CSE4094
Advanced Data Structures
Homework #1

“Comparison of Red Black Trees and AVL Trees”

150114022

Oğuzhan BÖLÜKBAŞ

Binary Search Tree

In binary search trees, which are a special form of Binary Tree, the information that lies on the nodes must have a small size relation. For example, if data from integers is kept, these data have a small-to-large relationship between them.

The binary search tree requires that all data that can be accessed from the arm to the left of each node is smaller than the value of the node, and that the value of data that can be accessed from the right arm is greater than the value of that node.

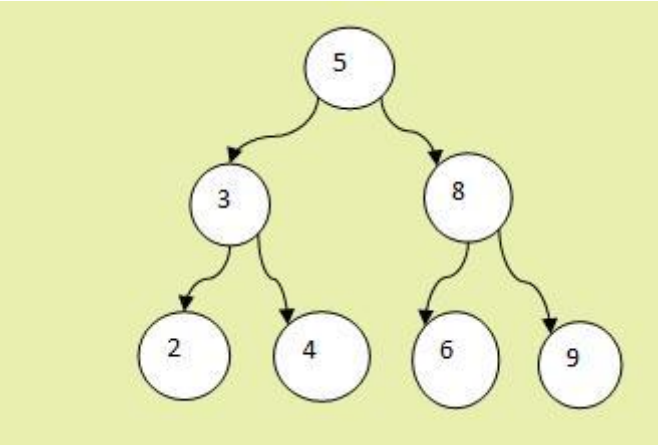


Figure 1: An example of Binary Search Tree [1]

For example, a binary search tree above is depicted. If you pay attention to this tree, all the numbers on the left of the root are smaller and all the numbers to the right are larger.

The binary search tree can be added or deleted according to this rule. However, after each operation, the structure of the binary search tree must be intact.

Problem of Binary Search Tree

Values are inserted in binary search trees with their coming order. A value which comes first before others is inserted firstly, then the second one and so on. Sometimes this order produces efficient BSTs which have height O($\log n$). Whereas, order of the keys which will be inserted can produce inefficient BSTs which have height O(n). For example, the BSTs below have same keys, but their coming order is different. Thus, we have two BSTs which have different construction. Maximum height of the first BST is 4 which is O($\log n$), on the other hand, maximum height of the first BST is 7 which is close to worst case, O(n).

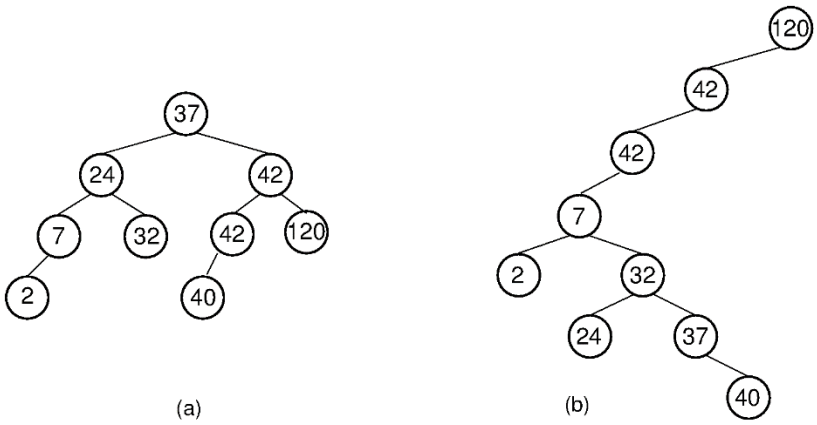


Figure 2: Two Binary Search Trees for a collection of values. Tree (a) results if values are inserted in the order 37, 24, 42, 7, 2, 40, 42, 32, 120. Tree (b) results if the same values are inserted in the order 120, 42, 42, 7, 2, 32, 37, 24, 40. [2]

This problem can be solved with balanced trees. The tree is only balanced if:

1. The left and right subtrees' heights differ by at most one, AND
2. The left subtree is balanced, AND
3. The right subtree is balanced

