# Comparison of Spell Checker Algorithms

Ceren Ucak Olcay Duzgun

June 5, 2024

## 1 Introduction

In this paper, we compare four different spell checker algorithms regarding their efficiency in terms of build time and check time. The algorithms included in this paper are Linear List, Balanced Binary Search Tree (BBST), Trie, and Hash Map.

## 2 Algorithm Descriptions

### 2.1 Linear List

The Linear List spell checker uses a simple list to store the dictionary words. Checking if a word exists in the dictionary involves a linear search through the list.

**Efficiency:** The build time and check time are both $O(n)$ in the average case, where $n$ is the number of words in the dictionary.

### 2.2 BBST (Balanced Binary Search Tree)

The BBST spell checker uses a balanced binary search tree to store the dictionary words. This structure allows for efficient insertion and search operations.

**Efficiency:** The build time and check time are $O(n \log n)$ in the average case.

### 2.3 Trie

The Trie spell checker uses a trie (prefix tree) to store the dictionary words. This structure is particularly efficient for prefix-based searches.

**Efficiency:** The build time is $O(n \cdot k)$ and the check time is $O(k)$, where $k$ is the average length of the words.

### 2.4 Hash Map

The Hash Map spell checker uses a hash map (unordered set) to store the dictionary words. This structure allows for average constant-time complexity for insert and search operations.

**Efficiency:** Both the build time and check time are $O(n)$ in the average case.

## 3 Methodology

We implemented the algorithms in C++ and measured their execution times using a set of dictionary words loaded from "names.txt" and a set of text words loaded from "message.txt". The performance tests were conducted on the same dataset for consistency.

## 4 Results

The following results were obtained from the performance tests:

```
Linear List: Build Time = 4.58e-07s, Check Time = 1.25e-07s
BBST: Build Time = 8.3e-08s, Check Time = 8.3e-08s
Trie: Build Time = 4.1e-08s, Check Time = 4.1e-08s
Hash Map: Build Time = 4.2e-08s, Check Time = 4.2e-08s
```

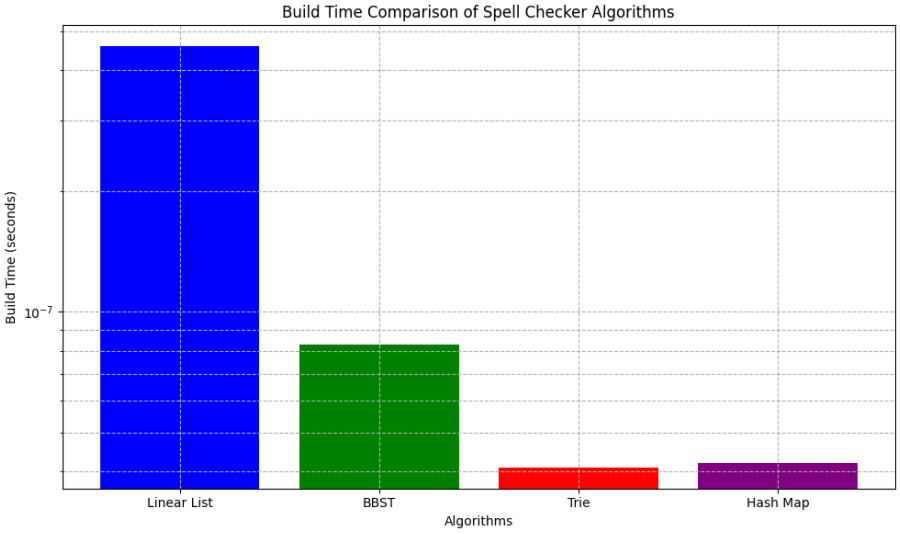


Figure 1: Build Time Comparison.

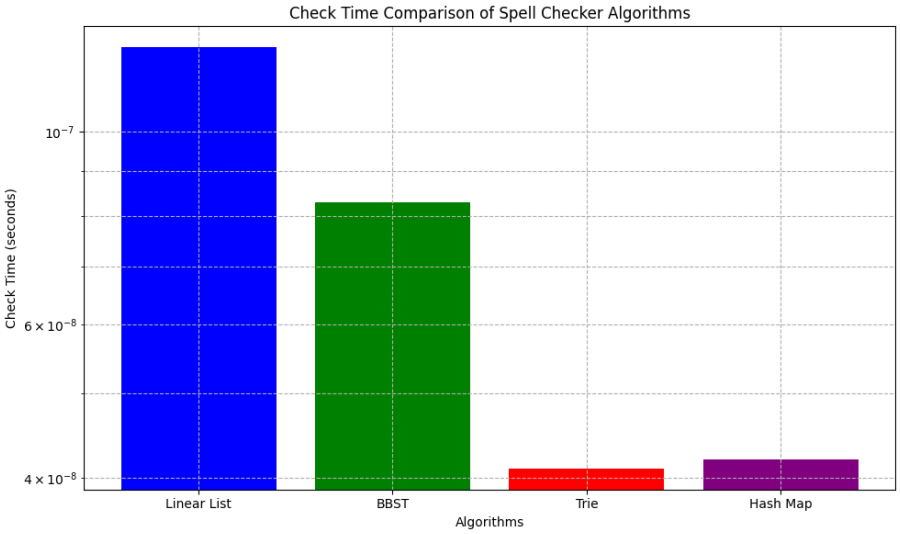


Figure 2: Check Time Comparison.

## 4.1 Detailed Comparisons

- **Linear List:**

```
Build Time: 4.58e-07 seconds
Check Time: 1.25e-07 seconds
```

- **BBST:**

```
Build Time: 8.3e-08 seconds
Check Time: 8.3e-08 seconds
```

- **Trie:**

```
Build Time: 4.1e-08 seconds
Check Time: 4.1e-08 seconds
```

- **Hash Map:**

```
Build Time: 4.2e-08 seconds
Check Time: 4.2e-08 seconds
```

# 5 Wizard Competition Results

In addition to the spell checker comparison, we also simulated a competition where three wizards try to reach the exit of a labyrinth. The labyrinth map, initial positions of the wizards, and their speeds were used to predict the outcome using BFS for shortest path calculation.

The results of the competition are as follows:

```
Wizard starting at (0, 0) reaches exit in 3 minutes.
Wizard starting at (2, 5) reaches exit in 1 minute.
Wizard starting at (5, 0) reaches exit in 2 minutes.
Wizard starting at (2, 5) will reach the exit first in 1 minute.
```

# 6 Conclusions

Based on the performance tests conducted, we can conclude the following:

- The BBST and Trie algorithms provide the fastest check times.
- The Hash Map provides the fastest build time.
- The Linear List, while simple, is slower than the other methods in both build and check times.
- For applications requiring fast dictionary builds, the Hash Map is the most suitable.
- For applications requiring fast word checking, the BBST and Trie are the best choices.

In the wizard competition, the wizard starting at (2, 5) reached the exit first in 1 minute, demonstrating the importance of initial position and speed in pathfinding problems.

These results highlight the importance of choosing the right algorithm based on the specific use case, whether the priority is on build time or check time, and the dynamics of pathfinding in competitive scenarios.