Analysis of Algorithms

BLG 335E

Project 3 Report

Berra Mutlu
mutlub22@itu.edu.tr

Faculty of Computer and Informatics Engineering
Department of Computer Engineering
Date of submission: 20.12.2024

1. Implementation

1.1. Data Insertion

1.1.1. Binary Search Tree

The Binary Search Tree was implemented using a `Node` class that stores the publisher object, containing the `publisher`'s name and cumulative sales in three regions: North America, Europe, and Others.

The lexicographical order of publication names determines which nodes are added to the BST. The sales data is updated by appending the new numbers to the current totals if there is already a node with the same publisher name.

The insertion process involves traversing the tree recursively or iteratively until the correct position for the new node is found. The process dynamically adjusts pointers to maintain the BST structure.

1.1.2. Red Black Tree

The RBT was implemented similarly to the BST but with additional rules to maintain balance using a coloring mechanism (Red = 1, Black = 0).

New nodes are inserted into the tree, and if any violations of RBT properties occur (e.g., two consecutive red nodes), rotations and color changes are applied using `RB_insert_fixup` method to restore the balance.

Like the BST, duplicate publisher names result in cumulative sales updates.

1.1.3. Time Complexity

The BST's performance heavily depends on the order of data insertion. If the data is inserted in a sorted or nearly sorted order, the tree will become unbalanced and degrade to a linked list, resulting in O(n) time complexities for insertion, search, and deletion operations. In the best-case scenario (a balanced tree), the time complexities are O(log n) for insertion.

The Red-Black Tree guarantees balanced height, maintaining logarithmic time complexity for all critical operations. It ensures O(log n) time for all operations, making it more efficient than the BST, especially for dynamic datasets where balancing is essential.

1.1.4. Insertion Time

You can see the total time of insertion that took for both Binary Search Tree and Red Black Tree in miliseconds on the table below.

Tree Type	Insertion Time (μs)	Observation
Binary Search Tree (BST)	13993	Slightly faster due to lack of balancing operations
Red-Black Tree (RBT)	14548	Slightly slower due to balancing (rotations and color fixes)

Table 1.1: Comparison of Insertion Times for BST and RBT

1.2. Search Efficiency

The goal was to evaluate and compare the search performance of the Binary Search Tree (BST) and the Red-Black Tree (RBT) by measuring the time taken to locate specific publishers in each tree. The efficiency of these structures is crucial for datasets where frequent searches are performed.

A list of unique publisher names was extracted from the dataset. This ensured that searches were performed on actual entries within the tree.

50 random publisher names were selected for the searches using a uniform random number generator (`std::uniform_int_distribution` and `std::mt19937`).

Each tree (BST and RBT) was traversed starting from the root node to locate a specific publisher.

The search operation compared the target publisher name lexicographically with the names stored in the nodes, proceeding to the left or right subtree based on the comparison result.

The time taken for each search was measured in nanoseconds (ns) using `std::chrono::high_resolution_clock`.

For each successful search, the time was recorded and added to the total search time.

At the end of 50 searches, the average search time was calculated by dividing the total search time by the number of successful searches.

1.2.1. Comparison and Time Measurement

The balance of the tree has a considerable impact on the search time in the BST. Average search times were slower for unbalanced trees because deeper levels had to be traversed.

Despite these inefficiencies, all searches completed successfully when the target publisher existed in the tree.

The RBT demonstrated more consistent and faster search times across all queries. This can be attributed to its balanced structure, which ensures that the height of the tree remains logarithmic relative to the number of nodes.

The balancing mechanisms of the RBT (e.g., rotations during insertion) contribute to its superior search performance.

Tree Type	Average Search Time (ns)	Observation
Binary Search Tree (BST)	342.64	Slower due to potential imbalance and deeper traversal
Red-Black Tree (RBT)	261.24	Faster due to balanced structure and logarithmic height

Table 1.2: Comparison of Average Search Times for BST and RBT

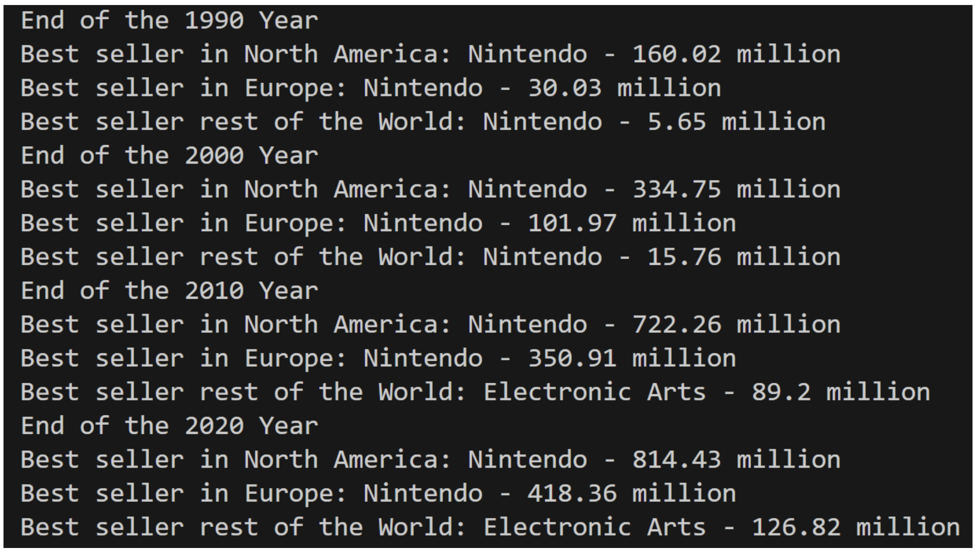
1.3. Best-Selling Publishers at the End of Each Decade

The dataset was divided into decades (e.g., 1990s, 2000s, etc.).