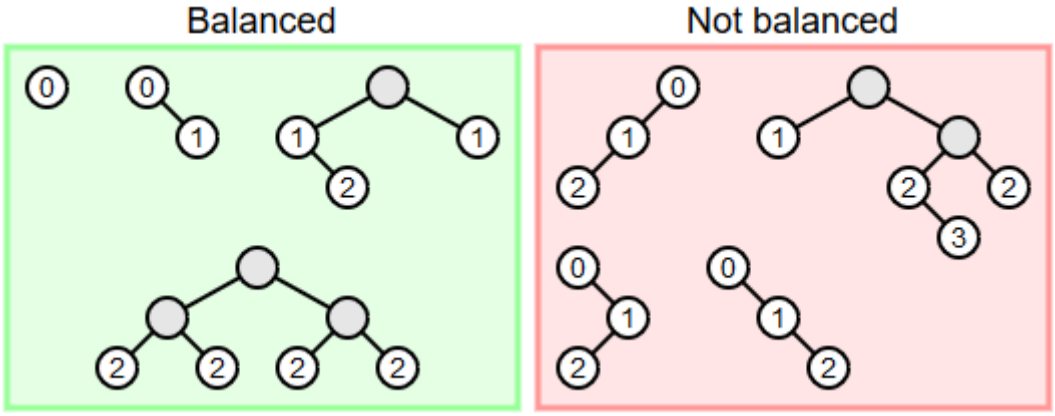


Figure 3: Balanced and not balanced trees [3]

AVL Trees and Red-Black Trees are special types of balanced trees.

AVL Tree

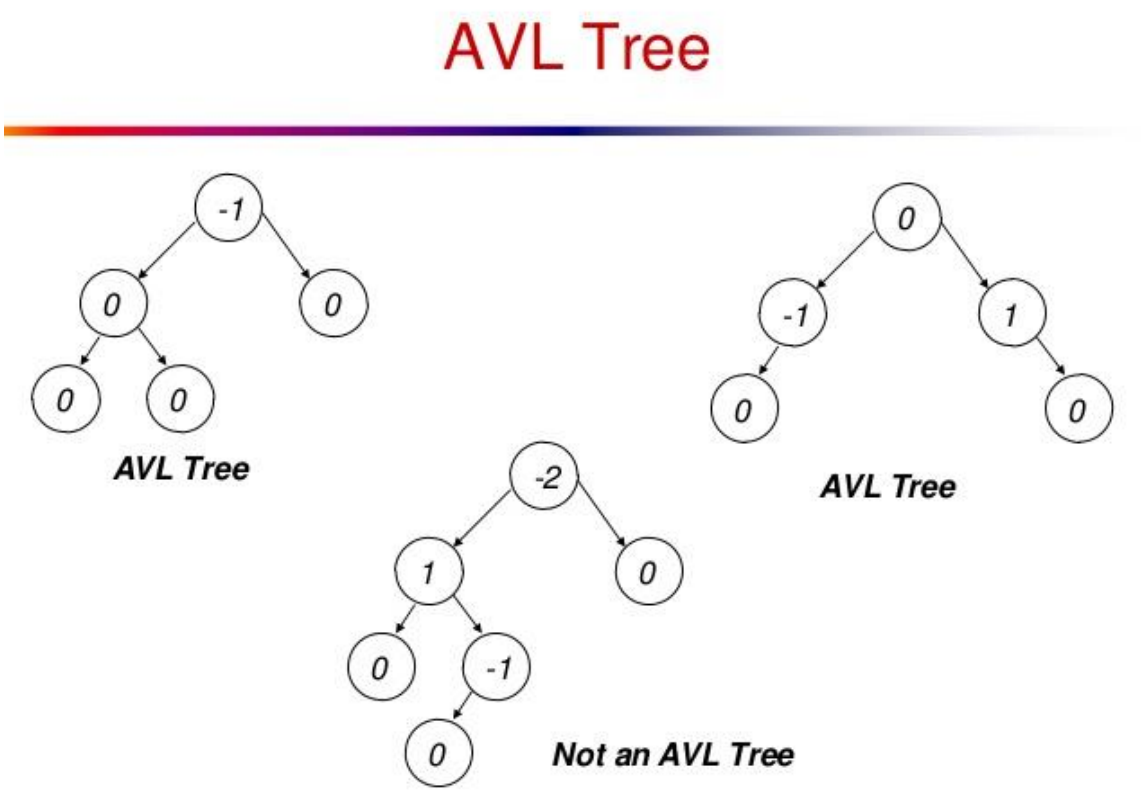
AVL Trees are consistently balanced binary search trees. Algorithm of the tree is developed by G. M. Adelson - Velsky and E.M. Landis and the name consists of the initials of these people.

