This method is an implementation of a console-based search engine. It takes user input in the form of a query, which is then pre-processed to remove any punctuation, converted to lowercase, and stemmed. The processed query is then passed to the executeQuery() method to retrieve the top 10 matching documents. If no results are found, a message is displayed indicating that no documents were found. Otherwise, the top 10

results (or less) are displayed, along with their corresponding document ID and relevance score. The loop continues to prompt the user for queries until the user enters an empty query.

To run the program, create a SearchEngine object by calling the constructor with the name of the collection and a boolean value indicating whether to create a new index or load an existing one. Then, use the executeQueryConsole method on the object. When the program is working, provide the query by typing in the words and pressing “ Enter”. The program will be running until an empty query is prompted.

Class diagram:


searchEngineVSR.SearchEngine
tf
idf
collectionName
__init__(self, collectionName, create)
preprocess(self, collectionName)
compute_idf(self, collectionName)
compute_tf(self, collectionName)
executeQuery(self, queryTerms)
executeQueryConsole(self)