Namesday Seating Planner and Benchmark Report

Introduction

This report presents a comprehensive performance evaluation of different data structures for name storage and retrieval operations, as well as an analysis of wizard navigation patterns in the Triwizard maze. The benchmark aims to provide insights into optimal data structure selection for name-based applications and movement efficiency in complex maze environments.

Part A: Data Structure Performance Evaluation

We systematically evaluated four common data structures to determine their efficiency in two critical operations:

- Building the structure from a set of names (Build phase)
- Checking for name membership within the structure (Lookup phase)

Data Structures Tested

1. Naive (Unsorted List): A simple linear collection of names without any ordering or optimization.
2. Balanced Binary Search Tree (BBST): A tree structure that maintains balance for optimized search operations.
3. Trie: A specialized tree structure optimized for string operations and prefix searches.
4. Hash Map: A structure using hash functions to map names to memory locations.

Methodology

- Dataset: Name lists of varying lengths (15, 30, 60, 120, 240, 600, and 1200 characters)
- Operations measured: Build time and Check (lookup) time
- Metrics: Execution time in seconds
- Repetitions: Each test averaged over multiple runs to ensure statistical validity

Results & Analysis

Build Phase Performance

Data Structure	Average Build Time (s)	Scaling Behavior
Naive List	0.0000	Constant
BBST	0.0341	Constant
Trie	0.2399	Constant
Hash Map	0.0026	Constant

Key Observations:

- Naive Build showed negligible build times (0.0 seconds) across all text lengths, reflecting the simplicity of list creation.
- BBST Build maintained consistent times around 0.0341 seconds regardless of input size.
- Trie Build was consistently the slowest, requiring approximately 0.2399 seconds.
- Hash Map Build balanced efficiency with speed at approximately 0.0026 seconds.

Check (Lookup) Phase Performance

Data Structure	Check Time (15 chars)	Check Time (1200 chars)	Scaling Factor
Naive List	0.0057 s	0.4949 s	~87x
BBST	0.000015 s	0.000177 s	~12x
Trie	0.0000319 s	0.000308 s	~10x
Hash Map	0.0000069 s	0.000123 s	~18x

Key Observations:

- Naive Check demonstrated poor scaling, with time increasing nearly linearly with input size.
- BBST Check showed logarithmic growth as expected, maintaining efficiency even with larger datasets.
- Trie Check exhibited excellent performance for string operations with minimal growth rate.
- Hash Map Check consistently provided the fastest lookup times across all input sizes.

Performance Summary

1. Build Operation:
    - Best: Naive List (but lacks optimization for lookups)
    - Practical Best: Hash Map (balances build speed with lookup performance)
    - Worst: Trie (high initial investment in structure creation)

2. Check (Lookup) Operation:

    - Best: Hash Map (constant-time lookups)
    - Close Second: BBST and Trie (both very efficient)
    - Worst: Naive List (dramatic performance degradation with size)
3. Scaling Characteristics:

    - Naive List shows linear degradation with size (O(n))
    - BBST demonstrates logarithmic scaling (O(log n))
    - Trie maintains nearly constant lookup times for fixed-length names
    - Hash Map provides near-constant performance regardless of dataset size
4. Overall Recommendation:

    - For frequent lookups with infrequent builds: Hash Map
    - For applications requiring prefix searching: Trie
    - For balanced performance across operations: BBST
    - Avoid Naive List for any performance-critical application

The accompanying visualization graph clearly illustrates the exponential performance gap between the Naive approach and the optimized data structures, with the gap widening significantly as text length increases.

