

BINUS INTERNATIONAL
OBJECT ORIENTED PROGRAMING & DATA STRUCTURE
FINAL REPORT



Created By:

Evelyn Jane Sutjiadi - 2802501054

Ryan Alexander Kurniawan - 2802530584

Table of Contents

Table of Contents.....	2
CHAPTER I.....	2
1.1 Background.....	3
1.2 Problem.....	4
1.3 Solution.....	4
CHAPTER 2.....	6
2.1 Data Structures Used.....	6
2.1.1 HashSet.....	6
2.1.2 ArrayList.....	6
2.1.3 Ternary Search Tree.....	6
2.2 Class Diagram.....	7
CHAPTER 3.....	9
3.1 Evidence of Working Program.....	9
3.2 Results.....	13
3.2.1 Time.....	13
3.2.2 Space.....	14
3.2.3 Comparison.....	16
3.3 Implementation.....	16
REFERENCES.....	17

CHAPTER I

INTRODUCTION

1.1 Background

As students, we have encountered many obstacles in our path. One of those pressing challenges lies in any mention of plagiarism in one's work. Presenting work or ideas from another source as your own, with or without consent of the original author, by incorporating it into your work without full acknowledgement. [1]. As defined in the University of Oxford's guidelines towards plagiarism, whether intentionally or not, plagiarism harms all academic aspects of our day-to-day lives.

To this day, plagiarism has been ever-evolving in manufacturing pieces of literature for all students across the world. What used to be a grueling task has taken a steep dive in difficulty with today's tools. We believe this technology could be the root cause of the spike in plagiarism cases worldwide, not just in academics. The inappropriate fabrication of research works has serious consequences for the fabricator, the fabricated, and the scientific community that relies on the integrity of these publications to make informed decisions about changes in sociology, economics, politics, and medicine, amongst others. [2] These problems sprout as a student and as a member of any linguistic form of communication.

As students, it is hard to find a reliable plagiarism checker that has a high percentage of accuracy while maintaining the reliability it proposes in detecting similarities within text. These imperfections have pushed us towards creating similar functioning programs using Java, which harbors simpler string-to-string checking to identify similarities and the similarity percentage between two pieces of literature. Current plagiarism detection technologies, such as text-matching software, have proven effective in identifying direct copying from established sources. However, they often struggle with nuanced forms of plagiarism, including paraphrasing and AI-generated text that closely mimics human writing. [3]. We understand that innovations must be made to enhance these checkers for all students further, so we must understand the fundamentals of how these checkers operate at such a level beforehand.

1.2 Problem

Plagiarism Checkers are essential tools used online to check literature to create an integral environment within academic and non-academic works of literature. When implementing these programs, some problems that we might run into are:

1. Storage
 - a. The storage of compared text in a small case should not be an issue; however, comparing a text file with a database will be difficult as managing the small-scale comparison on a string-to-string basis with every file will exceed the limit of what can be stored within one lifespan of the run code.
2. Comparison
 - a. A simple text-to-text file comparison can be done easily using recursive string-to-string comparison; however, this idea of code has issues with upscaling the program to compare hundreds of files at once.
3. Speed
 - a. As stated in the storage problem, another problem that arises when presented with a comparison on a larger scale is the speed at which it compares itself on a string-to-string basis with the many files within the database. If the current system in place is used, this can result in the program taking ages to run and finish its check.

1.3 Solution

We decided to build an Java-Based Plagiarism checker that detects plagiarism by comparing it with another file. This checker would store document texts, compare the document, and detect copied content, then show the percentage of plagiarism done by the user. However, there will be changes and improvements that we, as a group, make along the way, but that's the solid idea for now. We are also planning to use 4 data structure options for now, the SuffixTree, used for fast string matching and finding text patterns efficiently. Next, HashMap, which is used for rapid storage and lookup of document fragments. This would also speed up comparisons with other documents out there. We are also going to use inverted index, which quickly locates documents containing specific words / phrases and finally, we will also be using Trie (prefix tree)

to help detect partial plagiarism by finding prefix overlaps and also provides efficient substring searching, which is useful for scanning long documents quickly.