The algorithm is simply equalized if the depth difference between the branches of a node is 2. If the difference is less than 2 (i.e., 1 or 0) then there is no need for a compensation operation.

Algorithm's tree navigation algorithm is the same as binary search trees. However, due to the deterioration of the tree during the addition and deletion to the tree, the depth control is added to these functions.

After adding and deleting the binary search tree in addition and deletion operations, the following balancing is performed.

```
For each node that is added or deleted in a tree:
  left  <- Measure the depth of the left arm of the node
  right <- Measure the depth of the right arm of the node
  if (left - right) > 1
    balance to left
  if (left - right) < -1
    balance to right
```



[4]

Red-Black Tree

Red-black trees, by definition, are the binary search tree and in this sense, to the left of any node is expected to stop small and to the right of the large data. A color property is also maintained for each node in the tree. So, a node can carry a red or black color feature. These properties that the nodes in the tree must carry are as follows:

1. Each node in the tree is red or black.
2. The root node is always black.
3. All children of any red node are black.
4. An equal number of black nodes exist on all paths from any node to the leaf node.

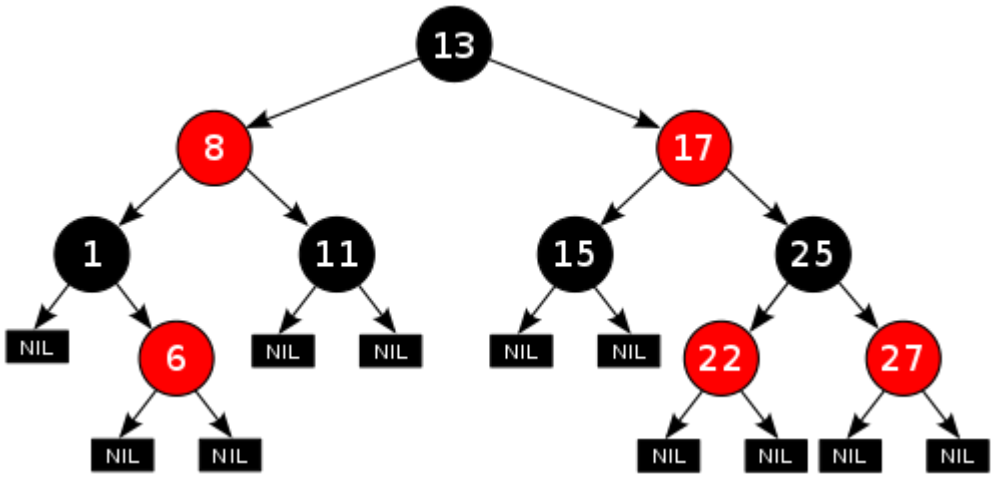
Insertion

Algorithmically, we can explain how a node is attached to a tree as below:

The insertion process starts as a regular binary search tree. During this process, we accept that the new node will be red.

There are 3 basic rules to follow during the insertion process:

1. The root node is always black.
2. In any path extending from any node to the leaves, there are an equal number of black nodes.
3. A red knot cannot have a red child.