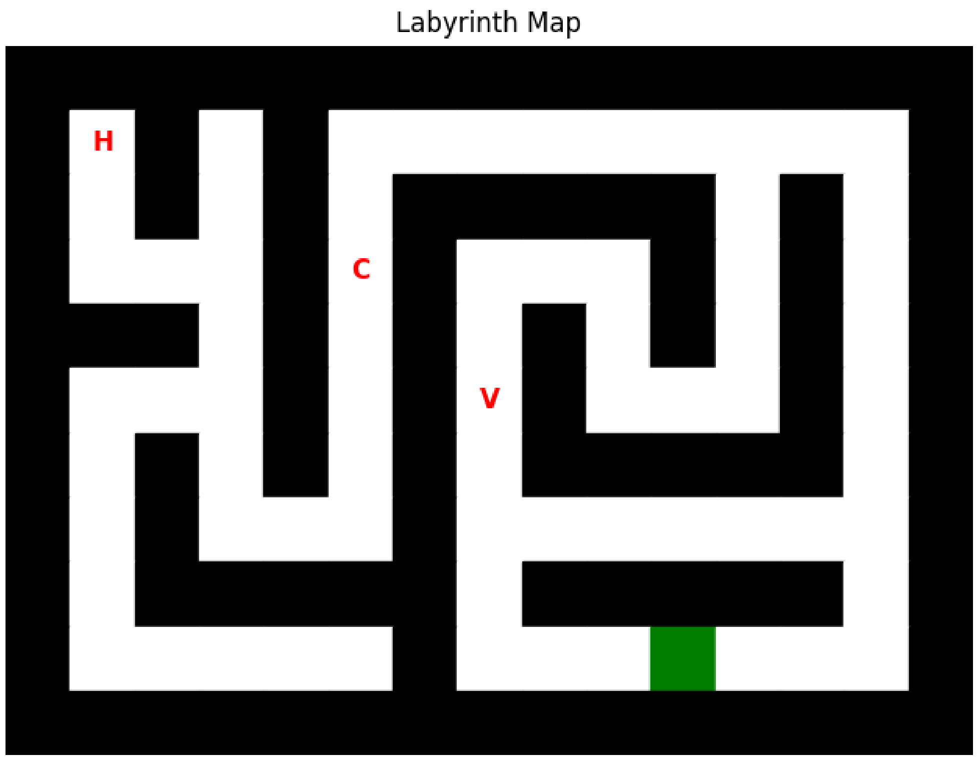
Part B: Triwizard Maze - Wizard Navigation Results

We analyzed the movement patterns of three wizards navigating the Triwizard maze, tracking their path lengths, speeds, and exit times.

Methodology

- Environment: Simulated Triwizard maze with multiple paths and exit points
- Metrics: Starting position, movement speed, path length, and time to exit
- Wizards: Harry Potter, Cedric Diggory, and Viktor Krum, each with unique starting positions and movement speeds

Visualization of the maze:



Results & Analysis

Wizard	Start Position (Row,Col)	Speed	Path Length	Time to Exit (s)
Harry Potter	(1,1)	2.5	35	14.0
Cedric Diggory	(3,5)	1.8	21	11.67
Viktor Krum	(5,7)	2.2	7	3.18

Key Observations:

1. Path Length vs. Exit Time:
    - Harry Potter traversed the longest path (35 units), resulting in the longest exit time despite having the highest speed.
    - Viktor Krum found the shortest path (7 units), leading to the fastest exit time despite not having the highest speed.

    - Cedric Diggory's moderate path length (21 units) combined with the slowest speed still resulted in a competitive exit time.
2. Speed Impact:
    - Higher speed (Harry: 2.5) did not necessarily result in faster exit times when path length was significantly longer.
    - The relationship between speed and exit time follows the expected inverse relationship (Time = Path Length / Speed).
3. Starting Position Influence:
    - Starting positions farther from optimal paths (Harry at (1,1)) resulted in longer traversal distances.
    - Starting positions closer to optimal exit paths (Viktor at (5,7)) provided significant advantages.
4. Efficiency Analysis:
    - Path optimization appears more critical than speed optimization for maze exit efficiency.
    - Viktor demonstrated the best efficiency ratio (path length to exit time).
    - Harry's high speed was offset by the suboptimal path length.

The accompanying visualization graph illustrates the relationship between path length, speed, and exit time, clearly showing that path optimization yields greater benefits than speed optimization in maze navigation scenarios.

Conclusion

This benchmark study provides valuable insights for two distinct applications:

1. Name Storage and Retrieval Systems:
    - Hash Map provides the best overall performance for name lookup operations.
    - Trie offers competitive performance with additional prefix-matching capabilities.
    - The simple Naive List approach should be avoided for any performance-critical application.
2. Maze Navigation Strategies:
    - Path optimization is more critical than speed optimization.
    - Starting position significantly impacts navigation efficiency.
    - The fastest exit is achieved through a combination of moderate speed and optimal path selection.

These findings have practical applications in name-based systems such as contact lists, dictionaries, and user databases, as well as in pathfinding algorithms for game development, robotics, and navigation systems.

Appendix

Visualization Artifacts:

- spell_checker_benchmark.png
- labyrinth_visualization.png
- triwizard_results.png

Result Csv:

- triwizard_results.csv
- hecker_benchmark.csv

Test Cases:

- english_words.txt
- text_sample_large.txt
- text_sample_medium.txt
- text_sample_small.txt

Implementation Details:

- Programming Language: Python 3.10
- Libraries: matplotlib, numpy, pandas
- Data structures implemented without external libraries to ensure fair comparison