CSCE 1101-01Spring 2023
Term Project (2)
Dr. Howaida Ismail
Simple Plagiarism Detection Utility using String Matching

Yassin Tantawy 900213424 section 1
Joudy El Gayar 900222142 section 1
Khaled Nana 900211952 section 5

Department of Computer Science and Engineering, AUC

Introduction:

The Plagiarism Checker Application is a software project aimed at detecting and measuring the level of similarity between text documents. It provides a tool for individuals and organizations to identify potential instances of plagiarism and assess the extent of copied content.

The project utilizes C++ programming language and Qt, to develop a user-friendly interface for interacting with the application.

Problem Definition:

Plagiarism refers to the act of using someone else's work, ideas, or content without proper attribution or permission, which is considered unethical and can lead to serious consequences in academic, professional, and creative fields.

The program provides a solution to this problem by analyzing the contents of multiple text documents and determining the level of similarity between them. It calculates the plagiarism percentage, indicating the extent to which the text in question matches existing reference documents or paragraphs. This helps users identify potential instances of plagiarism, allowing them to take appropriate actions such as providing proper citations, rewriting the content, or initiating disciplinary measures if necessary.

By automating the plagiarism detection process, the program saves users valuable time and effort that would otherwise be spent manually comparing and evaluating documents. It provides an objective and quantifiable measure of similarity, helping users make informed decisions about the originality of the content they are analyzing.

Overall, the Plagiarism Checker Application assists in maintaining academic integrity, upholding professional ethics, and promoting originality in written work across various domains.

Methodology:

1. Requirement Analysis: Identify the specific requirements and objectives of the application. This includes determining the desired features, user interface design, supported file formats, and any additional functionalities or constraints.
2. Design Phase:

a. User Interface Design: Create a visual design for the user interface using Qt Designer or any other suitable tool. Consider the placement of buttons, input fields, result displays, and other relevant elements for an intuitive and user-friendly experience.

b. System Design: Define the overall architecture of the application, including the class structure, data flow, and interaction between different components. Determine the necessary classes, methods, and data structures required to implement the desired features.

3. Implementation: Write the code based on the design specifications. Implement the necessary classes, functions, and algorithms to perform text comparison, calculate plagiarism percentages, and handle user interactions. Utilize Qt framework's libraries and features to build the graphical user interface, handle file operations, and display results.
4. Testing: Perform thorough testing of the application to ensure its functionality, accuracy, and reliability. Write test cases to cover various scenarios, such as comparing different text documents, handling empty files, or evaluating edge cases. Conduct both functional and usability testing to validate the application's behavior and user experience.

Specification of Algorithms to be used:

Hamming Distance Algorithm:

The Hamming distance algorithm measures the difference between two strings. It calculates the number of positions at which the characters in the two strings differ. In simpler terms, it counts how many characters need to be changed or swapped to transform one string into another.

The Hamming distance algorithm is used in the Plagiarism Checker Application to calculate the similarity or difference between the input text and the reference text. By measuring the Hamming distance, the application can determine how closely the texts match and provide a quantitative measure of their similarity.

Plagiarism Percentage Calculation Algorithm:

The plagiarism percentage calculation algorithm determines the percentage of plagiarized content in a given text. It compares the input text with a database of reference paragraphs or documents and calculates the similarity based on the Hamming distance.

To calculate the plagiarism percentage, the algorithm considers the length of the input text and the number of characters that differ from the reference text. It calculates the ratio of the number of characters that match to the total length of the input text..

The plagiarism percentage calculation algorithm provides a measure of how much of the input text is similar to the reference documents. A higher percentage indicates a higher level of similarity and potentially indicates the presence of plagiarism.

These algorithms work together in the Plagiarism Checker Application to detect and assess the level of similarity between text documents, allowing users to identify potential instances of plagiarism and make informed decisions about the originality of the content.

Document class:

This class is responsible for recording the path of the input and the database which then levels the ground for the rest of the algorithms to use as a starting point to calculate the percentage of plagiarism.

First of all, its constructor initializes a private member called filepath, which is of type string, which stores the file path input to it by the constructor. The other public function "read DocumentContents" opens the text file specified by file path and reads each line of the text file using the "getline" function, integrating them into a single string, which is then returned. If the file could not be opened, then the function will give an error to the user. The function "checkPlagiarismPercentage" takes the text to be checked for plagiarism as a parameter, checks for plagiarism by reading a text file, initializing a "bruteForce" object with its contents, and calling the "checkPlagiarismPercentage" method with the input text. The result is returned as a double.

Rabin Karp Algorithm:

The Rabin Karp Algorithm is an algorithm used for matching patterns using a hash function. It does not go through every character, unlike other algorithms, it filters the unmatching characters and then does the comparisons. Rabin Karp performs by taking a sequence of characters , checking for the required string from the pattern in the text and identifying the matches between the two . The hash function is used to generate hash values for the pattern and the substrings of the text. Matches between text and pattern are quickly determined when comparing the hash values, if a hash value match is found then a full comparison will be performed to make sure that both strings match. This algorithm is implemented in the Plagiarism Detector class along with another algorithm which is LCS.

