Implementation of Dictionaries using AVL Tree

Kanimozhi Balaraman
Indiana State University
Terre Haute IN, USA
kbalaraman@cs.indstate.edu

November 8, 2011

Abstract

The paper is to implement Sorted Dictionaries using AVL Tree.

Dictionaries map words to their definition. So, A dictionary can be implemented by mapping two sets A and B where the input member A is mapped to the output member B. This mapping can be done by a searching algorithm. AVL tree is used as the searching algorithm in this project. The searching algorithm matches the output member B for the given input memeber A.

AVL Tree is a self balancing binary search tree. AVL trees take O(log n)time for insert, delete and search operations. The tree must be rebalanced after insert and delete operations.

This project supports insert, delete and search operations of AVL tree. In the insert operation the value 'A' is inserted for the given key 'K'. In the search operation the value 'A' is retrieved for the given key 'K'. In delete operationthe given value 'A' is removed from the dictionary. Click here for abstract

1 Statement of Problem

In the implementation of Dictionary using Binary Search tree one of the problem is that they become badly unbalanced. So a balanced Binary Search tree like AVL tree can be used to implement Dictionary. AVL trees are always balanced. In AVL trees the heights of two child subtrees of any node differs by atmost one.

In worst case Binary Search tree is linear in 'n' for insert, search and delete operations. To avoid linear-in-n worstTime for these operations a balanced binary tree called AVL tree can be used to implement the dictionary. AVL tree is logarithmic in 'n' for insert, search and delete operations in worst case.

The height of Binary Search tree is linear in 'n' in worst case. But the height of AVL tree is logarithmic in 'n' in worst case where n is the number of nodes in the tree.

2 History

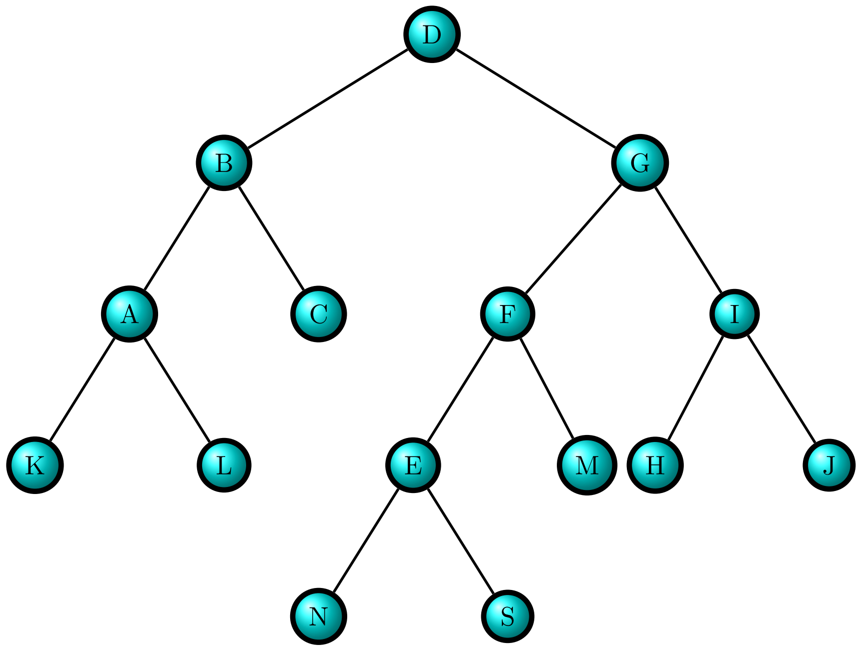
AVL trees are invented by two Russian mathematicians Adel'son-Vel'skii and Landis in 1962. AVL tree was published in the paper named "An algorithm for the organisation information". AVL tree is a balanced binary search tree which employs rotation to maintain balance. AVL tree was the first datastructure to be invented.

