Analysis of Algorithms

BLG 335E

Project 3 Report

Taha Temiz
temizt21@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

My BST implementation reads lines from the csv file, then inserts them with BSTinsert function. If the tree is empty, the BSTinsert function inserts the input node as root. If the tree is not empty, then the BSTinsert function finds the correct position for inserting and inserts the node. If the publisher already exists in the tree, the BSTinsert function updates its data. My construction function also checks decades and calls best selling publisher find function so the time taken for inserting includes the time taken for finding best sellers four times. The time taken to insert all data : 47989 microseconds.

My RBT implementation reads lines from the csv file, then inserts them with RBTinsert function as a red node. If the tree is empty, the RBTinsert function inserts the input node as root. If the tree is not empty, then the RBTinsert function finds the correct position for inserting and inserts the node. If the publisher already exists in the tree, the RBTinsert function updates its data. After the inserting operation RBTinsert function calls the fixinsert function after every insertion. The fixinsert function fixes the RBT tree cases by using rotate functions and coloring nodes according to the RBT rules. This routine keeps the balance of the tree. The construction function for RBT tree also checks decades and calls best selling publisher find function so the time taken for inserting includes the time taken for finding best sellers four times. The time taken to insert all data: 48916 microseconds.

The inserting times are flexible, but time taken to inserting BST is generally less than time taken to inserting RBT. I think it is because of the fixinsert part.

1.2. Search Efficiency

For this function, I created a vector that includes 50 random publishers. With the same search function, I searched for the same publishers in both trees.

Average time taken to search in BST : 474 nanoseconds.

Average time taken to search in RBT : 404 nanoseconds.

The search times are flexible but in general searching in RBT is faster. The RBT is more balanced than the BST, so the results are as expected.

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

My best seller finder implementation traverses the tree inorder by using a stack structure and compares sale data for the three regions of each node in the tree with the current best sellers of the tree. I checked every row of data before inserting. If the current row's year is start of a new decade (or 2020 because there is no data older than 2020), I found the best sellers of the current tree before inserting the row (except 2020. For 2020, after inserting). I also added bool variables to keep the printed decades to avoid unnecessary

prints.

```
C:\Users\TAHA\Desktop\DERSLER 24 GÜZ\ALGO\code\code\src>solution_BST_tree.exe VideoGames.csv
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
Inserting all the csv into BST tree: 47989 microseconds
Total search of 50 publisher BST: 23700 nanoseconds
Average search time BST: 474 nanoseconds
```

Figure 1.1: BST Results

```
C:\Users\TAHA\Desktop\DERSLER 24 GÜZ\ALGO\code\code\src>solution_RBT_tree.exe VideoGames.csv
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
Inserting all the csv into RBT tree: 48916 microseconds
Total search of 50 publisher RBT: 20200 nanoseconds
Average search time RBT : 404 nanoseconds
```

Figure 1.2: RBT Results

To traverse pre-order RBT, I used a stack of a pair of a node and integer.

1.4. Final Tree Structure

The BST is less balanced than the RBT. According to the input order, the balance of the BST can be changed. The time complexity for RBT is always O(logn). But for BST, it can be O(logn) with a balanced input order or O(n) with a sorted input order (the worst case). In some cases, the BST can behave as a linked list as we can see in the 6th section.

1.5. Write Your Recommendation

The RBT structure is significantly more efficient than the BST structure. The BST can be balanced but only with a balanced ordered input. Time complexity of RBT does not change according to the input order. In searching, the RBT structure is more stable than the BST structure as we can see in my measurements. According to the previous

results, the time taken to insert VideoGames to the RBT is greater than the time taken to insert VideoGames to the BST. But when the input is sorted, we can see that time taken for the operations in BST has significantly increased. So as mentioned before, the RBT structure is more effective than the BST structure.

1.6. Ordered Input Comparison

I ordered the data according to game name. Then I constructed both trees according to the game name. The performance of RBT did not change significantly. Because after every insertion the RBT keeps itself balanced with fixinsert function. But performance of BST has decreased significantly. In every insertion, the insert BST function inserted the node to the right of the greatest key. So the BST tree behaved like a linked list. So the time complexity of inserting and searching is O(n) (the worst case) for BST. The time complexity of inserting and searching for RBT remained the same (O(n)). The results can be seen below.

Figure 1.3: RBT Pre-order

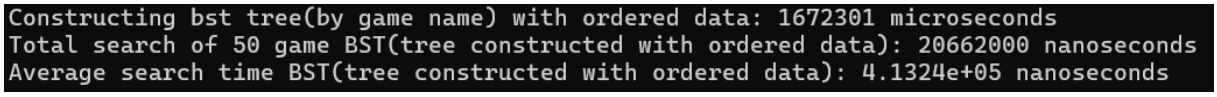

Figure 1.4: Search results for BST(ordered input)

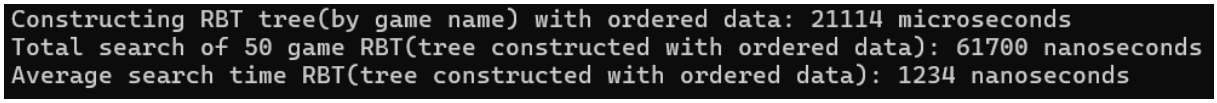

Figure 1.5: Search results for RBT(ordered input)