CHAPTER 2

PROJECT SPECIFICATION

2.1 Data Structures Used

2.1.1 HashSet

Hash set is an unordered data structure that allows no duplicates. Similar to Array lists, hash sets can store packets of information inside their hash tables. Each packet of data is unique inside the hash tables. Furthermore hash sets allow for a dynamic sizing for storing data, automatically resizing themselves as needed when adding an item or deleting them. The time complexity for most operations done in hash sets, add/remove, is around the range of $O(1)$. One example in the code is being used as an anchor for STOPWORDS, the function being to quickly check if a word is within the STOPWORDS list.

2.1.2 ArrayList

Arraylist similar to hashsets is an iterable data structure with a non fixed size when adding or removing items. Arraylist differs from hashset in terms of ordering, while hashset has no order, arraylist has an insertion order. Among other differences, array lists do allow duplicates to be present in the structure. Another difference between arraylists and hashsets is the time complexity, the lookup time in arraylists is longer due to the nature of the ordered structure, being $O(n)$ compared to the previous $O(1)$. The application of Arraylists in our code comes in the storage of the sequence of cleared words from the files, maintaining a structured order.

2.1.3 Ternary Search Tree

Ternary Search Tree (TST) is a hybrid between a binary search tree and a trie. The nodes in TST represent a space that can hold up to three children. The left child is for characters less than current, middle is for the next character in the string, and

the right is for greater characters. It is specifically suited for lookup functions and is considered the most memory efficient due to the 3 child system. The TST tree structure stores strings by character, optimizing for memory usage and retrieval.

