OBJECT ORIENTED PROGRAMMING
DATA STRUCTURES

Extractive Text Summarizer

Evaluating the Effectiveness of HashMap, TreeMap, and ConcurrentSkipListMap for an Extractive Text Summarizer in Java

16 June 2025

Juan Felix Kusnadi - 2802536386
Jonathan Mulyono - 2802537054
Iglesias Sidharta Handjo - 2802530621

Hello!

Text summarization helps people quickly understand lengthy texts by highlighting key information. This project uses the TF-IDF algorithm to build an extractive summarizer in Java. We compare HashMap, TreeMap, and ConcurrentSkipListMap to evaluate their impact on performance and memory usage.

Problem Description

- The explosion of daily textual data makes it challenging for humans to efficiently extract key information
- Summarization tools condense lengthy texts into concise, meaningful overviews without losing essential content
- Extractive summarization selects and compiles the most important sentences
- This study analyzes how different Java Map implementations affect the performance and behavior of a TF-IDF-based extractive summarizer

Approach

Evaluate the importance of a word within a document relative to a collection of documents (corpus) (Jain, 2025)

Concept I

Words that appear more often in a document are more important (higher Term Frequency)

Concept II

Words that are common across different documents in the corpus are less important as they are less useful for distinguishing different documents (lower Inverse Document Frequency)

Methodology

Preparing the text input

Tokenize, remove stop words, convert lowercase, stem

Calculate Term Frequency (TF)

TF = w/S, where

w = number of times word w appears

S = number of words in sentence S

Calculate Inverse Document Frequency (IDF)

IDF = log (N/SF(w)), where

N = number of sentences

SF(w) = sentence frequency containing the word w

Creating a summary

Calculate TF-IDF by TF × IDF

Store this key (word) and value (TF-IDF) pair

Rank top sentences and output top 20%-40% (Demilie, 2022)