3 Algorithm

In this project AVL tree is used to implement Dictionary. AVL tree is Balanced Binary Search tree that is either empty or has the following properties.

1. The height of the left and right subtrees differ by atmost 1. The height of AVL tree is always logarthmic in n.
2. The left and right and left subtrees are AVL trees.

3.1 Example of an AVL Tree



3.2 Rotation

The AVL tree may become unbalanced after insert and delete operation. Rotation is the basic mechanism that rebalance the unbalanced tree.

The rotation is an adjustment to the tree, around an item, that maintains the required ordering of items. A rotation takes a constant time. Nodes that are not in the subtree of the item rotated about are unaffected by the rotation.

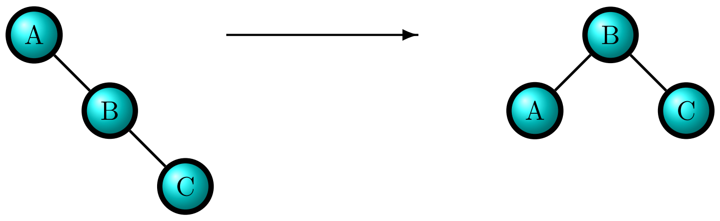
There are four kinds of rotation.

1. Single Rotation
   (a) Left Rotation
   (b) Right Rotation
2. Double Rotation
   (a) Left-Right Rotation
   (b) Right-Left Rotation

3.2.1 Left Rotation

In Left Rotation operation the tree is rotated left side to rebalance the tree. Consider a tree of three nodes of the following structure.

- A - Root node.
- B - A's Right child node.
- C - B's Right child node.



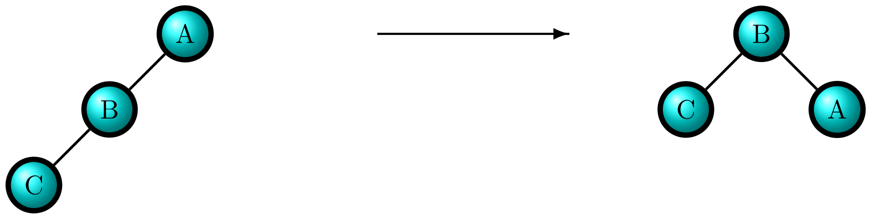
After rotation the resulting tree structure will be as following.

- B becomes the root node.
- A becomes the B's Left child node.
- C will be B's Right Child node.

3.2.2 Right Rotation

In Right Rotation operation the tree is rotated right side to rebalance the tree. A right rotation is mirror to left rotation operation. Consider a tree of three nodes of the following structure.

- A - Root node
- B - A's Left child node.
- C - B's Left child node.



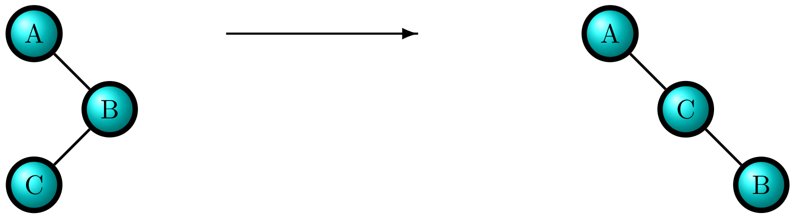
After rotation the resulting tree structure will be as following.

- B becomes the root node.
- A becomes the B's Right child node.
- C will be B's Left Child node.

3.2.3 Right-Left Rotation

In this case two rotation need to be performed inorder to balance the tree. First the tree is rotated right side and then to the left side. Consider a unbalanced AVL tree of following structure.

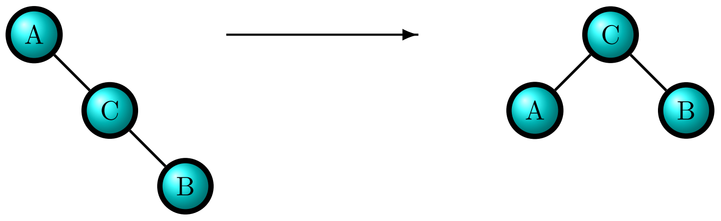
- A - Root node
- B - A's Right child node.
- C - B's Left child node.



