COMP3506

Jingwei WANG

4382484

This program is a general textual search application. There are three input files that included document file, index file and stop word file. The last two files are optional. The index file saved sections' index of the input document. The stop word file contains some words from document need to be ignored when searching some words.

Data structure

There are three types of data type which are implement to the Section, Line, and Word. The Word is a single linked list, which contains the column number of the first character at current line, the string content, string size, Boolean to indicate a stop word and the next Word. The Line is a single linked list which contains line number, first Word at this line and the next line. The Section is a double linked list which contains first line, title of current section, next section and previous section.

This program will load the input file and save single word into Word. Create a new Line when finished current line. Create a new Section when the current section is finished. This program will load the first section, the first line in this section and the first word in this line if there is no a section name is indicated. So that the worst case is O(n) where n is the number of words in this document. The reason of using double linked list is that program needs to get the first line number and the first line number of the next section to determine how many lines this section has. It is better if using trie because it is faster to match current word and stop words. Using trie to store all words and record the information for every word. The program does not need to iterate every word in the document. But it needs more time to pre-process data so the linked list is preferred.

Algorithm

The constructor of Autotester initials input files such as document, index and stopWord. Firstly, it detects the path of document file whether is valid. And it detects the path of index file. If the path of index file is valid and the content of index file is not empty, it will read the content of index file and save each index with title and line number. It creates Section for each index depends on the information of index. It initializes Line to store the information of each line in the current Section. It counts lines between two sections to determine how many lines should be initialized. The last of section should only have one Line. If no index file provided, it will initialize one section and one line in this section. After read the index file, it detects the path of stop words file. If the path of stop words file is invalid or the content of the stop words file is empty, it will do nothing. If the path of stop words is valid and the content of this file is not empty, it will read the content line by line. It creates StopWord for each single word. After read the stop word file, it starts to read document file. It loads the first section and gets the first line of this section and read the first line of the provided file. It saves each word at the current line and determine whether is a stop word or not. It loads next section if current section is finished. The last section has only one line after the initialization. So that it will creates Line before save the words at current line of the file.

There are three private methods which are isWordFound(), getWordCount() and getResult(). The isWordFound() is used to determine whether the required words is on the indicated line. It will start from the first word from the array of words and read the words on the indicated Line one by one. If the word is found, it returns true, otherwise, false. The getWordCount() is used to get the word count of required words on the indicated line. It starts from the first word from the array of required words and trying to match words of the indicated line. If the current word from indicated line is a stop word then skips it to the next word. If the word is matched then add it to the result. The getResult() is used to find the line number and column number of the first word of the input phrase. Firstly, it starts from the second words of the provided phrase. If the phrase is one word then add it to result and exit. If the phrase has more than one word, it will get the word from the indicated Line and compare to the second word. If the second word is match then compare the third one until the last word of this phrase found. If all words from the phrase matched then add the line number and column number of the first word from the phrase to the result and exit. It will switch to the next line and next section until the last line of the last section is loaded.

The phraseOccurrence() starts from the first word of the phrase. If the first word is matched, it will compare the second until reached the last word. If all words is matched returns the result.

The prefixOccurrence() starts with the first character of the prefix. If the first character is matched it will compare the second character until the last character is reached. It will add the information of this word to the result if the prefix is matched.

The wordsOnLine() starts from the first word of required words. If the first word is found on a line, it will try to find the second word until the last word is reached. If all words found are on the same line, it adds the information to the result.

The someWordsOnLine() starts from the first word of required words. Search the whole document to find the word until the last word of the last line of the last section is reached. And then it will try to find the second word until the last word of the required words is reached and the last word of the last line of the last section is reached.

The wordsNotOnLine() starts from the first word of required words which is the same as the above one. It will try to find the excluded words in the line which is found the required words. If no excluded words are found, it adds information to the result.

The simpleAndSearch() starts with the first word of the required words and the first section of required sections until the last word and the last section are reached. If all required words found in a provided section, it adds to the result.

The simpleOrSearch() is the same as above but it adds information to result if any required word found in section.

The simpleNotSearch() is the same as above but no excluded words in section.

The simpleAndOrSearch() is the same as above but required at least one orWord found in the section which is found required words.