Data Structures

Map Implementations

HashMap

Unordered key-value store with hashing

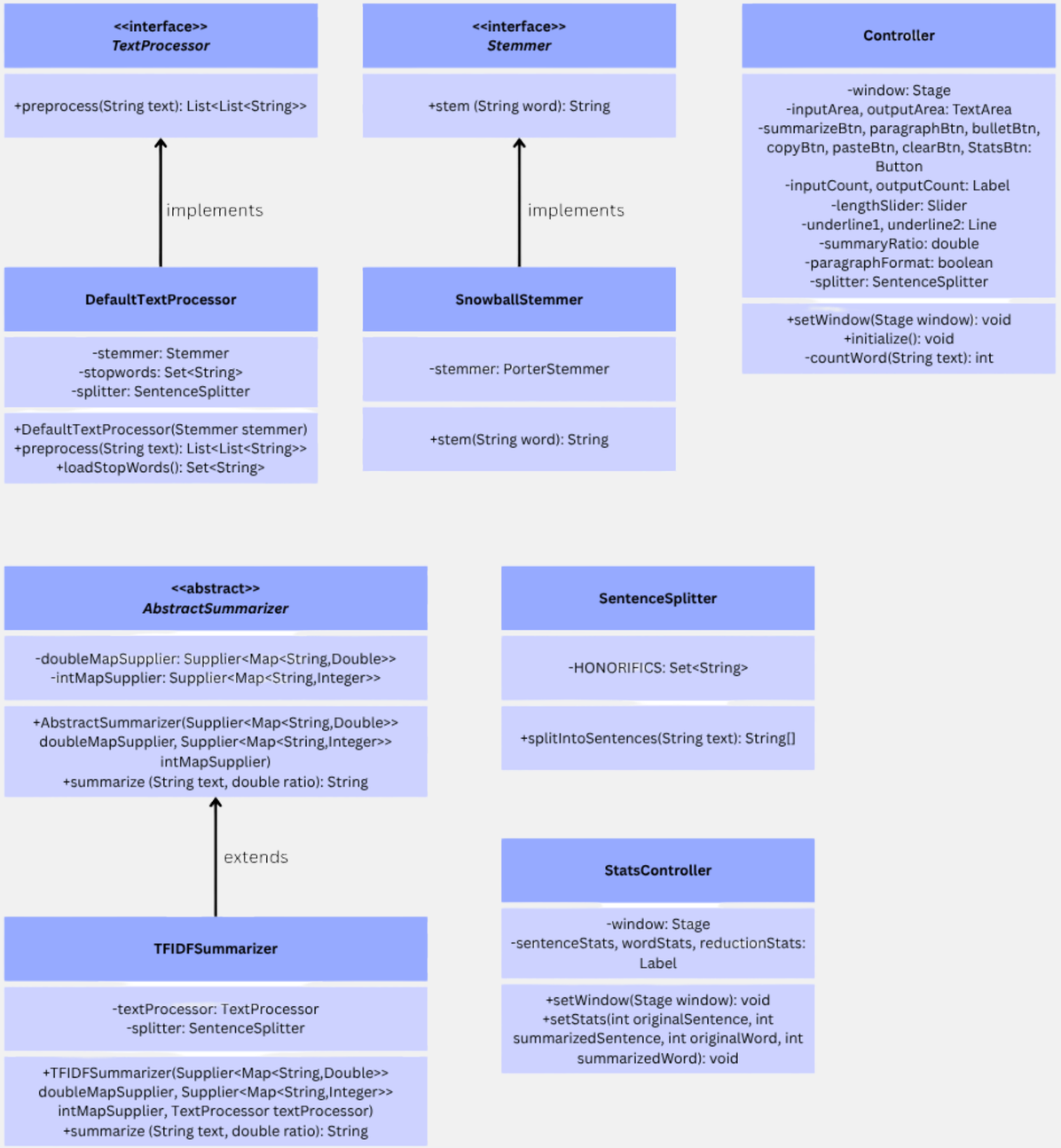
TreeMap

Sorted key-value map using tree

ConcurrentSkipListMap

Concurrent sorted map using skip-list

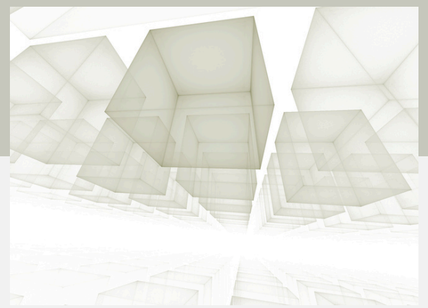
Program Demo


<<interface>>
TextProcessor
+preprocess(String text): List<List<String>>
implements
DefaultTextProcessor
-stemmer: Stemmer
-stopwords: Set<String>
-splitter: SentenceSplitter
+DefaultTextProcessor(Stemmer stemmer)
+preprocess(String text): List<List<String>>
+loadStopWords(): Set<String>
<<interface>>
Stemmer
+stem (String word): String
implements
SnowballStemmer
-stemmer: PorterStemmer
+stem(String word): String
Controller
-window: Stage
-inputArea, outputArea: TextArea
-summarizeBtn, paragraphBtn, bulletBtn, copyBtn, pasteBtn, clearBtn, StatsBtn: Button
-inputCount, outputCount: Label
-lengthSlider: Slider
-underline1, underline2: Line
-summaryRatio: double
-paragraphFormat: boolean
-splitter: SentenceSplitter
+setWindow(Stage window): void
+initialize(): void
-countWord(String text): int
<<abstract>>
AbstractSummarizer
-doubleMapSupplier: Supplier<Map<String,Double>>
-intMapSupplier: Supplier<Map<String,Integer>>
+AbstractSummarizer(Supplier<Map<String,Double>> doubleMapSupplier, Supplier<Map<String,Integer>> intMapSupplier)
+summarize (String text, double ratio): String
extends
TFIDFSummarizer
-textProcessor: TextProcessor
-splitter: SentenceSplitter
+TFIDFSummarizer(Supplier<Map<String,Double>> doubleMapSupplier, Supplier<Map<String,Integer>> intMapSupplier, TextProcessor textProcessor)
+summarize (String text, double ratio): String
SentenceSplitter
-HONORIFICS: Set<String>
+splitIntoSentences(String text): String[]
StatsController
-window: Stage
-sentenceStats, wordStats, reductionStats: Label
+setWindow(Stage window): void
+setStats(int originalSentence, int summarizedSentence, int originalWord, int summarizedWord): void

The 4 Principles of OOP

Encapsulation

Classes hide their internal data and helper methods, exposing only defined public methods.



Abstraction

Interfaces and the abstract base class define clear contracts for operations without revealing details.

Inheritance

Reusing common supplier wiring and constructor logic while providing its own summarize() implementation.