The first rotation performed on B and C nodes in the tree.

- A - Root node
- C - A's Right child node.

- B - will become C's Right child node.



Then the tree is rotated left side inorder to obtain the balanced tree.

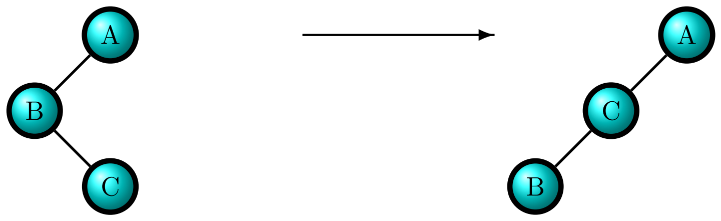
- C - Root node
- A - will become C's left child node.
- B - will become C's right child node.

3.2.4 Left-Right Rotation

A Left-Right rotation is the mirror image of Right-Left Rotation opertaion. In this case two rotation need to be performed inorder to balance the tree. The tree is first rotated on left side and then on right side.

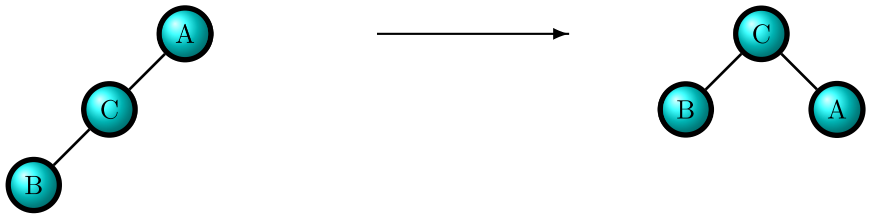
Consider a imbalanced tree of following structure to perform Left-Right rotation.

- A - Root node
- B - A's Left child node.
- C - B's Right child node.



After left rotation performed on B and C nodes, the tree structure looks like the following.

- A - Root node
- C - will become the left child node of A
- B - will become the left child node of C.



Now the tree is rotated right side inorder to obtain the balanced AVL tree structure.

- C - Root node
- B - will become the left child node of C
- A - will become the Right child node of C.

4 AVL-Operations

This project supports three basic opearations of AVL tree.

1. Insert
2. Search
3. Delete

4.1 Insert Operation

A new node can be inserted if the node value is not already present in the tree. After the insertion of each node the tree should be checked for balance. The only nodes that need to be checked are the ones along the insertion path of the newly inserted node. Once the tree is found to be unbalanced then rebalance it using the appropriate rotation operation.

4.2 Search Operation

The Search operation can be performed with the key value that need to be searched in the given AVL tree. The search operation returns a node from a tree if the node value matches with the key value. If the key value does not match with any node value in the tree no value is returned.

4.3 Delete Operation

Deletion is more complicated than insertion. The deletion of a node from a tree may decrease the height of the tree which may lead to unbalanced tree structure. So appropriate rotation operation is performed to rebalance the tree.

5 Time Complexity

AVL tree is logarithmic in 'n' for insert, search and delete operations in both average and worst case.

References

[1] Allen B. Tucker *Computer science handbook* Chapman and Hall/CRC, USA 2004

[2] Dinesh P. Mehta, Sartaj Sahni *Handbook of data structures and applications* Chapman and Hall/CRC, USA 2005

[3] Hermann A. Maurer *New results and new trends in computer science* Graz, Austria 1991

[4] Frank Dehne, Marina Gavrilova, Jorg-Rudiger Sack, Csaba D.Toth(Eds.) *Algorithms and Data Structures:11th International Symposium* WADS 2009, Banff, Canada 2009