**Table of Contents**

|  |  |  |
| --- | --- | --- |
| Sl.No | Content | Pg.No |
| 1 | Abstract | 2 |
| 2 | Introduction | 3 |
| 3 | Problem Statement | 4 |
| 4 | Objectives | 5 |
| 5 | Design and Algorithm | 6-11 |
| 6 | Results | 12 |
| 7 | Conclusion | 13 |
| 8 | References | 14-15 |

1.**Abstract**

Plagiarism detection is a critical task in academic and professional settings to ensure the originality and integrity of written content. This project presents a plagiarism checker based on the Longest Common Subsequence (LCS) algorithm, a dynamic programming technique used to identify the longest sequence of words or characters that appear in the same order in two documents, without necessarily being contiguous. By comparing the LCS between a given document and a set of reference texts, the system can effectively measure the degree of similarity and flag potential instances of copied content. The approach is language-independent, simple to implement, and particularly effective in detecting reworded or slightly altered plagiarized text, which often bypasses exact string-matching methods. This method serves as a foundational step in building efficient, real-time plagiarism detection tools that maintain fairness and uphold academic standards.

Here are the features of a Plagiarism Checker using Longest Common Subsequence (LCS):

Features:

**1. LCS-Based Similarity Detection:**

Uses the Longest Common Subsequence algorithm to detect similarities between texts, identifying matching sequences even if they are not contiguous.

**2. Partial Plagiarism Identification:**

Capable of detecting paraphrased or slightly modified plagiarized content, not just exact copies.

**3. Text Preprocessing:**

Includes normalization techniques such as lowercasing, punctuation removal, and stop word filtering to improve matching accuracy.

**4. Similarity Score Calculation:**

Computes a plagiarism score as a percentage based on the length of the LCS relative to the input text.

**2.Introduction**

In today’s digital age, the rapid growth of online information and easy access to content has increased the risk of plagiarism, particularly in academic, literary, and professional domains. Plagiarism, the act of presenting someone else’s work or ideas as one’s own without proper acknowledgment, undermines the integrity of intellectual work and poses serious ethical and legal challenges. To address this issue, various plagiarism detection systems have been developed, each utilizing different algorithms to compare and analyze text similarity.

This project focuses on developing a plagiarism checker using the Longest Common Subsequence (LCS) algorithm, a dynamic programming technique widely used in computer science for sequence comparison. Unlike exact string-matching methods, the LCS algorithm identifies the longest sequence of matching elements (words or characters) that appear in the same order in two documents, even if they are not contiguous. This makes LCS particularly effective in detecting partially plagiarized content where minor modifications or rewording have been made to avoid detection.

By applying the LCS algorithm, the system calculates a similarity score that indicates the extent of overlap between the input document and a set of reference texts. This method is computationally efficient, language-independent, and capable of highlighting both exact matches and rephrased content, making it a valuable tool in maintaining academic honesty and content originality.

One of the foundational approaches to text similarity detection is based on the Longest Common Subsequence (LCS) algorithm. The LCS technique identifies the longest sequence of characters or words that appear in the same order in both the source and target texts, though not necessarily in a continuous block. This feature makes LCS highly suitable for detecting subtle forms of plagiarism where text has been paraphrased or rearranged to evade exact matching algorithms. Unlike naive string matching, which fails to detect structural changes, LCS captures a broader range of similarities, providing a more reliable measure of potential plagiarism.

3.**Problem statement**

Plagiarism has become a significant issue in the digital era due to the ease of copying and reproducing content from online sources. In academic and professional settings, the detection of plagiarized content is critical for maintaining originality and ethical standards. Existing plagiarism detection systems often rely on exact string matching or external databases, which may fail to detect reworded or partially modified content. There is a need for a plagiarism detection method that is efficient, accurate, and capable of identifying both exact and near-duplicate content, including paraphrased sections.

**Key Challenges:**

**1. Detecting Partial and Paraphrased Plagiarism:**

Most simple algorithms detect only exact copies and struggle with identifying reworded or rearranged content.

**2. Handling Large Text Comparisons Efficiently:**

Comparing large documents against multiple references requires optimized and scalable algorithms to ensure reasonable performance.

**3. Language and Format Variability:**

Differences in language, punctuation, and formatting can obscure plagiarism, making detection more difficult.