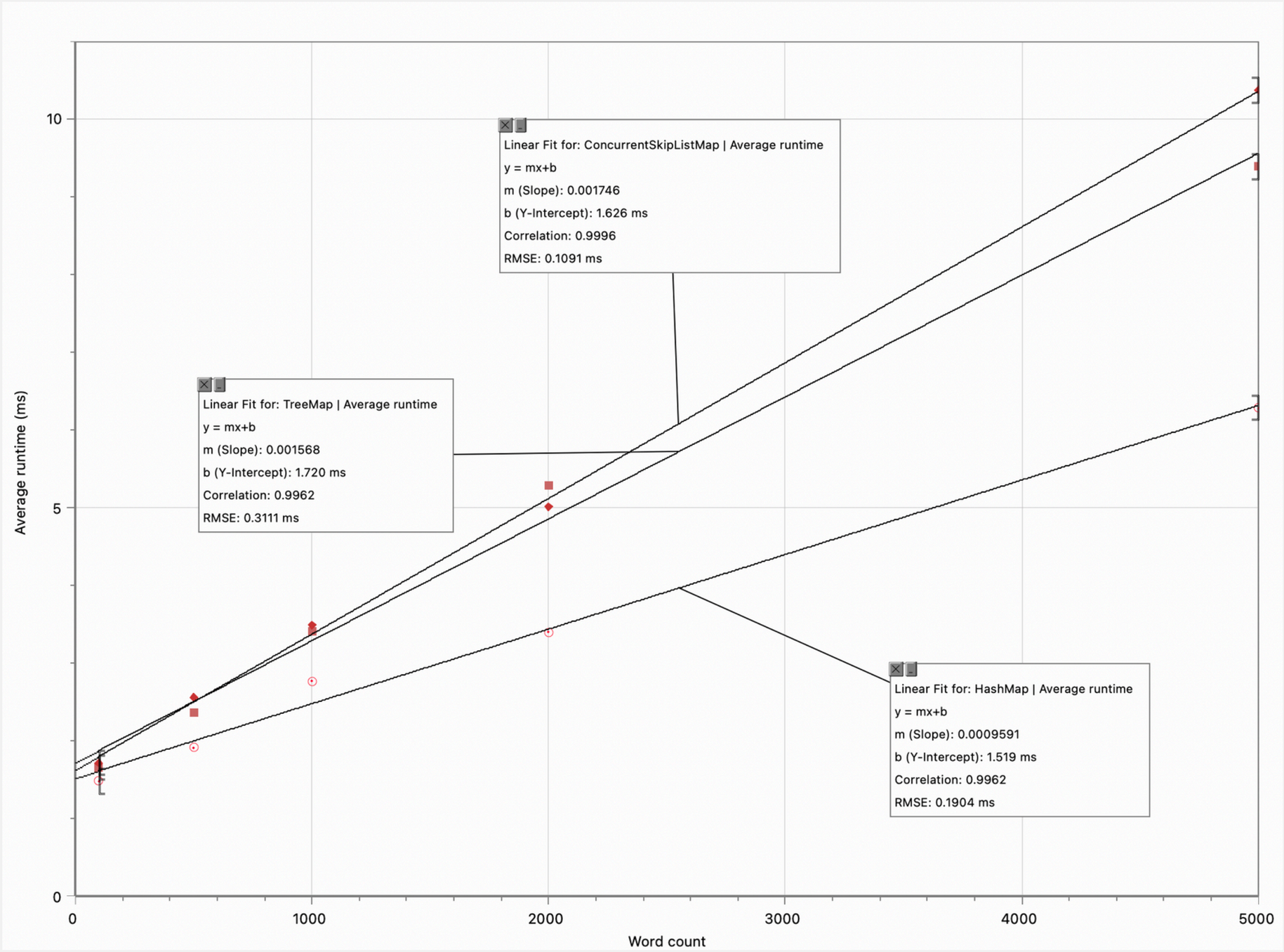
Polymorphism

Abstract references allow different summarizer or stemmer implementations to be interchanged at runtime.

