Word Frequency Analysis Based on Binary-Search Tree and HashMap

Abstract

This project explores large-scale word frequency analysis, a common task in digital humanities and social computing, using two contrasting map implementations. I applied binary search trees (BST) and hash tables (HashMap) via a shared WordCounter pipeline to ingest and count words from two corpora: Reddit comments from 2015 and the complete works of Shakespeare. Across both datasets, HashMap construction was consistently faster (e.g., 940 ms vs. 4,944.6 ms on Reddit; 17.8 ms vs. 105.6 ms on Shakespeare), and BSTs exhibited much greater depth (maxDepth 38/35) than hash chains (6/5), which explains the performance gap through imbalance vs. near-constant-time insertions. Content-wise, top-word profiles reflect different styles in conversational fillers on Reddit and Early Modern pronouns/titles in Shakespeare, and show how corpus genre shapes frequency distributions.

Results

I designed the Experiments class to do three analyses: (1) top-10 words by corpus, (2) build times across BST and HashMap, and (3) post-build structural depth. All sub-experiments for analysis in the Experiments class used the provided WordCounter pipeline with identical tokenization and counting logic; only the backing map (BST vs. HashMap) varied. Metrics recorded were: total tokens, unique tokens, average build time over 5 runs (ms), and structure depth after construction (maxDepth for BST; maximum bucket chain length for HashMap). Datasets: CLEANED_reddit_comments_2015.txt (14,975,699 tokens, 43,968 uniques) and CLEANED_shakespeare.txt (365,342 tokens, 13,120 uniques). The data for the results is from report.txt under the src folder. Run and compile Experiments.java to get new data, and the result will be written to report.txt.

Analysis 1: Top-10 Words by Corpus (Reddit 2015 and Shakespeare)

- **Experiment**

I computed word counts and ranked tokens by frequency for each corpus. The tokenizer lowercases and strips punctuation (apostrophes removed), so “don’t” appears as dont. I did not apply stop-word removal; results reflect raw frequency.

- **Hypothesis**

I assume that Reddit (2015) should emphasize conversational, stance-taking, and filler language; Shakespeare should emphasize Early Modern pronouns/titles and verbs common to dramatic dialogue.

- **Table of Results**

Table 1. Top-10 Word Frequencies by Reddit 2015 (total number of tokens (N) = 14,975,699 tokens, number of unique tokens (U) = 43,968)

Rank	Word	Count
1	like	180,784
2	dont	137,848
3	would	130,045
4	get	124,499
5	one	123,123
6	people	111,374
7	think	98,193
8	really	85,492
9	time	82,538
10	good	80,540

The top counts cluster around conversational tokens, with the highest at 180,784 and a gradual decline to 80,540. The distribution reflects informal, discourse-heavy language; naive top-words analyses on social media corpora primarily capture register and stance rather than topical content. The data is from report.txt under the src folder.

Table 2. Top-10 Word Frequencies by Shakespeare (total number of tokens (N) = 14,975,699 tokens, number of unique tokens (U) = 43,968)

Rank	Word	Count
1	thou	5,485
2	thy	4,032
3	shall	3,591
4	thee	3,178
5	lord	3,059
6	king	2,861
7	good	2,812
8	sir	2,754

Rank	Word	Count
9	come	2,507
10	well	2,462

Pronouns/titles dominate, led by thou at 5,485 and tapering to well at 2,462, showing a steady drop across role- and relation-marking terms. Frequency peaks encode historical register and dramaturgy; analyses must account for genre-era effects (e.g., remove outdated function words) before drawing thematic conclusions. The data is from report.txt under the src folder.

- **Synthesis**

The findings from the data match the hypothesis. Reddit's leaders (like, dont, would, get, one, people, think) are discourse markers and general verbs, consistent with informal, argumentative, and anecdotal speech. Shakespeare is dominated by pronouns/titles (thou, thy, thee, lord, king, sir) and modal/verb forms (shall, come, well), consistent with scripted dialogue, hierarchy markers, and verse.

Analysis 2: Build Times Across Data Structure (BST and HashMap)

- **Experiment**

For each corpus and structure, I ran buildMap(words) five times and averaged the running times (in milliseconds). The WordCounter code path was identical across structures; only the backing map changed.

- **Hypothesis**

I expect HashMap inserts to be amortized O(1) with short chains; BST inserts are O(h), where h is the tree height. Thus, HashMap should build faster, with the gap widening on the larger corpus.

- **Table of Results**

Table 3. Average Build Time by Data Structure (BST and HashMap) and Corpus (Reddit 2015 and Shakespeare)

Corpus	Data Structure	Avg Build Time (ms)
Reddit 2015	BST	4,944.60
Reddit 2015	HashMap	940.00
Shakespeare	BST	105.60
Shakespeare	HashMap	17.80

For both corpora, HashMap build times are consistently lower; on Reddit the time drops from 4,944.6 ms (BST) to 940.0 ms (HashMap), and on Shakespeare from 105.6 ms to 17.8 ms. Insert cost remains near-constant in the hash table while BST cost inflates with height, making HashMap the practical choice for large, naturally ordered text streams. The data is from report.txt under the src folder.

- **Synthesis**

HashMap clearly outperforms BST on both datasets, and the relative advantage is larger on Reddit's scale. This aligns with expected operation costs: near-constant inserts vs. depth-dependent inserts that suffer when the BST becomes tall.

