Plagiarism Checker

The report describes the implementation of the plagiarism checker implemented in C.

1. **Procedure:** For each sentence in the corpus document, a check is made whether that sentence exists or not in the test document. The **unique sentences** are stored in a **dynamically allocated char**.** To make the checking of sentences efficient, for each document **Tries** are used for storing the sentences as well. Thus, each sentence stored in the char** is queried for its presence in the Trie for the test file.

2. **Similarity Metric:** The check function returns the number of characters that are matched which is divided by the length of the sentence ultimately yielding the fraction matched. This is done for all the unique sentences for the corpus documents (stored in char**). Based on the experiments with different aggregation techniques of the fractions and check results, averaging all the fractions resulted in promising results for the corpus and the test text file provided.

3. **The complexity of the approach:**
    a. **Preprocessing Step -** All the characters of the input are read sequentially and whenever a full stop is received, the current aggregation of the characters is inserted into the Trie if it isn't present. This process takes **O(L)** where L is the length (number of characters) of the current aggregated sentence. The sentence if not present in the Trie is also stored in the dynamically allocated char**. The amount of space occupied would be **O(128 * L)** as at each level of the Trie, an array of size 128 corresponding to the ASCII indices of the character is declared, and **O(L)** for the storage in the char** array. So for the entire preprocessing step, the **overall worst-case time complexity** would be **O(C)** where **C** is the total number of characters in the corpus

document. The **overall worst-case space complexity** for the procedure would be **O(128 * C),** where **C** is the total number of characters in the corpus document.

b. **Checking -** An iteration is performed over all the unique sentences of the corpus document and they are queried for their presence in the Trie of the test document. The operation of searching for a sentence would take **O(L)** where L is the length (number of characters) of the sentence. This is repeated for each sentence in the char**. Therefore the **overall worst -case time complexity** for this process would be **O(C)** where **C** denotes the number of characters in the document.

   Therefore the **overall worst case time complexity** for **analysing a corpus document and a test document** would be **O(max(C1,C2))** where **C1** is the total number of characters in the corpus document and **C2** is the total number of characters in the text document.

   The **overall worst case space complexity** for **analysing a corpus document and a test document** would be **O(128 * max(C1,C2))** where **C1** would be the total number of characters in the corpus document and **C2** is the total number of characters in the text document.

4. **Results and Remarks:**
   a. The results are promising as the documents from which the content is plagiarised have significant differences in the similarity percentage as compared to the other documents.
   b. The method implemented would work for copied portions of the text from one document to other and, it doesn't take into account the sentiments or etc.
   c. The space complexity is high, but can be reduced by using sophisticated implementations of the Trie data structures such as compressed Tries etc, but the implementation complexity is high.

5. **Contents of the Top Directory:**
    a. **Src:** The file main.c is the source code for the implementation. Along with it there is a makefile.
    b. **Obj:** The object file main.o along with the makefile
    c. **Lib:** There are two folders:
        i. **Corpus_files:** contains all the .txt files of the corpus.
        ii. **Test_file:** contains the file to be tested.
    d. **Bin:** The executable file plagChecker along with the makefile.
    e. **Makefile** which calls the makefiles of the other subdirectories.

In order to compile the script,use the command: $ make
In order to execute the script, use the command: $ make execute

Param Khakhar
2018CS10362