Performance Comparison of Spell Checking Data Structures

In this experiment, we compared the performance of four different data structures for spell checking: Naive List, Balanced Binary Search Tree (BBST) using Python's set, Trie, and Hash Map. We measured the time taken to load a dictionary of words and the average spell check time for each data structure.

We used a large set of test words containing various English words.

Data Structures

Four different data structures were implemented for spell checking: Naive List, BBST (set), Trie, and Hash Map.

Implementation

Each data structure was implemented and tested using Python.

Performance Metrics

We measured the time taken for dictionary loading in seconds and the average spell check time in nanoseconds.

Naive List:

Description: The Naive List approach stores the dictionary of words in a simple list structure. Spell checking involves iterating through the list to find a match.

Performance: Although simple to implement, the Naive List exhibited the slowest performance among the evaluated data structures. Its spell check time was notably high due to linear search complexity.

BBST (set):

Description: The BBST using Python's set implementation employs a balanced binary search tree structure, ensuring efficient searching and insertion operations.

Performance: BBST demonstrated rapid dictionary loading and spell checking times. Its logarithmic search complexity resulted in fast spell check operations, making it one of the top-performing data structures.

Trie:

Description: Trie is a tree-like data structure composed of nodes that represent letters of words. Words are stored by traversing down the tree, with each path representing a distinct word.

Performance: Trie exhibited moderate performance in dictionary loading but relatively slower spell check times compared to BBST. However, Trie offers advantages in memory efficiency and prefix-based search operations.

Hash Map:

Description: A Hash Map stores key-value pairs where keys are hashed to unique indices, enabling rapid lookup operations.

Performance: Hash Map showcased exceptional performance in both dictionary loading and spell checking. Its constant-time lookup complexity led to the fastest spell check times among the evaluated data structures.

A chart with blue rectangles

Description automatically generated

Observations

1. Dictionary Load Time: Naive List had the longest load time, followed by Trie, BBST (set), and Hash Map, in increasing order of efficiency.

2. Average Spell Check Time:

- BBST (set) and Hash Map demonstrated the fastest average spell check time, with Hash Map being slightly faster than BBST.

- Trie exhibited a higher average spell check time compared to BBST and Hash Map.

- Naive List had the slowest average spell check time, significantly slower than the other data structures.

Conclusion

A graph with numbers and a bar

Description automatically generated

- The choice of data structure significantly impacts spell checking performance.

- While Naive List is simple, it lacks efficiency for large-scale applications due to its linear search complexity.

- BBST (set) and Hash Map emerged as top performers, offering efficient spell checking capabilities with minimal time complexity.

- Trie, though slightly slower than BBST and Hash Map, provides advantages in memory efficiency and prefix-based search operations.

Part B: Triwizard Tournament Labyrinth Challenge

Introduction

In the Triwizard Tournament, contestants must navigate through a labyrinth to reach the exit as swiftly as possible. This report analyzes a scenario where three wizards—Wizard A, Wizard B, and Wizard C—participate in this challenge. The objective is to determine which wizard will emerge victorious based on their starting positions and speeds.

Implementation

To simulate the labyrinth challenge, a breadth-first search (BFS) algorithm was employed to discover the shortest path from each wizard's starting position to the exit. The labyrinth was represented as a 2D grid, with '1' indicating corridors and '0' indicating walls. Each wizard was defined with their name, starting position, and speed. The BFS algorithm determined the shortest path and calculated the time each wizard would require to reach the exit based on their speed.

, A maze map with a diagram

Description automatically generated with medium confidence

Results

In the provided scenario, Wizard C triumphed as the winner of the labyrinth challenge. With a calculated time of 3.33 minutes, Wizard C demonstrated the most efficient navigation strategy among the competitors. Wizard A took 10.00 minutes to reach the exit, while Wizard B required 16.00 minutes. The paths and times for each wizard were as follows:

Wizard A: Time = 10.00 minutes, Path = [(0, 0), (0, 1), (1, 1), (1, 2), (1, 3), (2, 3), (3, 3), (4, 3), (4, 2), (4, 1), (4, 0)]

Wizard B: Time = 16.00 minutes, Path = [(2, 0), (2, 1), (2, 2), (2, 3), (3, 3), (4, 3), (4, 2), (4, 1), (4, 0)]

-Wizard C: Time = 3.33 minutes, Path = [(4, 4), (4, 3), (4, 2), (4, 1), (4, 0)]

Analysis

Wizard C's victory can be attributed to several factors. Despite having a slower speed compared to Wizard A and Wizard B, Wizard C's starting position allowed for a direct path to the exit with minimal obstacles. In contrast, Wizard A and Wizard B encountered longer and more convoluted paths, resulting in increased travel times. The layout of the labyrinth and the strategic positioning of the wizards played critical roles in determining the outcome.

Conclusion

The labyrinth challenge in the Triwizard Tournament tests contestants' abilities in strategic planning and efficient navigation. Through the utilization of BFS and careful analysis of starting positions and speeds, the winner of the challenge can be accurately predicted. In this scenario, Wizard C emerged victorious by employing an effective navigation strategy, showcasing the importance of tactical decision-making in competitive wizardry.

This report offers insights into the implementation, results, and analysis of the Triwizard Tournament's labyrinth challenge, highlighting the significance of strategic planning and efficient pathfinding in competitive wizardry competitions.