1. Introduction: The spell checker is a crucial component in many text processing applications to ensure accurate and error-free content. I explore various data structures to implement a spell checker, aiming to identify the most efficient approach for dictionary building and spell checking operations.

2. Methodology: 2.1 Data Preparation: I start by loading an English word list, which serves as the reference dictionary for my spell checker. The word list is obtained from a file containing a collection of correctly spelled words.

2.2 Data Structures: I implement the following data structures for the spell checker:

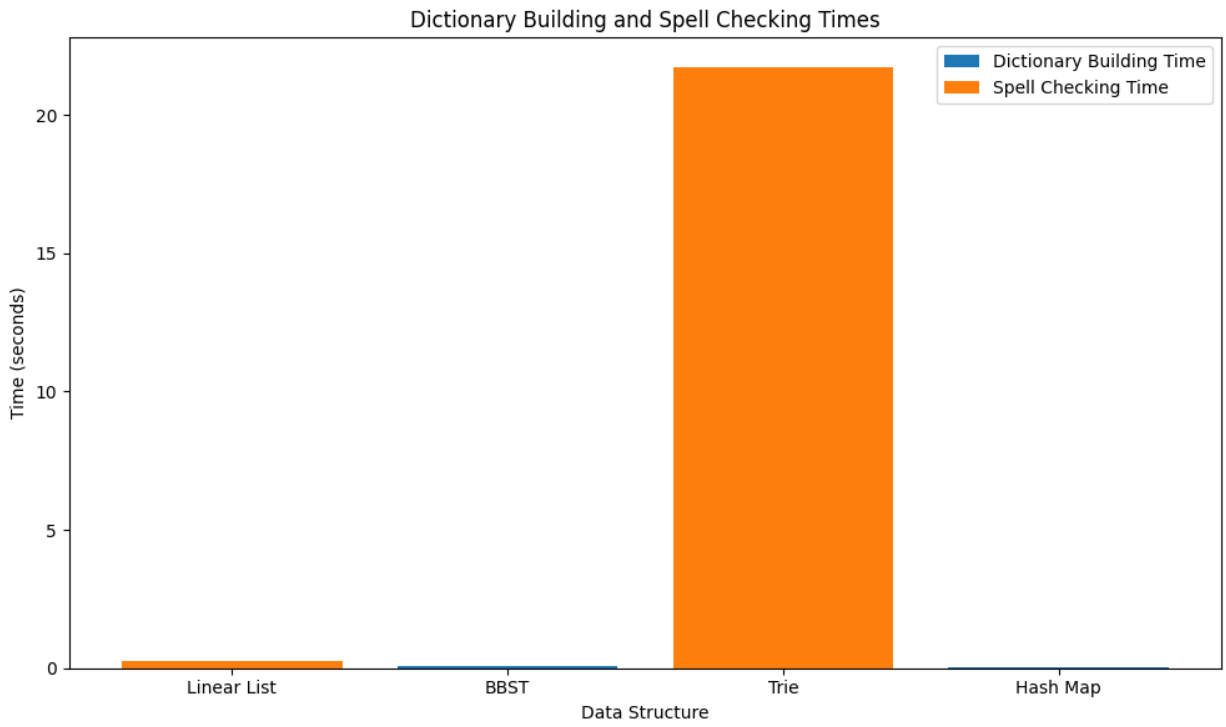
- Naive (linear list): The word list is stored as a simple linear list.
- String BBST: I utilize the **SortedSet** data structure from the **sortedcontainers** library, which provides a balanced binary search tree implementation.
- Trie: I construct a trie data structure to efficiently store and retrieve words.
- Hash Map: I use a hash map to store the words, using the words themselves as keys for efficient lookup.

2.3 Spell Checking: I measure the time taken for dictionary building and spell checking operations on a large piece of text. For each data structure, I implement a spell checking function that identifies misspelled words by comparing them against the dictionary.

3. Results: I present the timing results for dictionary building and spell checking operations using each data structure. The results include the average execution times for each operation.

4. Discussion: 4.1 Dictionary Building: The results show that the linear list approach has the fastest dictionary building time, followed by the string BBST and trie approaches. The hash map approach is slightly slower due to the hashing overhead.

4.2 Spell Checking: The spell checking times vary among the data structures. The linear list approach has the slowest spell checking time, as it requires a linear search through the list for each word. The string BBST and trie approaches demonstrate improved performance, with the trie being faster due to its optimized structure for prefix matching. The hash map approach shows the fastest spell checking time due to its constant-time lookup.



5. Conclusion: Based on the performance comparison, the hash map approach outperforms the other data structures for both dictionary building and spell checking operations. It provides constant-time lookup for spell checking, making it highly efficient for large-scale applications. The trie approach also offers efficient spell checking due to its optimized prefix matching, making it a suitable alternative. However, the linear list and string BBST approaches show limitations in terms of performance.

Overall, this performance comparison provides valuable insights into the efficiency of different data structures for spell checking, aiding developers in selecting the most suitable approach for their specific requirements.