Data Structures

Assignment Title: Plagiarism Checker

Section: E

Fatima Ishtiaq, 23i-0696

Table of content

Objective

The primary objective of this assignment was to implement a **Plagiarism Checker System** that analyzes multiple documents to detect similarities between their contents. The system preprocesses text data by removing unwanted characters, filtering out stop words, and standardizing the text for comparison. It then identifies overlapping content to evaluate the extent of plagiarism.

Features and Functionalities

The plagiarism checker was developed in C++ and provides the following core functionalities:

1. **Document Reading and Management**
   - The system reads input from files containing paths to documents and a list of stop words.
   - It stores the contents of these files dynamically and organizes them for comparison.
2. **Text Preprocessing**
   - **Removing Punctuation Marks:** Eliminates punctuation marks to ensure a clean comparison.
   - **Case Conversion:** Converts all text to lowercase for uniformity.
   - **Stop Word Removal:** Filters out commonly used words (e.g., "and," "the") that do not contribute to meaningful analysis.
3. **Similarity Detection**
   - The system compares processed documents to detect overlapping content.
   - Similarities are quantified by calculating the percentage of matching words between documents.
4. **Dynamic Memory Management**
   - Dynamically allocates memory to handle document contents and scales with the size of input files.
5. **Error Handling**
   - Handles scenarios like empty files or missing files gracefully, ensuring the program does not crash.

Implementation Details

Class: `fileStorage`

The `fileStorage` class was used to manage and process document data efficiently.

- **Attributes:**
    - `DocumentContent` (dynamic array): Stores the content of each document.
    - `stopWordsList` (2D array): Holds the list of stop words for preprocessing.
    - `numOfDoc`: Tracks the number of documents being analyzed.
    - `UniqueWords` and `DocumentFrequency`: Stores unique words and their frequencies for comparison.
- **Methods:**
    - `setNumOfDoc`: Sets the total number of documents.
    - `setLetterInDocumentContent`: Dynamically updates document content during processing.
    - `printStopWordsList`: Debugging method to display stop words.

Core Functions

1. **readInput:** Reads document paths and stop words from input files.
2. **removePunctuationMarks:** Strips all punctuation from text data.
3. **convertUpperToLowerCase:** Converts text to lowercase.
4. **removeStopWords:** Removes words that add little value to plagiarism analysis.
5. **compareDocuments:** Calculates the percentage similarity between two documents based on overlapping words.

Algorithm for Plagiarism Detection

1. **Input Processing:**
    - Read document contents and stop words from files.
    - Preprocess each document (remove punctuation, convert to lowercase, filter stop words).
2. **Tokenization:**
    - Split document text into individual words for comparison.

3. **Comparison:**
   - Compare the processed text of each document pair.
   - Calculate the similarity score as:

     Similarity Score = (Number of Matching Words / Total Words in Both Documents) x 100

4. **Output Results:**
   - Display similarity percentages for each document pair.

Challenges Encountered

1. **Efficient Stop Word Removal:**
   Removing stop words from large datasets proved challenging due to the high computational cost.
   Solution: Used optimized loops and string comparison to enhance performance.
2. **Memory Management Issues:**
   Managing dynamically allocated memory for document storage occasionally caused segmentation faults.
   Solution: Carefully debugged pointers and ensured proper memory deallocation.
3. **Error Handling in File Operations:**
   Handling scenarios like empty files and missing files required robust error handling mechanisms.
   Solution: Used appropriate checks to validate file existence and contents.
4. **Scalability of Similarity Computation:**
   Comparing multiple documents involved processing large amounts of text data, leading to performance bottlenecks.
   Solution: Implemented efficient algorithms for word comparison.

Results

The implemented system successfully:

- Reads and preprocesses multiple document files.
- Identifies and quantifies text similarities across documents.
- Outputs plagiarism scores, providing insights into the level of overlap.

Future Enhancements

1. **Improved Tokenization:**
   Use advanced natural language processing techniques for more accurate word splitting.
2. **Semantic Analysis:**
   Extend the system to detect paraphrased content using semantic similarity measures.
3. **Database Integration:**
   Store processed documents and plagiarism results in a database for easier management and scalability.
4. **Web-Based Interface:**
   Create a user-friendly web interface to allow users to upload documents and view plagiarism reports.
5. **Advanced Algorithms:**
   Incorporate hash-based or n-gram-based algorithms for faster and more accurate similarity detection.

Conclusion

This assignment provided an opportunity to develop a robust plagiarism detection system while exploring key programming concepts such as dynamic memory management, file handling, and text processing. The system demonstrates the ability to detect similarities between documents effectively and lays the groundwork for more advanced plagiarism detection solutions.