Project Report: Plagiarism Detection System Using Data Structures

🔍 Abstract

This report presents the design and implementation of a **Plagiarism Detection System** that utilizes **hash tables** and **cosine similarity** for comparing textual similarity across multiple documents. The system tokenizes words, hashes them into a table, tracks frequency across documents, and computes the cosine similarity between frequency vectors. By leveraging these data structures and algorithms, the system efficiently identifies potential cases of plagiarism in academic or literary texts. The report explores algorithm design, time complexity, and the benefits of using hashing in textual analysis.

---

1. □ Introduction

In an academic world drowning in assignments, plagiarism detection has become more critical than ever. Traditional methods are slow and inconsistent, hence the need for automation. This project offers a lightweight, in-memory solution for comparing multiple documents and calculating their **textual similarity** using a **frequency-based model** and **cosine similarity metric**.

Objectives

- Implement an automated system for plagiarism detection using data structures.
- Use hashing to store and retrieve word frequencies efficiently.
- Apply cosine similarity to quantify similarity between documents.
- Analyze and visualize document similarity results.

---

2. □ Algorithms and Data Structures Used

2.1 Data Structure: Hash Table

A **hash table** is used to store unique words as keys and an array of frequencies (one per document) as values.

Structure of an Entry:

```
typedef struct {
   char word[MAX_WORD_LEN];
   int freq[MAX_DOCS];
} WordEntry;
```

Why Hash Tables?

- **O(1)** average-case insertion and lookup.
- Efficient handling of word frequency data.
- Avoids duplicates automatically through hashing.
- Collisions resolved via **linear probing**.

2.2 Cosine Similarity Algorithm

Cosine similarity measures the angle between two frequency vectors:

similarity(A,B)=A⋅B∥A∥∥B∥\text{similarity}(A, B) = \frac{A \cdot B}{\|A\|\|B\|}similarity(A,B)=∥A∥∥B∥A⋅B

Where:

- A⋅BA \cdot BA⋅B is the dot product.
- ∥A∥\|A\|∥A∥ and ∥B∥\|B\|∥B∥ are the magnitudes of the vectors.

Time Complexity:

- O(n), where n = number of hashed words.

---

3. 🛠 Implementation Details

3.1 Document Processing

- Each document is tokenized by whitespace and punctuation.
- Tokens are hashed and added to the global hash table.
- Frequencies are incremented per document index.

Tokenization snippet:

```
char *token = strtok(text, " .,;:\"\n\t\r");
```

3.2 Similarity Calculation

- After all documents are processed, cosine similarity is calculated pairwise.
- Results are printed as percentage similarity between each document pair.

Output Sample:

```
Similarity between Doc 0 and Doc 2: 85.44%
Similarity between Doc 1 and Doc 3: 10.27%
```

---

4. ✅ Advantages of This Approach

- **Efficient Memory Usage**: Only unique words stored.
- **Scalable**: Easily supports more documents with minimal change.

- **Fast Comparison**: Cosine similarity allows for quick and meaningful document comparisons.
- **Collision Handling**: Linear probing ensures no data is lost during hash collisions.

---

5. □ Conclusion

The Plagiarism Detection System demonstrates the power of **hash tables** and **cosine similarity** in identifying textual similarities. With efficient frequency tracking and comparison, the system offers a foundational tool for spotting plagiarism with minimal overhead.

Key Takeaways:

- Hash tables simplify word frequency tracking.
- Cosine similarity provides a robust, math-based similarity metric.
- Linear probing handles collisions gracefully in hash tables.
- Lightweight, effective, and beginner-friendly design.

🚀 Future Enhancements

- Add stemming and case-folding for better matching.
- Implement TF-IDF for deeper context-aware similarity.
- Use priority queues to rank most similar document pairs.
- Add GUI or web interface for uploading and comparing documents.

GitHub Link

https://github.com/ruchitaneja12/Plagiarism-detection-system

References

1. Text Mining and NLP with Hash Tables – ACM
2. Cosine Similarity in Text Analysis – Sciencedirect
3. Hash Table Collision Handling Techniques – GeeksForGeeks
4. Plagiarism Detection Algorithms – SpringerLink