**4. Balancing Accuracy and Speed:**

An effective system must strike a balance between detection accuracy and computational efficiency, especially in real-time applications.

**Proposed Solution:**

This project proposes the use of the Longest Common Subsequence (LCS) algorithm for detecting plagiarism in textual content. The LCS method compares sequences of words or characters between documents to find the longest ordered matching subsequence, even if the sequence is not continuous. This approach is particularly effective at identifying similarities in paraphrased content. The system preprocesses text by normalizing case, removing stop words, and tokenizing it for analysis. A similarity score is computed based on the LCS length relative to the input document length. By implementing LCS-based comparison, the system can effectively detect both exact and near-duplicate content, offering a lightweight, language-independent, and accurate plagiarism checking solution.

**4.Objectives**

**1. Implement an LCS-Based Detection Algorithm**

Develop a core plagiarism detection module using the Longest Common Subsequence (LCS) algorithm, capable of identifying the longest matching subsequences between input and reference texts.

**2. Detect Partial and Paraphrased Plagiarism**

Design the system to recognize not just exact matches but also reworded or structurally altered text by analyzing non-contiguous but ordered matches.

**3. Preprocess Text for Accurate Comparison**

Apply preprocessing techniques such as lowercasing, punctuation removal, stop word filtering, and tokenization to ensure clean and consistent input for LCS comparison.

**4. Calculate Plagiarism Similarity Score**

Develop a scoring mechanism that computes the percentage of plagiarism based on the length of the LCS relative to the total length of the original input.

**5. Compare Input with Multiple Documents**

Enable the system to check one document against multiple sources or a repository of documents to identify potential plagiarism from any of them.

**6. Build a Simple and Intuitive User Interface**

Create a user-friendly interface (web or desktop-based) allowing users to upload documents, start comparisons, and view results easily.

**7. Highlight Matching Content in Reports**

Incorporate a feature that visually highlights matched sequences within the documents to help users clearly identify the plagiarized sections.

**8. Ensure Language and Format Flexibility**

Design the system to handle different languages and text formats, making it usable in diverse academic and professional contexts.

**9. Optimize for Speed and Scalability**

Implement performance optimizations in the LCS algorithm and data handling to ensure fast comparison times, even with large texts or multiple documents.

**5.Design Technique & Algorithm**

**1. Modular Design Approach**

The system is divided into independent modules such as text preprocessing, LCS computation, similarity scoring, user interface, and report generation. This modularity enhances maintainability, reusability, and scalability.

**2. Text Preprocessing Design**

Preprocessing includes steps like:

Normalization: Converting all characters to lowercase.

Tokenization: Splitting text into words or characters.

Stop Word Removal: Eliminating common words that do not add meaning (e.g., "the", "is").

Punctuation Removal: Stripping special characters that can affect matching accuracy.

These steps ensure uniform input for the LCS algorithm and reduce noise during comparison.

**3. LCS Algorithm Design (Dynamic Programming)**

The LCS is implemented using a dynamic programming approach:

A 2D matrix is used to store the length of common subsequences.

Time complexity is O(m × n), where m and n are the lengths of the two strings.

The matrix allows backtracking to extract the matching subsequence.

This technique ensures optimal computation of sequence similarity.

**4. Similarity Score Computation Design**

The similarity percentage is calculated using the formula:

Similarity (%) = (Length of LCS / Length of Input Text) × 100

This design provides a quantitative measure of plagiarism severity.

**5. Multi-Document Comparison Framework**

A loop or batch comparison system is designed to check one input against multiple reference texts. Each result is stored and sorted based on similarity score for user review.

**6. Interface Design (UI/UX Principles)**

The user interface is designed for simplicity and clarity:

Upload fields for documents.

Display of plagiarism results with color-coded matched sections.

Downloadable report feature.UI may be implemented using tools like Tkinter (Python), web frameworks (Flask/Django), or JavaFX.

**7. Highlighting Algorithm**

A separate component parses the LCS results and highlights matched portions in the original documents using HTML or markup, improving readability for end-users.

**8. Report Generation Design**

A reporting module compiles all comparison data—document names, LCS length, similarity percentage, and matched sections—into a downloadable format (e.g., PDF or TXT).

**9. Error Handling and Validation**

The system is designed to handle various inputs and errors:

Unsupported file formats.