2.2 Class Diagram

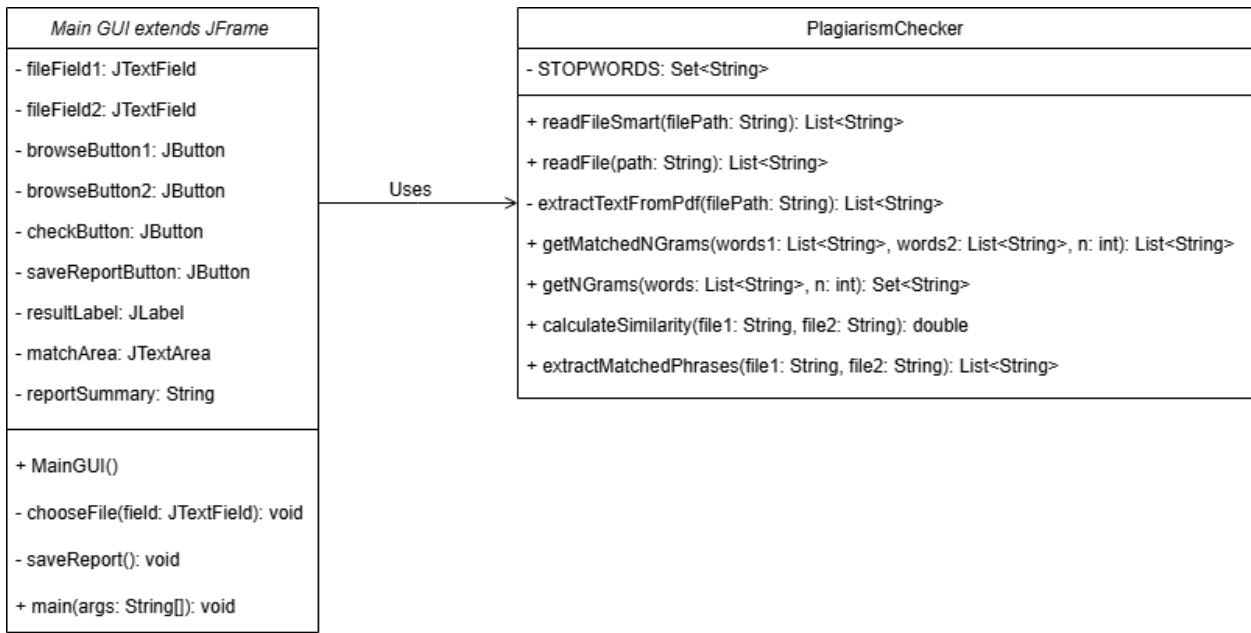


Figure 1.1 Class Diagram Main Class