At the end of each decade, a pre-order traversal was performed to aggregate sales data and identify the best-selling publishers in three regions: North America, Europe, and Others.

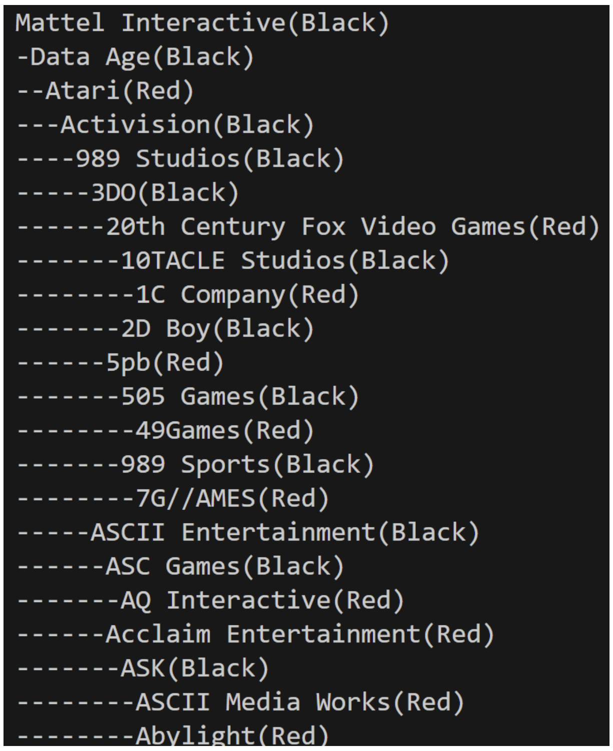
For Binary Search Tree the aggregated sales data was computed during traversal, and the best-selling publisher for each region was determined based on the highest cumulative sales values.

Similar to the BST, the RBT also identified the best-selling publishers using a pre-order traversal. However, the output for the RBT also included additional details about the tree structure, such as the depth and color of each node, providing insight into its balance.



```
End of the 1990 Year
Best seller in North America: Nintendo - 160.02 million
Best seller in Europe: Nintendo - 30.03 million
Best seller rest of the World: Nintendo - 5.65 million
End of the 2000 Year
Best seller in North America: Nintendo - 334.75 million
Best seller in Europe: Nintendo - 101.97 million
Best seller rest of the World: Nintendo - 15.76 million
End of the 2010 Year
Best seller in North America: Nintendo - 722.26 million
Best seller in Europe: Nintendo - 350.91 million
Best seller rest of the World: Electronic Arts - 89.2 million
End of the 2020 Year
Best seller in North America: Nintendo - 814.43 million
Best seller in Europe: Nintendo - 418.36 million
Best seller rest of the World: Electronic Arts - 126.82 million
```

Figure 1.1: Output for both BST and RBT



```
Mattel Interactive(Black)
-Data Age(Black)
--Atari(Red)
---Activision(Black)
----989 Studios(Black)
-----3DO(Black)
------20th Century Fox Video Games(Red)
-------10TACLE Studios(Black)
--------1C Company(Red)
-------2D Boy(Black)
------5pb(Red)
-------505 Games(Black)
--------49Games(Red)
-------989 Sports(Black)
--------7G//AMES(Red)
-----ASCII Entertainment(Black)
------ASC Games(Black)
-------AQ Interactive(Red)
------Acclaim Entertainment(Red)
-------ASK(Black)
--------ASCII Media Works(Red)
--------Abylight(Red)
```

Figure 1.2: RBT preorder output.

1.4. Final Tree Structure

The BST's structure depends heavily on the order of data insertion. For unsorted input, the tree may become unbalanced, resembling a linked list in the worst case. This results in poor performance for search operations.

The RBT ensures a balanced structure regardless of input order, maintaining a height proportional to the logarithm of the number of nodes. This balance is achieved through rotations and color adjustments during insertion.

The node depth and traversal patterns showed that the RBT had a far more balanced structure than the BST. This balance favored the RBT and had a direct effect on how well search and insertion operations performed.

1.5. Write Your Recommendation

The Red-Black Tree is the recommended structure for managing the Dataset Valley.

- **Efficiency:** The RBT consistently outperformed the BST in terms of search and insertion times, owing to its balanced structure.
- **Scalability:** The RBT is well-suited for large datasets, as its operations remain efficient regardless of input order.
- **Robustness:** The balancing mechanism of the RBT ensures consistent performance across various data distributions, making it a reliable choice for dynamic datasets.

1.6. Ordered Input Comparison

With ordered input, the BST became highly unbalanced, resulting in significantly increased depth and slower search operations.

The RBT maintained its balance even with ordered input, ensuring consistent and efficient search times.

The average search time for the RBT was consistently lower than that for the BST, highlighting the RBT's ability to handle edge cases effectively.

The assignment demonstrated the superiority of the RBT over the BST for both unsorted and sorted data scenarios. The RBT's balancing properties ensure better performance under all conditions.

You can see the insertion times for name ordered input for both trees below.

Tree Type	Insertion Time (µs)	Observation
Binary Search Tree (BST)	27549 µs	Becomes unbalanced with ordered input, leading to a long path for insertion.
Red-Black Tree (RBT)	15692 µs	Maintains balance, ensuring more efficient insertions even with ordered input.

Table 1.3: Comparison of Insertion Times for Ordered Input