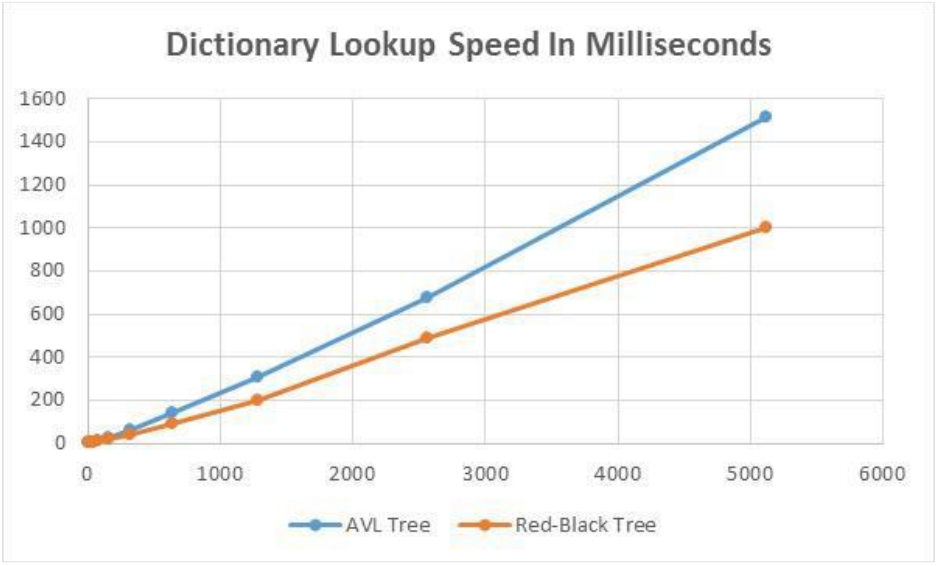


[5]

Differences

- Lookup (Execution Time)

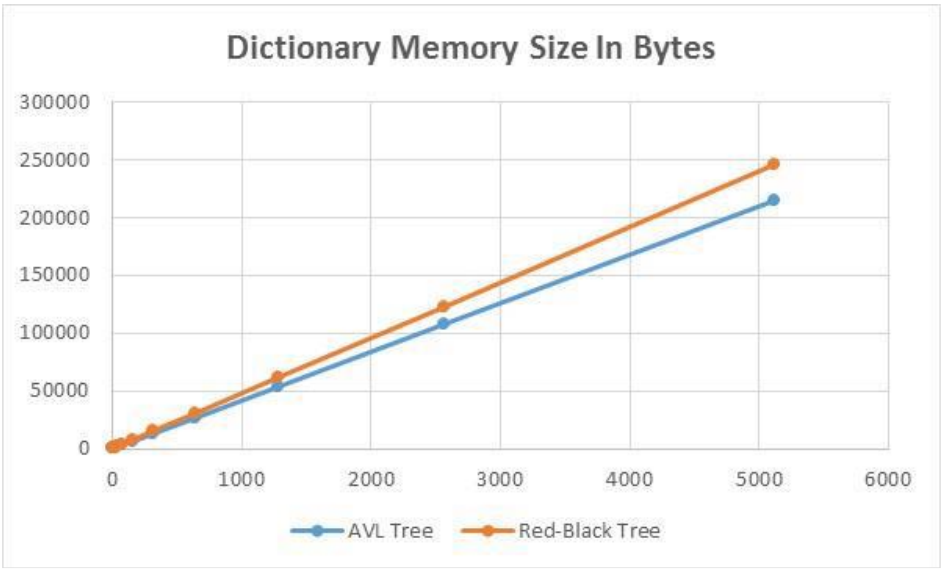
AVL trees are more rigidly balanced and hence provide faster look-ups than Red Black Trees.



[5]

- Storage

AVL trees store balance factors or heights with each node, thus requires storage for an integer per node whereas Red Black Tree requires only 1 bit of information per node.



[5]

- Insertion

Red Black Trees provide faster insertion operation than AVL trees as fewer rotations are done due to relatively relaxed balancing.

- Removal

Red Black Trees provide faster removal operation than AVL trees as fewer rotations are done due to relatively relaxed balancing.

- Usage

Red Black Trees are used in most of the language libraries like *map, multimap, multiset* in C++, *TreeSet* and *TreeMap* in Java, whereas AVL trees are used in databases where faster retrievals are required.

- Implementation

In general, the rotations are harder to implement and debug for an AVL tree than a Red-Black tree.

Conclusion

The AVL trees are more objective compare to Red-Black Trees, but they may cause more rotation during insertion and deletion. An application involves many frequent insertions and deletions, then Red Black trees should be preferred. And if the insertions and deletions are less frequent and search is more frequent operation, then AVL tree should be preferred over Red Black Tree.

References

[1] http://bilgisayarkavramlari.com/wp-content/uploads/2008/05/iaa.jpg

[2] https://opendsa-server.cs.vt.edu/ODSA/Books/CS2/html/BST.html

[3] https://www.growingwiththeweb.com/images/2015/11/14/balanced-trees.svg

[4] https://image.slidesharecdn.com/lecture11-140420071306-phpapp01/95/avl-tree-15-638.jpg?cb=1397978063

[5] https://upload.wikimedia.org/wikipedia/commons/thumb/6/66/Red-black_tree_example.svg/500px-Red-black_tree_example.svg.png

[6] A Comparative Study on AVL and Red-Black Trees Algorithm, P. Sathya et al.