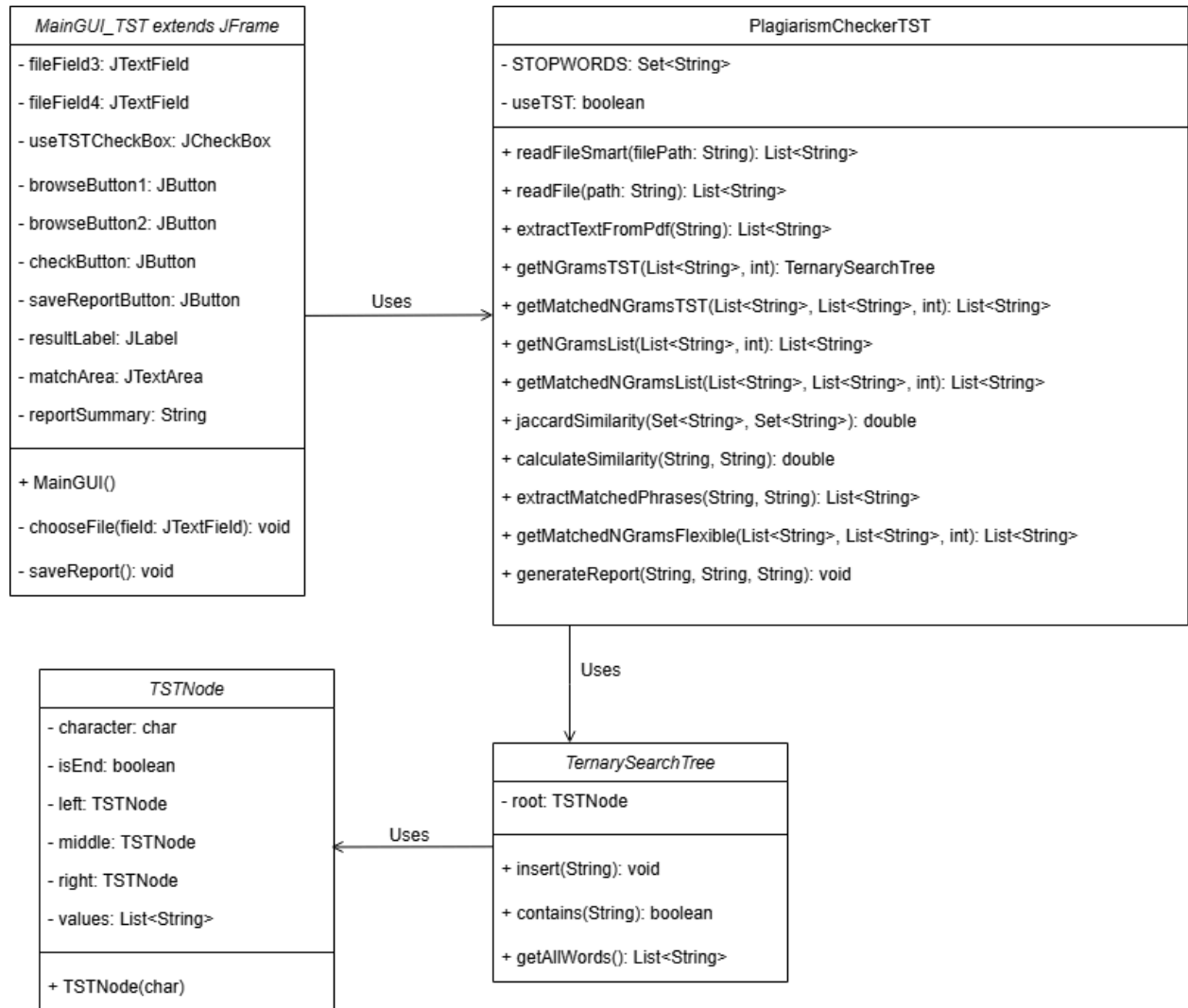
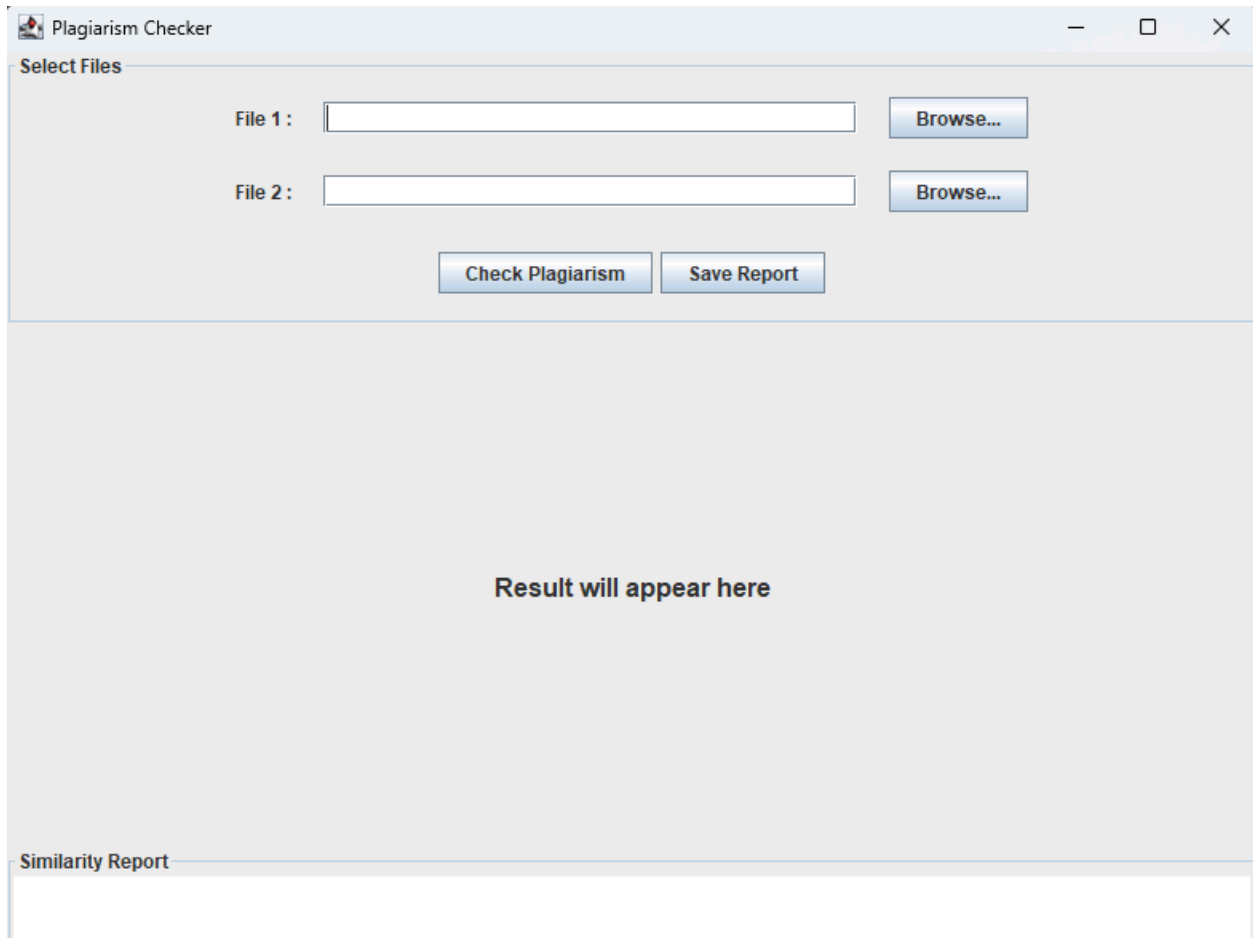


