Data Structures:

- Binary Trees
- Lists
- Dictionary
- Tuples

Architecture

Phrase query

This algorithm checks whether a given query occurs in a document by analyzing the positions of the query terms. It utilizes a dictionary to efficiently retrieve the positions of the terms and determines if they appear in the correct sequence.

Boolean retrieval

This algorithm breaks down a query into parts separated by logical operators, such as AND, OR, and NOT. It then finds the intersection, union, or complement of the resulting lists of documents according to the logical operator used in the query.

Wildcard

This algorithm utilizes two binary trees for forward and backward propagation (the word is reversed) to efficiently search for documents that match a query containing wildcard characters. The algorithm traverses the binary trees to find the resulting list of matching documents.

Spellcheck

The algorithm defines a spell checker that can correct the spelling of words in a text based on their probability of occurrence in a training corpus. It uses a Counter object to keep track of the frequency of each word in the training corpus. The correction function takes an input word and returns the most probable spelling correction for that word by generating a set of candidate corrections using the candidates function, and selecting the candidate with the highest probability using the P function. The candidates function generates possible spelling corrections for a word by applying a set of edit operations such as deleting, transposing, replacing, or inserting characters, and returning the subset of those corrections that appear in the training corpus. The algorithm has an extended version, correction_v2, which generates candidate corrections with up to three edits instead of just one, using the known_edits2 and known_edits3 functions to return the subset of edits that appear in the training corpus.

Acronym Expansion

The acronym_expander function takes an acronym as input and returns a list of possible expansions of the acronym. It does this by constructing a URL based on the input acronym and sending an HTTP request to that URL using the requests library. The HTML response is then parsed using BeautifulSoup to extract the full forms of the acronym. The resulting list of full forms is returned after being cleaned up by removing the text before the hyphen. We return the intersection of queries(one with expansion and one without expansion)

Summarizer

This code defines a function named summarize that takes in a text string and a summary_length integer (default value of 5), and returns a summary of the text as a string. The summary is created by computing the frequency of each word in the text, and then scoring each sentence in the text based on the sum of the frequencies of the words in that sentence. The function returns the top N (N = summary_length) sentences with the highest scores as the summary. The text is tokenized into sentences and words, and stop words and punctuation are removed before computing word frequencies and sentence scores.

The nltk library is used to tokenize the text and remove stop words, and the heapq library is used to efficiently select the top N sentences with the highest scores.