SEARCH ENGINE
DSA-Project

Description

You need to write search engines with and without hashing. For that matter you should use the following two data structures**(In C++)**:

1. **Unordered_Map(Hash_map)**
2. **Tries (Bonus with equal weightage to part 1)**

After writing the search engines you need to compare the performance of two search engines and tell which one is better. You can use the time taken to process a query for this comparison. And may report an average of 'k' same queries on each engine. System time before and after the search call may be helpful.

The queries from the search engine could be words or sentences and your search engines should return the list of documents with the searched words (or sentences). The documents should be shown in an order s.t. a document with more occurrences of the searched word (or sentenced) appears first and the rest of the documents follow. For example, see the following search queries:

a. **Single Word: WORD**

we should get all the documents containing this word, sorted by highest # of time this word occurrences)

b. **Two Words: WORD1 WORD2**

We should get all the documents which should contain either of the two words, sorted by those which has both the words and then the priority is which contains the highest occurrences of either of the word

c. **Multiple words** (space separated) **W1 W2 W3.....Wn**

We should get all the documents which should contain either of the n words, sorted by those which has most the words and then the priority is which contains the highest occurrences of either of the word (example input: this is search engine)

d. **Two Words: WORD1 + WORD2**

you should keep in mind the occurrence of both words in all documents. The documents with occurrence of only one of the words should not be part of the result. The order of the documents should again be based on the occurrence of the word.

e. Multiple words (+ separated) **W1+ W2+W3+.....Wn**
you should keep in mind the occurrence of all words in all documents. The documents with no occurrence of all the words should not be part of the result. The order of the documents should again be based on the occurrence of the word.

f. Sentence "**W1 W2 W3"** (if the first word is " then its a sentence search, and in that case the last word must be " as well , example input: **"**this is search engine**"**)

g. Subtraction W1 - W2 (BONUS)
It should return all the documents which should have the word **W1** but not the word **W2.**

h. Subtraction with sentence as well as multiple words "W1 W2 W3" - "W3 W4 W5" (BONUS)
It should return all the documents which should have sentence 1 but not sentence 2.

Inputs:

A. **Constantinople**
B. **Constantinople Turkey**
C. **Constantinople Turkey Pakistan ITU**
D. **Constantinople+Turkey**
E. **Constantinople+Turkey+Pakistan+ITU**
F. **"Constantinople Turkey Pakistan ITU"**
G. **Constantinople-Turkey**
H. **"Constantinople Turkey Pakistan ITU" - "Lahore Pakistan ITU"**

Example:
For query ***a,*** list the names of all documents that contain the word Constantinople. Suppose the word was found in *Document 1* (3 times), *Document 7* (9 times), and *Document 23* (2 times). Your result should display the documents and the occurrence in the following order
Result**:** Document 7 (9 times), Document 1(3 times), Document 23 (2 times).

Instructions:

1. As a first step you need to download this file https://drive.google.com/file/d/1hA1TtFBpPGjH7ZHxaq28nbZ_yYZMwu8U/view?usp=sharing run this code : (you can use any data but text files should be more than 20000)
Or download this folder : https://drive.google.com/drive/folders/13Fy3CHtdOtoodQB_w9MsQNHwtYWI5gOz?usp=sharing
*it's only for text files extraction , your project should be in cpp

```
import csv
import os

csv_file_path = "Reviews.csv"
output_folder = "review_text"

def create_review_text_file(review):
    file_name = f"{output_folder}/review_{review['Id']}.txt"
    with open(file_name, 'w', encoding='utf-8') as file:
        file.write(f"ProductId: {review['ProductId']}\n")
        file.write(f"UserId: {review['UserId']}\n")
        file.write(f"ProfileName: {review['ProfileName']}\n")
        file.write(f"HelpfulnessNumerator: {review['HelpfulnessNumerator']}\n")
        file.write(f"HelpfulnessDenominator: {review['HelpfulnessDenominator']}\n")
        file.write(f"Score: {review['Score']}\n")
        file.write(f"Time: {review['Time']}\n")
        file.write(f"Summary: {review['Summary']}\n")
        file.write(f"Text: {review['Text']}\n")

def csv_to_text(csv_file_path):
    if not os.path.exists(output_folder):
        os.makedirs(output_folder)

    with open(csv_file_path, 'r', encoding='utf-8') as csv_file:
        reader = csv.DictReader(csv_file)
        for row in reader:
            create_review_text_file(row)

if __name__ == "__main__":
    csv_to_text(csv_file_path)
      print("Conversion complete. Text files created for each review in the 'review_text' folder.")
```

This code will make a folder “review_text” inside folder you can find text files in which you have to search

2. You should have a class in your program that keeps record of position and number of occurrences of a word in a particular document. The class may look like as follows:

```
class WordInDocument
{
```

```
        string documentName;
        vector <int> position;

public:
        // Write appropriate functions
}
```

This class should be instantiated when placing a word in Map/Hash_Map and position of the word can be saved in the vector. If a word has occurred more than once in a document the vector position will have more than one entry. For example if the word ***Constantinople*** is 9th and 50th word in *Document 23*, widJ is an instance of WordInDocument, then the vector widJ.position will have 2 entries which are 9 and 50

.

3. Your program should read (one by one) each file generated in step 1 and store each word of each file in Map as well as Hash_Map/Unordered_Map. Storing a word means that you should actually store an instance of WordInDocument class with its documentName and position variables populated. In case a word is already part of the Map or Hash_Map you may only need to update the position vector instead of adding two instances of the same class if the word has occurred in the document being read. In case this is the first occurrence of a word, then a new instance of WordInDocument may be added in Map/Unordered_Map.