Empty or null text.

Excessively large inputs. This ensures robustness and user-friendly error messages.

**Algorithm: Plagiarism Detection Using Longest Common Subsequence (LCS)**

**Input:**

**Doc1**: The input document (suspected text)

**Doc2**: The reference document (original or source text)

**Output:**

Similarity Score: Percentage of similarity between Doc1 and Doc2

**LCS:** The longest common subsequence (for highlighting or reporting)

**Steps:**

**1. Preprocess the Input Documents**

Convert both Doc1 and Doc2 to lowercase.

Remove punctuation and special characters.

Tokenize the text into words (or characters).

Optionally remove stop words to reduce noise.

**2. Initialize LCS Matrix**

Let m = length of Doc1 tokens

Let n = length of Doc2 tokens

Create a 2D array LCS[m+1][n+1] initialized to 0

**3. Compute LCS Length (Dynamic Programming)**

for i = 1 to m:

for j = 1 to n:

if Doc1[i-1] == Doc2[j-1]:

LCS[i][j] = LCS[i-1][j-1] + 1

else:

LCS[i][j] = max(LCS[i-1][j], LCS[i][j-1])

**4. Backtrack to Extract the LCS Sequence (Optional)**

From LCS[m][n], trace back to find the matching tokens.

Store these tokens for highlighting or reporting.

**5. Calculate Similarity Score**

similarity = (LCS[m][n] / m) × 100

This gives the percentage of the input document (Doc1) that matches the reference.

**6. Compare Against Multiple Documents (Optional)**

Repeat steps 1–5 for each reference document.

Store the highest similarity score or all scores, depending on requirements.

**7. Generate Output**

Display or return:

LCS match

Similarity percentage

Highlighted matches (if GUI is available)

Report generation (PDF, TXT, etc.)

**Time Complexity:**

O(m × n) for each document comparison, where m and n are the lengths of the documents in tokens.

**Implementation**

import java.io.IOException;

import java.nio.file.Files;

import java.nio.file.Paths;

public class project {

public static int lcsLength(String X, String Y) {

int m = X.length();

int n = Y.length();

int[][] dp = new int[m + 1][n + 1];

for (int i = 0; i < m; i++) {

for (int j = 0; j < n; j++) {

if (X.charAt(i) == Y.charAt(j)) {

dp[i + 1][j + 1] = dp[i][j] + 1;

} else {

dp[i + 1][j + 1] = Math.max(dp[i + 1][j], dp[i][j + 1]);

}

}

}

return dp[m][n];

}

public static double calculateSimilarity(String text1, String text2) {

text1 = text1.replaceAll("\\s+", "").toLowerCase();

text2 = text2.replaceAll("\\s+", "").toLowerCase();

int lcsLen = lcsLength(text1, text2);

int maxLen = Math.max(text1.length(), text2.length());

return ((double) lcsLen / maxLen) \* 100;

}

public static String readFileContent(String path) throws IOException {

return new String(Files.readAllBytes(Paths.get(path)));

}

public static void main(String[] args) {

String filePath1 = "C:\\Users\\ASUS\\Desktop\\ada project\\AI.txt";

String filePath2 = "C:\\Users\\ASUS\\Desktop\\ada project\\DS.txt";

try {

String doc1 = readFileContent(filePath1);

String doc2 = readFileContent(filePath2);

double similarity = calculateSimilarity(doc1, doc2);

System.out.printf("Similarity based on LCS: %.2f%%\n", similarity);