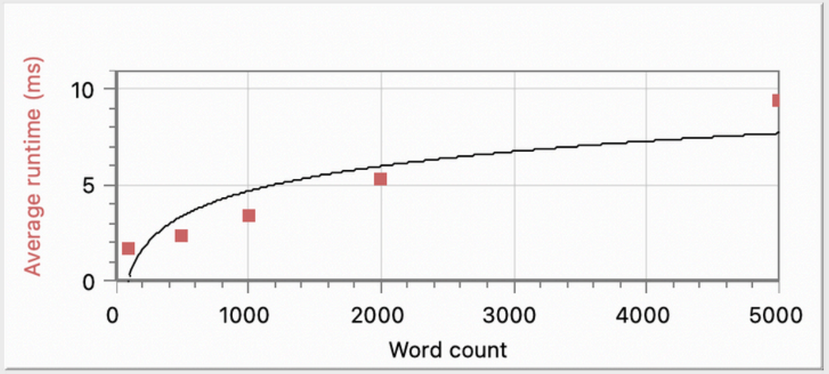
Input (AI Generated)	Output (medium summary length)
Cybersecurity is essential in today's ... keeping information secure. (100 words)	Cybersecurity is essential in today's ... sensitive personal data. (20 words)
Artificial intelligence (AI) is revolutionizing the healthcare industry ... determines their success in global health. (500 words)	Natural language processing (NLP) allows AI ... success in global health. (51 words)
Renewable energy refers to sources ... high-carbon infrastructure and transition directly. (1000 words)	Understanding how these sources work, their advantages ... renewable portfolio standards. (112 words)
Social media has become an integral part of modern society ... of social media while minimizing harm. (2000 words)	For example, passive use, defined as scrolling ... limits the ability to draw causal inferences. (124 words)
Coral reefs are among the most diverse and productive ... science to maintain credibility and trust. (5000 words)	Fertilization occurs externally, and resulting ... emphasizing the economic logic of proactive investment. (192 words)

Average Runtime (ms)	HashMap	TreeMap	ConcurrentSkipListMap
Trial 1	1.481	1.654	1.717
Trial 2	1.917	2.365	2.559
Trial 3	2.764	3.402	3.487
Trial 4	3.399	5.278	5.010
Trial 5	6.284	9.386	10.372

Average Memory (MB)	HashMap	TreeMap	ConcurrentSkipListMap
Trial 1	0.722	0.768	0.772
Trial 2	0.802	1.054	0.990
Trial 3	1.144	1.133	1.302
Trial 4	1.526	1.617	1.512
Trial 5	2.873	2.899	2.812


Linear Fit for: ConcurrentSkipListMap | Average runtime
y = mx+b
m (Slope): 0.001746
b (Y-Intercept): 1.626 ms
Correlation: 0.9996
RMSE: 0.1091 ms
Linear Fit for: TreeMap | Average runtime
y = mx+b
m (Slope): 0.001568
b (Y-Intercept): 1.720 ms
Correlation: 0.9962
RMSE: 0.3111 ms
Linear Fit for: HashMap | Average runtime
y = mx+b
m (Slope): 0.0009591
b (Y-Intercept): 1.519 ms
Correlation: 0.9962
RMSE: 0.1904 ms
Average runtime (ms)
Word count
0
5
10
0
1000
2000
3000
4000
5000

Curve Fit



General Equation

A*10^(Bx)+C	Base-10 Exponent
A*log(Bx)	Base-10 Logarithm
A*(1-exp(-C*x))+B	Inverse Exponent
A*sin(Bx+C)+D	Sine
A*cos(Bx+C)^2+D	Cosine Squared
1/(sqrt(2*pi)*S)*exp(-(x-M)^2/(2*S^2))	Normalized Gaussian

Time Offset | Define Function | Delete Function

Status:

?

Try Fit

Fit Type

Automatic | Manual

Create Calculated Column

Weight Column ----

Coefficients

A*log(Bx)