Figure 1.2 Class Diagram Main Class TST and Array List

CHAPTER 3

RESULTS

3.1 Evidence of Working Program



The screenshot shows a window titled "Plagiarism Checker" with standard Windows window controls (minimize, maximize, close). The window is divided into three main sections. The top section, titled "Select Files", contains two rows of input fields. The first row is labeled "File 1:" and the second "File 2:". Each row has a text input field and a "Browse..." button to its right. Below these fields are two buttons: "Check Plagiarism" and "Save Report". The middle section is a large, empty gray area with the text "Result will appear here" centered in it. The bottom section is titled "Similarity Report" and contains a large, empty white rectangular area for displaying the results.

Image 3.1 UI for Plagiarism Checker before Input

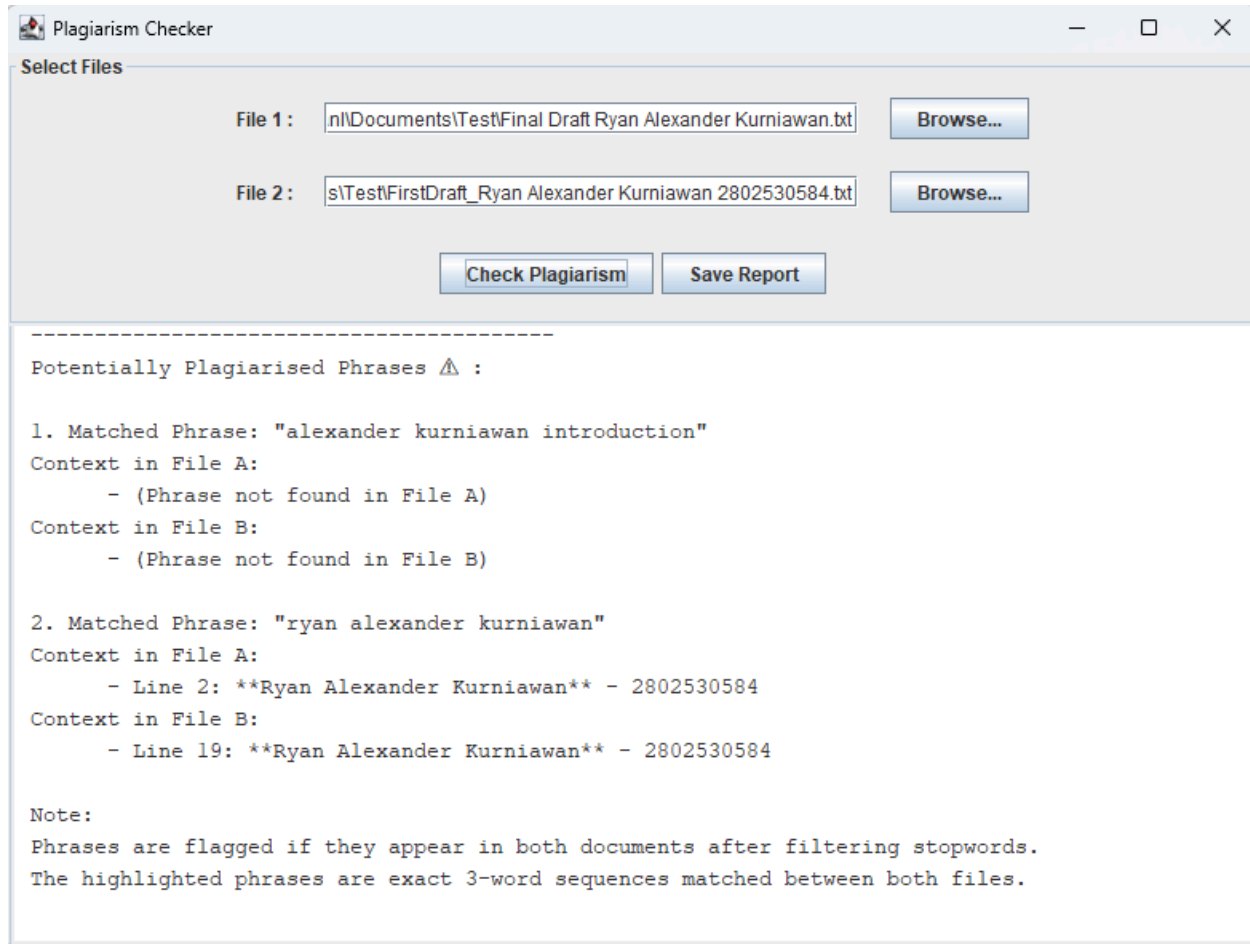


Image 3.2 UI After checking two files

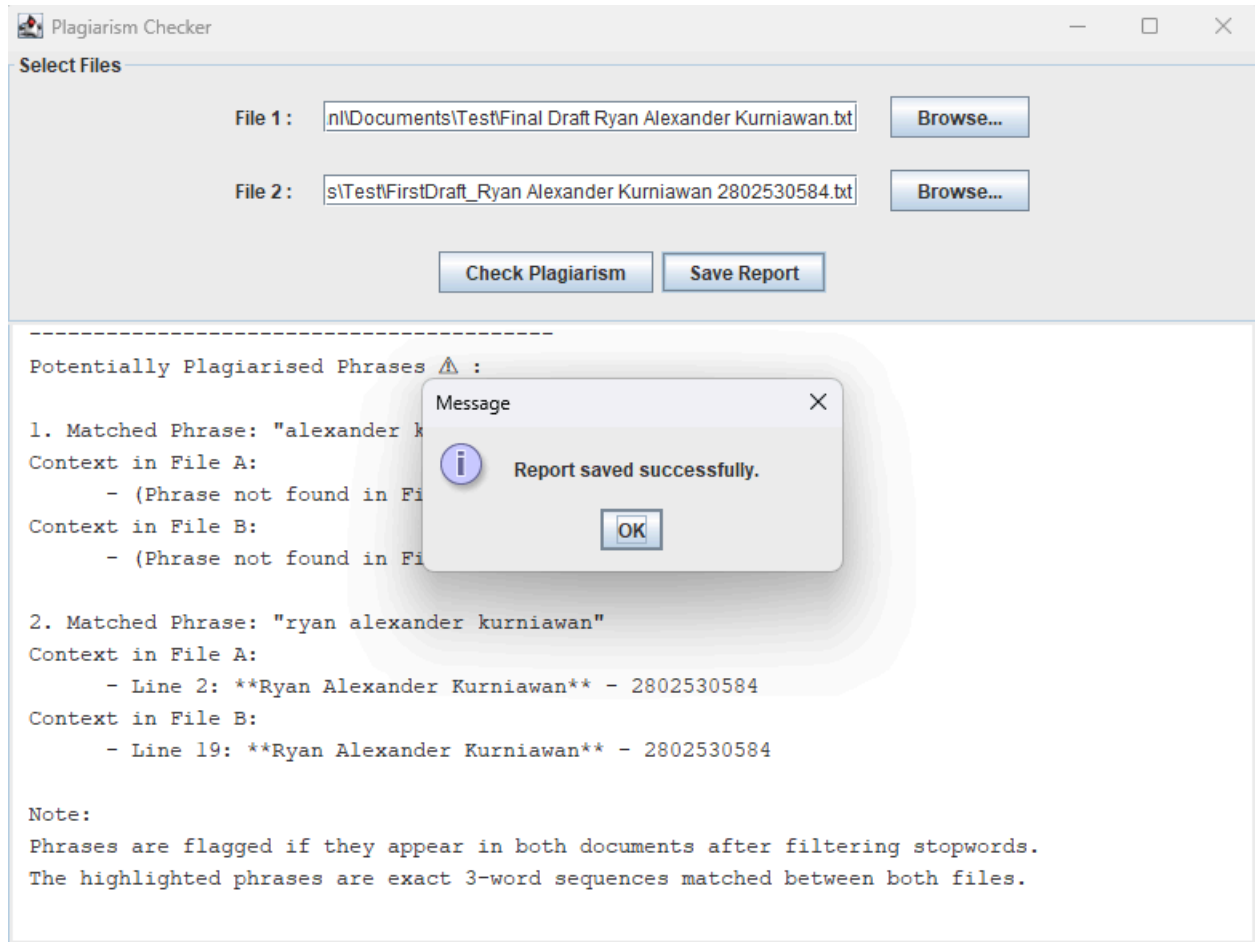


Image 3.3 UI After successfully saving the report file into a folder

Plagiarism Checker

Select Files

File 1 :

File 2 :

☒ Use Ternary Search Tree (Fast Match)