} catch (IOException e) {

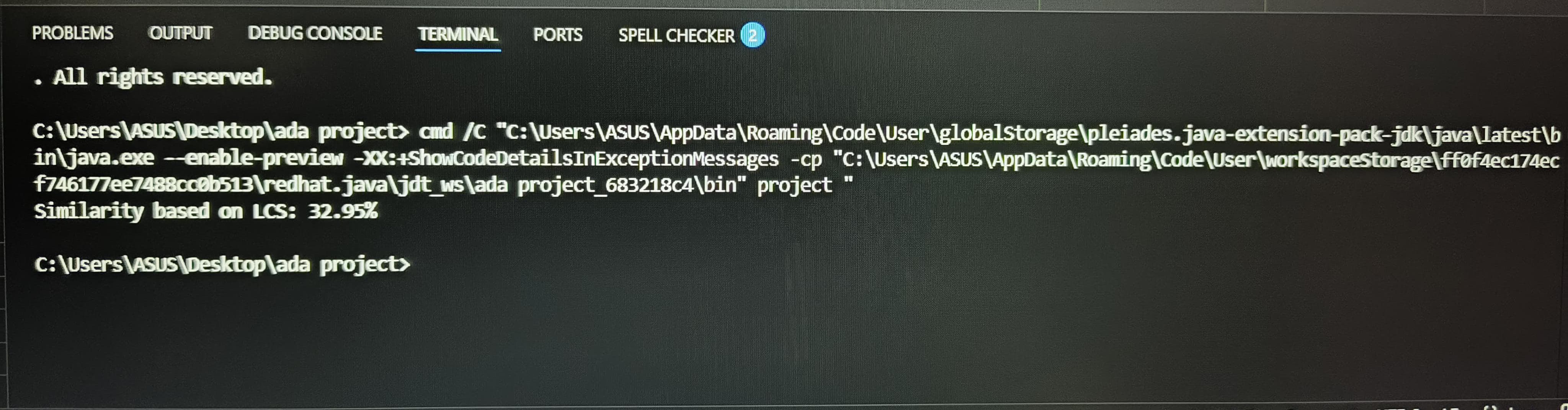
System.out.println("Error reading file: " + e.getMessage());

}

}

}

**6.Results**

****

**7.Conclusion**

The implementation of a plagiarism checker using the Longest Common Subsequence (LCS) algorithm provides a fundamental yet effective approach to detecting textual similarities between two documents. LCS works by identifying the longest sequence of characters or words that appear in both texts in the same order, even if they are not contiguous. This makes it suitable for detecting copied or slightly altered content where the structure remains similar but superficial changes like rewording or spacing have been applied.

The advantage of using LCS lies in its simplicity and clarity. It allows for a quantifiable similarity score based on how much of the content overlaps between two sources. For educational institutions and content creators, this can be a useful metric in flagging potential cases of plagiarism. It also supports preprocessing techniques such as case normalization and whitespace removal to enhance comparison accuracy.

However, while LCS is valuable for basic plagiarism detection, it is not foolproof. It may miss complex forms of plagiarism, such as paraphrasing or idea theft, and does not consider semantic meaning. For more comprehensive detection, LCS could be integrated with more advanced techniques like fingerprinting, machine learning, or natural language processing.

In conclusion, LCS-based plagiarism detection is a useful starting point for identifying direct or slightly modified content overlaps. It provides a foundation upon which more sophisticated systems can be built, balancing simplicity with functional utility in academic or content integrity contexts.

**8.References**

Here are some references you can use to support a project or report on a Plagiarism Checker using Longest Common Subsequence (LCS):

**📚 Academic References**

**1. Gusfield, D. (1997).**

Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology

Cambridge University Press.

A comprehensive book that covers LCS and related string algorithms.

**2. Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009).**

Introduction to Algorithms (3rd ed.)

MIT Press.

Chapter on dynamic programming includes detailed explanation of the LCS algorithm.

**3. Ramesh, D., & Kesavulu, B. (2014).**

“A Survey on Plagiarism Detection Techniques”

International Journal of Computer Applications, 100(16).

DOI:10.5120/17696-8563

Discusses LCS and other common algorithms used in plagiarism detection.

**4. GeeksforGeeks – Longest Common Subsequence**

https://www.geeksforgeeks.org/longest-common-subsequence-dp-4/

Detailed code examples and explanation of how LCS works.

**5. TutorialsPoint – LCS Algorithm**

https://www.tutorialspoint.com/Longest-Common-Subsequence

A simplified tutorial explaining the steps and code to implement LCS.

**6. ResearchGate – Plagiarism Detection Using LCS**

You can find papers and projects where LCS is used in plagiarism detection by searching:

https://www.researchgate.net/

If you are writing a formal report, you can cite the following like this (APA style):

Gusfield, D. (1997). Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology. Cambridge University Press.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms (3rd ed.). MIT Press.

Ramesh, D., & Kesavulu, B. (2014). A Survey on Plagiarism Detection Techniques. International Journal of Computer Applications, 100(16). https://doi.org/10.5120/17696-8563