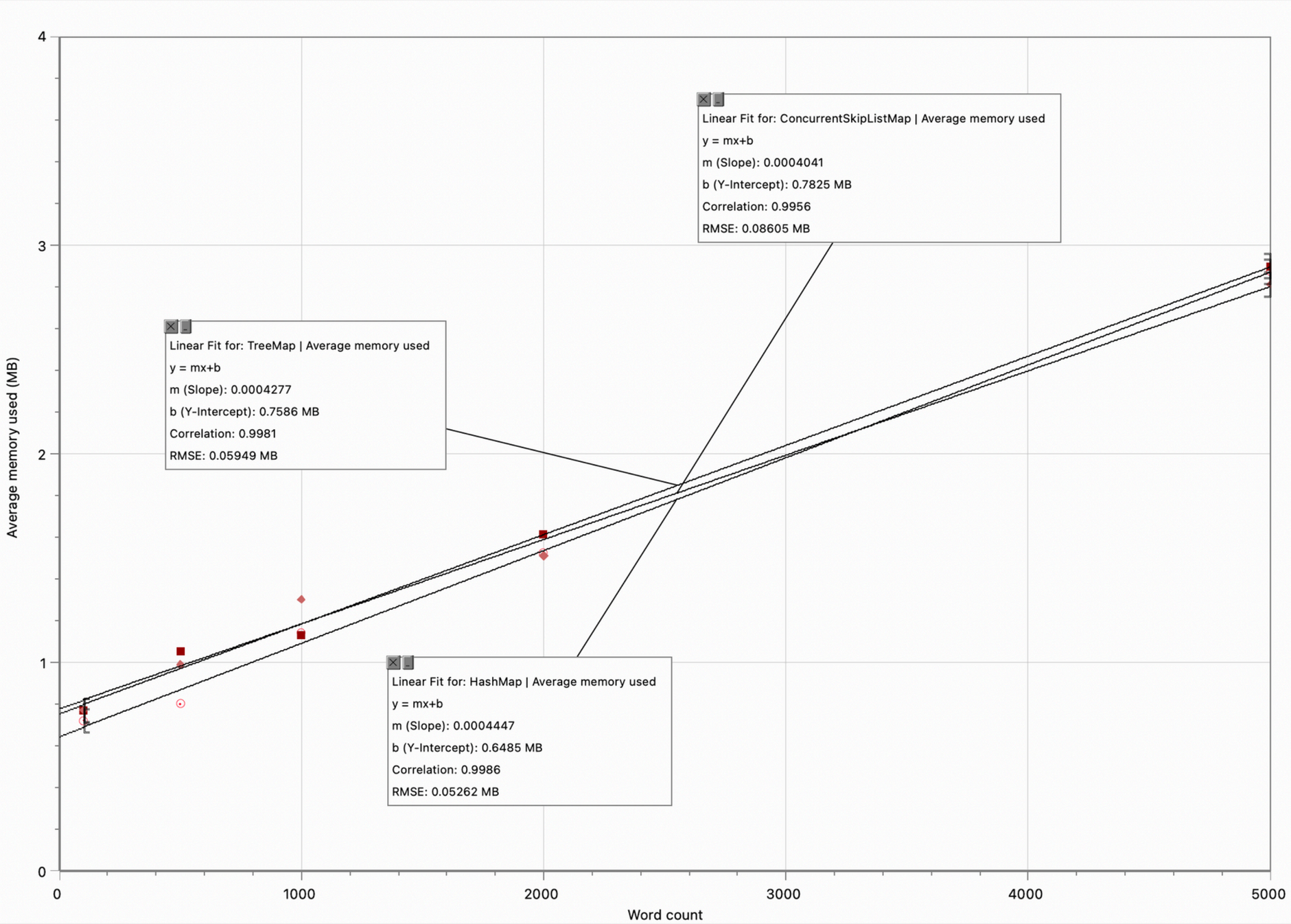
A 4.307 Δ

B 0.01219 Δ

Correlation: 0.894241

RMSE: 1.59940

Cancel | OK


Linear Fit for: ConcurrentSkipListMap | Average memory used
y = mx+b
m (Slope): 0.0004041
b (Y-Intercept): 0.7825 MB
Correlation: 0.9956
RMSE: 0.08605 MB
Linear Fit for: TreeMap | Average memory used
y = mx+b
m (Slope): 0.0004277
b (Y-Intercept): 0.7586 MB
Correlation: 0.9981
RMSE: 0.05949 MB
Linear Fit for: HashMap | Average memory used
y = mx+b
m (Slope): 0.0004447
b (Y-Intercept): 0.6485 MB
Correlation: 0.9986
RMSE: 0.05262 MB
Average memory used (MB)
Word count
0
1
2
3
4
0
1000
2000
3000
4000
5000

Conclusion

Performance:

HashMap > TreeMap > ConcurentSkipListMap

Memory behaviour:

HashMap ≈ TreeMap ≈ ConcurentSkipListMap

Best data structure: HashMap

1. Lowest time complexity
2. Reasonable space complexity

Evaluation

- Gradient of best fit for HashMap is non-zero (0.0009591)
- Caused by hash code computations and collisions
- String equals() operations may result in growth proportional to N

- Inability to use a logarithmic function for TreeMap and ConcurrentSkipListMap
- $\log(100) = 2$, $\log(5000) \approx 3.70$
- Dominated by the fixed-cost overheads, such as pointer traversal and tree rotation
- 5000 is too small to see asymptotical behaviour

References

- Demilie, W. (2022, September 13). Comparative analysis of automated text summarization techniques: The case of Ethiopian languages. *Wireless Communication and Mobile Computing*, *2022*, 13-15. https://doi.org/10.1155/2022/3282127
- Jain, S. (2025, February 7). Understanding TF-IDF (Term Frequency-Inverse Document Frequency). *GeeksforGeeks*. https://www.geeksforgeeks.org/understanding-tf-idf-term-frequency-inverse-document-frequency/

Thank You

Powered by Canva