Data structure project-2023

Rocky

Beihang University

This is a comprehensive performance testing assignment that involves large amounts of test data and assesses students' mastery of data structures and algorithms. Topics covered may include sequential lists, linked lists, binary search trees and search (indexing and hashing), sorting, etc. There are two identical questions with different test data sets. The grading criteria for the assignment are based on the sum of the scores for both questions,.

1. The first question is a small data set testing question. We have the dictionary file (*dictionary.txt*), stopword file (*stopword.txt*), text file (*article.txt*), and sample running result file (*results(example).txt*) for students to use for debugging the program. There is no performance testing for this question,
2. The second question is a large data set testing question. Points are awarded only if the program runs correctly (passes the test cases), with correct results accounting for 40% of the score and performance accounting for 60%. Performance is evaluated based on the average of the fastest two programs, with scores calculated accordingly. Programs that do not produce results or produce incorrect results will not receive points.

It is recommended that students try different methods to implement the program to understand how the combination of different data structures and algorithms can affect program performance. Additionally, students are encouraged to use the knowledge learned in this course to solve the problem.

**Keyword-Based Large-Scale Document Search (Comprehensive - Small Data)**

[Problem Description] Search engines such as Baidu and Google provide efficient webpage and document search functions, allowing users to query information of interest through one or more keywords. To implement a large-scale text document search, efficient indexing and query algorithms are typically required. Implement a document search program based on keywords to achieve quick searching and sorting of large-scale text documents. The specific method is as follows：

1. Extract words (in English) from a given collection of documents (web pages) containing ***N*** documents and count the frequency of each word ***k*** appearing in each document ***d*** (i.e., the number of occurrences , where the total number of words in the document is ), from which the term frequency can be calculated.

To improve the accuracy of the algorithm, only words that in the dictionary but not belong to stop words will be counted. Words consist only of letters, and uppercase letters should be converted to lowercase letters before calculating the term frequency. The course website provides a dictionary file "dictionary.txt" and an stop word list file "*stopwords.txt*" (only contains words, sorted alphabetically).

Note: In natural language processing, stop words refer to a list of words that do not provide additional semantic information during text analysis, such as "a", "an", "he", "you" etc.

1. Count how many documents has the word ***k*** ( , i.e., the number of documents in which the word appears), and calculate the inverse document frequency , (log base 10). The definitions are as follows:
2. Based on the input keywords , use TF-IDF to score the relevance of the documents in the collection. For any given document d, the relevance score calculation formula for the input keywords is as follows:

If a certain keyword does not appear in the document collection, its does not need to be calculated, and its relevance to all documents is 0.

1. Sort the search results in descending order based on relevance and return the top ***N*** results.

The work of crawling relevant web pages (documents) from the Internet to simplify the implementation of the search engine has been completed, and the crawled web page document data has been stored in a text file (*article.txt*), where the first line of each webpage is the webpage ID (such as XX-XXXX, can be input as a string), followed by the webpage content, and web page documents are separated by a page break character (\f). A sample *article.txt* file for testing is available for download on the course website.

[Input Format]

Enter the number of search results to be returned (NUM) and the search keywords from the command line.

The format is as follows:

*search NUM*

where "***search***" is the search engine program, and there is a space between ***NUM*** and the keywords. The web page documents are processed according to the requirements above based on the "*dictionary.txt*", "*stopwords.txt*" file, and the web page data file "*article.txt*" in the current directory.

Note:

1. Stop words and non-dictionary words in the input string will not be analyzed for relevance.
2. Due to the difference in line endings between Windows and Linux systems, the '\n' carriage return character in text files under Windows will be converted to two characters '\r' and '\n' in Linux (evaluation environment). It is recommended to use ***fscanf(fp, "%s", ...)*** to process English words in the dictionary and stop word files.

[Output Format]

First, output the information of the top 5 web pages ranked by Sim value to the screen. The output format should be as follows: the relevance score Sim value (with six decimal places), the corresponding web page number (numbered from 1 according to the order of the web pages in the *article.txt* file), and the ID of the web page in the *article.txt* file. The three values should be separated by a space, and there should be a line break at the end.

At the same time, output the information of the top ***N*** web pages ranked by Sim value to a file named "*results.txt*". The output format should be the same as the screen output, with each web page information followed by a line break. If the number of web page documents found (i.e., the number of documents with a Sim value greater than 0, i.e., the number of documents containing the given keywords) is less than ***NUM***, output the actual number of documents found instead (the screen output should be the same).

Note: If the Sim value is the same, sort the web pages in ascending order by web page number.

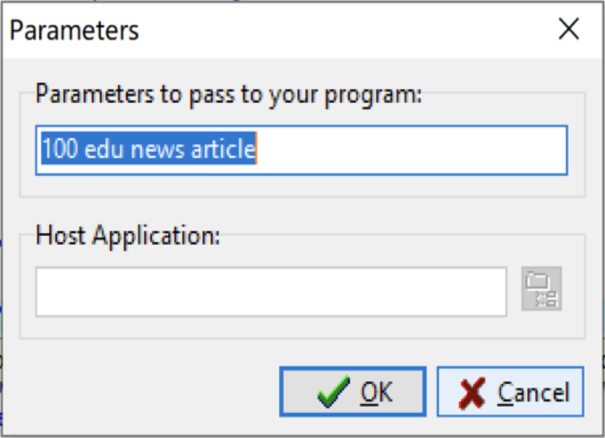
[Sample Input]

Assuming that "*search.exe*" is the search engine program, and the program run as follows:

*search 100 edu news article*

(Before running the program, download the files *article.txt, dictionary.txt, stopwords.txt, results (sample).txt* from the course website to the local directory.)

Explanation: If the local programming environment is dev-C++, click on the menu "*Execute\Parameters...*" and enter the corresponding command line parameters in the dialog box."



[Sample Output]

After running the program, the top 5 results on the screen are:

*0.581744 24 1-24*

*0.466224 230 1-230*

*0.447891 543 1-543*

*0.446951 54 1-54*

*0.440138 87 1-87*

The contents of the "*results.txt*" file generated should be exactly the same as the "*results (sample).txt*" file available for download.

[Sample Explanation]

The sample screen output shows the top 5 results ranked by relevance in descending order. The first part of each line is the relevance score (Sim) of the web page document, the second part is the corresponding document number in the file, and the third part is the ID of the document in the file. The "*results.txt*" file contains the top 100 results ranked by relevance in descending order.

[Scoring Criteria]

For testing, one that passing the test data will get full marks, and one that do not produce results or produce incorrect results will get no mark.

**Keyword-Based Large-Scale Document Search (Comprehensive - Big Data)**

The problem description is the same as **Keyword-Based Large-Scale Document Search (Comprehensive - Small Data)**

[Scoring Criteria]

This is a comprehensive performance test question, and the scoring criteria is based on the program with the fastest running time receiving full marks, while the scores of other programs are calculated based on the running time of the fastest program. Programs that produce no results, timeout (not exceeding 120 seconds), or incorrect results will not receive any points.