Result will appear here

Similarity Report

Image 3.4 UI for Plagiarism Checker using TST or Array List

```

Plagiarism Detection Report
=====
Compared Files:
- File A: C:\Users\Ryan1\Documents\Test\Lack of Sleep Final.txt
- File B: C:\Users\Ryan1\Documents\Test\Lack of Sleep.txt
Overall Similarity: 18.48%
=====
Potentially Plagiarised Phrases Δ :

1. Matched Phrase: "often stretched thin"
Context in File A:
- Line 3: Throughout the span of university, students are **often stretched thin** between many responsibilities, academic, social obligations, part time work, and extracurricular activities. Reports from students say as much as 50% have daytime sleepiness and 70% attain insufficient sleep (Hershner, S. D, et al. 2014). As a result, a critical measure of health and performance is the sleep they get. Sleep deprivation in students has become a widespread issue among many universities. Among the many factors that contribute to a students academic struggle, a major cause of the decline in performance is directly tied to sleep deprivation. Consequences of constant sleep deprivation does not stop at feeling tired throughout your days, sleep loss has measurable effects on academic prowess, mental health, and cognitive function.
Context in File B:
- Line 21: Throughout the span of university, students are **often stretched thin** between many responsibilities, academic, social obligations, part time work, and extracurricular activities. As a result, a critical measure of health and performance is the sleep they get. Sleep deprivation in students has become a widespread issue among many universities. Among the many factors that contribute to a students academic struggle, a major cause of the decline in performance is directly tied to sleep deprivation. Consequences of constant sleep deprivation does not stop at feeling tired throughout your days, sleep loss has measurable effects on academic prowess, mental health, and cognitive function.

2. Matched Phrase: "sleep they get"
Context in File A:
- Line 3: Throughout the span of university, students are often stretched thin between many responsibilities, academic, social obligations, part time work, and extracurricular activities. Reports from students say as much as 50% have daytime sleepiness and 70% attain insufficient sleep (Hershner, S. D, et al. 2014). As a result, a critical measure of health and performance is the **sleep they get**. Sleep deprivation in students has become a widespread issue among many universities. Among the many factors that contribute to a students academic struggle, a major cause of the decline in performance is directly tied to sleep deprivation. Consequences of constant sleep deprivation does not stop at feeling tired throughout your days, sleep loss has measurable effects on academic prowess, mental health, and cognitive function.
Context in File B:
- Line 21: Throughout the span of university, students are often stretched thin between many responsibilities, academic, social obligations, part time work, and extracurricular activities. As a result, a critical measure of health and performance is the **sleep they get**. Sleep deprivation in students has become a widespread issue among many universities. Among the many factors that contribute to a students academic struggle, a major cause of the decline in performance is directly tied to sleep deprivation. Consequences of constant sleep deprivation does not stop at feeling tired throughout your days, sleep loss has measurable effects on academic prowess, mental health, and cognitive function.

```

Image 3.5 Example of Saved Report in txt form

3.2 Results

The expected results of our code is a UI design that is simple yet clear to understand for the average user, sporting two simple sections. The first section involves selecting the files, with browsing buttons that lead to each individual's files, a start checking button, and a save report button. The second UI has one different feature since we are using a different type of checking method the second version supports a check mark on using the TST (Ternary Search Tree) or if ticked off uses the data structure Array List instead. The second section of the UI supports the results of the plagiarism check, which can also be saved using the save report button.

To compare the time and memory consumption that each data structure combination uses, we prepared the 3 conditions that they have to fulfill each. Each program will have to test two of the same files, two different files, and one small file with a large file. Each test is run 3 times to gather the average time and memory consumption.

3.2.1 Time

The tables below regarding Time is counted in milliseconds and recorded in microseconds to get an accurate decimal reading.