Longest Common Subsequence (LCS) Algorithm :

The LCS Algorithm is an algorithm used to find the longest subsequence that is common between two givens that could be arrays or strings. Here, the algorithm focuses on the length of the common strings and fills a table with all lengths , then compares each and every element of one subsequence to the other and updates the table.What makes this algorithm singular of the other matching algorithms is that it detects the similarity between both the text and pattern files even if the elements of the subsequence are not in a consecutive positions as the original pattern. This algorithm is used in the Plagiarism Detector class to provide more specific results than what is offered by Rabin Karp.

Data Specifications:

In our approach, we prompt the user to choose whether they would like to input their own database/corpus or they would like to use our existing corpus to check if their input was plagiarized or not. If the user chose to input their own corpus along with their own text, they are asked to enter the directory of the file they are intending to check if they plagiarized from or not. The directory that the user enters is input as the database for the Document class. This is then used to calculate the percentage of plagiarism using the brute force method, Rabin-Karp method, and the LCS method. The same thing takes place when the user chooses to use the existing corpus, the only difference is that the directory is already known in the code, which will not need to be input by the user. The other input data that will be used in our work is the text the user is testing for plagiarism. Depending on the algorithm, the text input by the user will be used in different ways to calculate the percentage of plagiarism.

Experimental Results:

The results of processing our input data shows the LCS and brute force methods to be the most accurate. We reached this conclusion as a result of using the text input by the user as the input and comparing it to the database, and the result for both the LCS and brute force algorithms were the same except for the Rabin Karp algorithm, which only shows a plagiarism percentage of 100% if the text is the same as the database, but if there is any difference in the input then the percentage of plagiarism will turn out to be 0% for the Rabin Karp. We tried increasing the size of the input to see if it was going to make any difference, but it did not change and the percentage was still 0%. Therefore, we were oriented to the LCS algorithm and in terms of results and percentage, it showed the most accuracy. So, as a result, the LCS and Brute force algorithms have proven to be the best in detecting plagiarism in terms of the accuracy of similarity percentage.

Analysis and Critique:

Using these three different algorithms while conducting the project, made us face some complexities whether in the comparison between both pattern and text or the accuracy in terms of the plagiarism percentage. However, we came to a conclusion that The Brute force is the simplest approach but inefficient in some cases as for large texts or patterns because it will require checking each and every position. Then as an improvement we used Rabin Karp which is more efficient than Brute Force because of the usage of hash functions that make the comparison between the pattern and text easier. Nevertheless, the provided plagiarism percentage is never accurate when applying Rabin Karp and we suffered with it until we came to a conclusion that Rabin Karp is unable to detect partial similarity between the text and the pattern , instead it only detects the exact similarity between the two of them, that make the plagiarism percentage either 0 % or 100% and cannot get any middle percentages. That's why we implemented the LCS algorithm hoping for a more efficient and accurate percentage. Even if the LCS algorithm is not primarily used in string matching, we found it to be very efficient in that purpose. In terms of time Complexity efficiency, Rabin Karp is the best option with a time complexity of O(m+n) because of the hashing functions that permits it to be quicker. However, the time complexity of LCS and Brute Force algorithms is the same with a rate of O (m*n) with m being the length of the pattern and n is the text's.

Conclusions:

In conclusion, the invention of a plagiarism detection tool that employs string matching algorithms like Rabin-Karp, Brute force, and Longest Common Subsequence (LCS) has turned out to be an effective tool for detecting instances of plagiarism within textual content. By providing teachers, researchers, and content creators a dependable and effective tool, this project aims to address the rising worry of academic dishonesty and intellectual property theft. Overall, the three string matching algorithms for plagiarism detection were successfully constructed and tested as part of this study. Each approach has advantages and disadvantages, and the selection of an algorithm might be influenced by elements like dataset size, needed speed, and desired accuracy levels. The outcomes of these algorithms offer a strong starting point for more study and advancement in the area of plagiarism detection. Machine learning techniques could be used in the plagiarism detection tool in the future to increase accuracy and decrease false positives. More sophisticated algorithms that are known for their effectiveness in pattern matching, like the Boyer-Moore algorithm or KMP algorithm, could also be included in the tool.

References:

JavaTpoint. (n.d.). DAA - Rabin-Karp Algorithm. JavaTpoint.
https://www.javatpoint.com/daa-rabin-karp-algorithm#

Coding Ninjas. (n.d.). Rabin Karp. CodeStudio.
https://www.codingninjas.com/codestudio/library/rabin-karp

Programiz. (n.d.). Longest Common Subsequence. Programiz.
https://www.programiz.com/dsa/longest-common-subsequence#:~:text=It%20stores%20the%20result%20of,be%20used%20in%20future%20computations

Open AI. ChatGPT.2023

https://chat.openai.com