**Chandigarh University**  
**Mini Project Report**  
**Plagiarism Detection System**  
**Submitted by:** Gurleen Saini,RUCHI,,,….

**Extract and Outcome**

**Extract**

The **Plagiarism Detection System** is designed to address the challenge writers face in detecting plagiarized content, which can often lead to legal issues. This project leverages various Data Structures and Algorithms (DSA) concepts in C to analyze and compare textual documents. It integrates text preprocessing, efficient string matching, and clustering techniques to identify similarities between documents.

**Outcome**

Upon successful completion, this project will help you:

* **Implement String-Matching Algorithms:** Learn and use algorithms like Knuth-Morris-Pratt and Boyer-Moore for effective text comparison.
* **Utilize Data Structures:** Employ hash tables, tries, suffix arrays, and suffix trees for indexing and managing document data.
* **Optimize for Efficiency:** Optimize algorithms to handle large datasets, ensuring that the system scales well for extensive document collections.
* **Develop Practical Skills:** Gain expertise in text preprocessing (tokenization, stemming, stop word removal) and similarity measures (cosine similarity, Jaccard index, n-gram matching).
* **Build a Functional Tool:** Create a user-friendly interface for uploading documents and visualizing plagiarism detection results, with potential applications in education, publishing, legal sectors, and content creation.

**System Configuration and Concepts Used**

**System Configuration**

* **Programming Language:** C
* **Compiler:** GCC
* **Operating System:** Windows 10
* **Development Environment:** VS Code

**Concepts and Techniques**

* **Data Structures:**
  + **Hash Tables / Tries:** Used for storing and indexing document text efficiently.
  + **Suffix Arrays / Suffix Trees:** Employed for advanced pattern matching and rapid substring searches.
  + **Arrays & Linked Lists:** Fundamental structures to handle and iterate through text data.
* **Algorithms:**
  + **String Matching:**
    - *Knuth-Morris-Pratt (KMP) Algorithm*
    - *Boyer-Moore Algorithm*
  + **Text Preprocessing:**
    - *Tokenization* – Splitting text into meaningful elements.
    - *Stemming* – Reducing words to their root form.
    - *Stop Word Removal* – Eliminating common words that add little value.
  + **Similarity Measures:**
    - *Cosine Similarity*
    - *Jaccard Index*
    - *n-gram Matching*
  + **Clustering Techniques:**
    - *K-means* and *Hierarchical Clustering* for grouping similar documents.
  + **Locality-Sensitive Hashing (LSH):** For handling large document collections with high efficiency.
* **DSA and Programming Constructs:**
  + **Loops and Iterations:** For traversing arrays and lists during data processing.
  + **Functions:** Modular design of code to encapsulate tasks such as preprocessing, matching, and similarity calculation.
  + **Pointers:** Critical for dynamic memory management and efficient manipulation of data structures.
  + **Structures:** To define complex data types that represent documents, tokens, and hash table entries.
* **Additional Resources and Code Structure:**
  + **Data Preprocessing Libraries:** Custom functions to tokenize and normalize text.
  + **Modular Design:** Each major function (e.g., text cleaning, string matching, clustering) is implemented in separate modules to ensure clarity and maintainability.
  + **Documentation:** Inline comments and a README file will be provided to explain the code structure, usage instructions, and potential areas for future improvements.

**Project Execution Details**

**What It Takes to Execute This Project**

* **Data Structures & Indexing:**
  + Utilize a hash table or trie to store and quickly access document text.
* **Text Preprocessing:**
  + Implement algorithms for tokenization, stemming, and stop word removal.
* **String Matching Algorithms:**
  + Deploy efficient string matching techniques like KMP and Boyer-Moore to detect plagiarized segments.
* **Similarity Measures:**
  + Compute document similarity using cosine similarity, Jaccard index, or n-gram matching.
* **Clustering for Document Grouping:**
  + Use k-means or hierarchical clustering to group documents with similar text.
* **Advanced Pattern Matching:**
  + Implement efficient data structures such as suffix arrays and suffix trees.
* **Scalability:**
  + Handle large collections using techniques like shingling or locality-sensitive hashing (LSH).
* **User Interface:**
  + Provide a simple UI for uploading documents and displaying detection results.
* **Integration:**
  + Optionally, integrate with external document databases for comprehensive checking.

**Real-World Applications**

**Educational Institutions**

* **Usage:** Detect plagiarism in academic papers and student assignments to maintain academic integrity.

**Publishing Houses**

* **Usage:** Ensure that content for articles, books, and multimedia is original before publication.

**Content Creators**

* **Usage:** Validate the originality of content before submission to clients or release on various platforms.

**Legal Firms**

* **Usage:** Analyze legal documents, contracts, and patents to detect unauthorized copying and safeguard intellectual property.

This detailed report not only presents the core idea of the plagiarism detection system but also explains the technical execution using C programming constructs, making it clear how loops, functions, pointers, and structures come together to form an efficient solution.

Would you like to add any diagrams, code snippets, or further explanations to any section?