	Data Structures		
Conditions	HashSet & Array list	Ternary Search Tree	Array list
Two Same Files	22.8	29.721	29.478
	19.89	25.554	25.511
	12.199	12.192	12.257
Average	18.29633333	22.489	22.41533333

Table 3.1 Testing with Two of the same files

	Data Structures		
Conditions	HashSet & Array list	Ternary Search Tree	Array list

Two Different files with similar size	2.586	3.799	3.188
	2.483	2.911	1.94
	2.798	2.828	2.179
Average	2.622333333	3.179333333	2.435666667

Table 3.2 Testing with Two different files with similar size

	Data Structures		
Conditions	HashSet & Array list	Ternary Search Tree	Array list
Small file against large file	43.895	34.077	19.727
	23.528	24.048	19.309
	23.212	24.345	19.099
Average	30.21166667	27.49	19.37833333

Table 3.3 Testing with small against a large file

From the data stated above, on average purely using arraylist for all the conditions will result in the most efficient time complexity. When presented with individual tables the array list by itself has the fastest time (in ms) on both the similar sized files and the small against large. Hash set and arraylist are the most efficient for two of the same files comparison while ternary search tree has neither the most efficient or least efficient on all the conditions.

3.2.2 Space

The Tables regarding Space is counted in mega bytes, and calculated from bytes to get an accurate consideration of decimals.

	Data Structures		
Conditions	HashSet & Array list	Ternary Search Tree	Array list

Two Same Files	10.677952	11.138488	10.928976
	10.721448	10.742904	13.026128
	10.60268	10.537888	10.678152
Average	10.66736	10.80642667	11.54441867

Table 3.4 Testing with Two of the same files

	Data Structures		
Conditions	HashSet & Array list	Ternary Search Tree	Array list
Two Different files with similar size	0.961624	1.084936	2.097152
	1.922776	1.068552	2.885704
	1.923248	1.06868	1.3086
Average	1.602549333	1.074056	2.097152

Table 3.5 Testing with Two different files with similar size

	Data Structures		
Conditions	HashSet & Array list	Ternary Search Tree	Array list
Small file against large file	25.27664	30.208528	43.868864
	44.286016	44.832456	43.891152
	61.06988	40.556616	44.040192
Average	43.54417867	38.53253333	43.93340267

Table 3.6 Testing with small against a large file

Regarding the data above we can conclude that on average the best space complexity goes to the ternary search tree. Being most efficient in comparing two different files of similar size and small against large, ternary search tree has a significant lead compared to the rest of the

data structures. The reason being is due to TST being the most memory efficient using their three child system for similar lookup for certain words.

3.2.3 Comparison

All three data structures work by storing phrases of N words as single strings. The TST splits each phrase into characters and organizes them into a tree, enabling memory efficient search/lookup. The ArrayList holds each phrase as an entry, ideal for ordered processing but slow for searching. HashSet eliminates duplicate N-grams and allows fast detection of shared phrases. Depending on whether uniqueness, order, or search speed is more important, each structure brings different advantages to N-gram processing and comparing them with each other.

3.3 Implementation

The results of our tests suggest that on average, for time efficiency arraylist is the best, however for the best memory consumption ternary search tree is the best. Depending on the goals of the program, we can conclude that these two have their own respective uses. Overall, although for the increase of time in two of the same files, and two different files of similar size, the best data structure is Ternary Search Tree.

Ternary Search tree is different in sorting and retrieving characters because it uses a three child tree structure that stores strings by character. ArrayList stores full phrases directly in a list, with a maintained order but this ends up in slower lookup times. Meanwhile, Hashset uses the hashing method but does not have order, consuming more space. In the end, TST is best for matching phrases by common characters, Arraylist for simple iteration, and Hashset for speed and unique characters.

REFERENCES

- [1] <https://www.ox.ac.uk/students/academic/guidance/skills/plagiarism>
- [2] Elali, F. R., & Rachid, L. N. (2023). AI-generated research paper fabrication and plagiarism in the scientific community. *Patterns*, 4(3), 100706. <https://doi.org/10.1016/j.patter.2023.100706>
- [3] Muazu, M. (2024). AI and ethics: Academic integrity and the future of quality assurance in higher education. *Handbook on AI and Quality Higher Education in Honour of Prof. Abubakar Adamu Rasheed*, Volume 3. Sterling Publisher. Retrieved from https://www.researchgate.net/publication/384969429_AI_and_EthicsAcademic_integrity_and_the_future_of_Quality_Assurance_in_Higher_Education_Hand_Book_on_AI_and_Quality_Higher_Education_in_Honour_of_Prof_Abubakar_Adamu_Rasheed_Volume_3