Analysis 3: Post-build Structural Depth

- **Experiment**

After building, I recorded maxDepth. For BST, this is the maximum root-to-leaf path length. For HashMap (chaining), this is the longest bucket chain length (i.e., worst-case probe length). I also compare BST depth to the ideal balanced height $\log_2(U)$, where U stands for unique tokens.

- **Hypothesis**

If BSTs are fed words in corpus order (not randomized), the resulting trees should be noticeably unbalanced (height $\gg \log_2 U$), inflating insertion cost. Hash chains should remain short, sustaining O(1) inserts.

- **Table of Results**

Table 4. Post-Build Structure Depth vs. Ideal Height

Corpus	U (Unique Tokens)	$\log_2(U)$	BST maxDepth	HashMap max chain
Reddit 2015	43,968	15.42	38	6
Shakespeare	13,120	13.68	35	5

BST heights far exceed balanced expectations (e.g., Reddit 38 vs. $\log_2(U) \approx 15.4$; Shakespeare 35 vs. ≈ 13.7), while HashMap's longest chain remains short (≤6). The BSTs are substantially unbalanced under corpus order, resulting in higher traversal costs; the short hash chains explain the 5–6× build-time advantage observed in the performance table. The data is from report.txt under the src folder.

- **Synthesis**

BST heights (35–38) are far above $\log_2(U)$ (≈13–15), confirming heavy imbalance and explaining the time gap from Analysis 2. Hash chains remain short (5–6), consistent with near-constant insertion costs and the observed performance.

Overall Synthesis

Across corpora, the qualitative content differences underscore that frequency lists are primarily style/genre detectors without further normalization. Quantitatively, build times align with structure dynamics: unbalanced BSTs incur depth-scaled costs, while hash tables retain short probe paths. For real-world pipelines that ingest naturally ordered text at scale, HashMap (with sensible load factor and table sizing) is the pragmatic default. If a tree is required (e.g., for ordered traversals), a self-balancing variant (AVL) would be necessary to keep height near O(log U) and narrow the performance gap.

Acknowledgements

I would like to thank Professor Harper and Professor Al Madi for providing the project template and guidance site, as well as the TAs for guiding me through the debugging process. I also appreciate that my classmates help me debug. Thanks to Oracle and W3Schools for the detailed Java documentation.

Extensions:

What

I implemented a self-balancing search tree, AVLMap<K,V>, that conforms to the provided MapSet<K,V> interface (with the same API as BSTMap/HashMap), and a tester class, AVLMapTester, to test that AVLMap works correctly. The motivation was to eliminate the severe height inflation observed in the plain BST when inserting words in natural corpus order (e.g., Reddit): BST heights reached 38 (vs. an ideal $\log_2(U) \approx 15.4$), which drives up insertion cost. An AVL tree guarantees height O(log U) by performing local rotations after inserts/deletes, so the hypothesis was: keep ordered traversal while avoiding the worst-case behavior of an unbalanced BST. More specifically, each node tracks height; after every insert/delete the tree rebalances using the standard LL/LR/RL/RR rotations. Ordering uses the same Comparator<K> approach as BSTMap, falling back to natural Comparable<K>. maxDepth() returns the AVL height so it's directly comparable to BSTMap.maxDepth().

Outcome

The AVL structure produces the same counts and top-10 lists as BST/HashMap for both corpora (see identical "top 10 words" across structures in the report.txt under the ext folder), confirming functional equivalence.

Table 5. Build Time & Depth Summary (BST vs. AVL vs. HashMap)

Corpus	U (unique)	$\log_2(U)$	Structure	Avg Build (ms)	maxDepth
Reddit 2015	43,968	15.42	BST	5,120.00	38
			AVL	7,293.80	19
			HashMap	1,350.40	6
Shakespeare	13,120	13.68	BST	105.00	35
			AVL	146.60	16
			HashMap	21.60	5

HashMap is fastest on both corpora; AVL dramatically reduces depth compared to BST (Reddit: 38 -> 19; Shakespeare: 35 -> 16), keeping height near $\log_2(U)$ (≈15–14). Self-balancing fixes the structural problem (height), but on this frequency-count workload rotations add constant-factor overhead, so AVL can be slower than a plain BST in practice; use HashMap for raw throughput, AVL when you need ordered traversal without worst-case BST blowups. The data is from report.txt under the ext folder.

Table 6. Relative Speed vs. HashMap

Corpus	BST / HashMap (× slower)	AVL / HashMap (× slower)
Reddit 2015	3.79×	5.40×
Shakespeare	4.86×	6.79×

Relative to HashMap, BST is ~3.8–4.9× slower and AVL is ~5.4–6.8× slower across the two datasets. For large text streams and hot inner loops, the hash table's near-O(1) inserts dominate; balanced trees trade speed for order guarantees and tail-risk control (predictable O(log U) ops), which matters if you need sorted keys or range traversals. The data is from report.txt under the ext folder.

In conclusion, the AVL trees are much shorter than BST, which is better, but on this workload, the constant-factor cost of many rotations dominates, so AVL builds slower than plain BST. HashMap remains fastest end-to-end due to near-O(1) inserts and short chains (maxDepth 5–6).

How to run

Run and compile Experiments.java. Experiments.main iterates over three structures: "BST", "AVL", "HashMap". You can edit the runs variable, which is set to 5 by default in Experiments.java.