CS575: Final Project Report

`Project Title: Red-Black Trees, AVL Trees and Skip Lists

Team Member(s): Garvita More, Pranjal Parekh

I. Problem

In this paper we have explored the implementation of Dictionary using Red black trees, AVL trees and Skip List for the primary functions. The goal of this project was to perform comparisons between these data structures and determine which of these gives the best results in terms of processing time and also taking into consideration the difficulty of implementation.

II. Algorithms

In all self-balancing binary trees are trying to achieve the same thing: when inserting or deleting nodes, perform a rebalancing using tree rotations , making sure this rebalancing also only takes logarithmic time, and that the resulting balance guarantees logarithmic search time .In this project we have used three algorithms that are Red Black Trees, AVL trees, Skip List.

A. Red-Black Trees

A red-black tree is a finite binary tree whose inner nodes are associated with keys. Keys are elements of a totally ordered set. A red-black tree is a binary search tree with one extra bit of storage per node: its color, which can be either RED or BLACK. Red-Black trees require only 1 bit instead of 2 per node for their state and provide faster insertion by doing less balancing work and enforcing a less strict balancing requirement. Red-Black trees have nodes with different colors which are used to keep the tree balanced [1]. The colors of the nodes follow the following rules:

1. Every node is either red or black.
2. The root is black.
3. Every leaf (NIL) is black.
4. If a node is red, then both its children are black.
5. For each node, all simple paths from the node to descendant leaves contain the same number of black nodes.

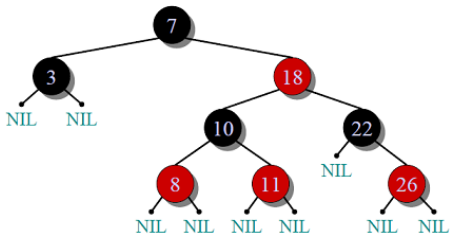

Figure 1: Example of Red-Black Tree

Time Complexity of Red-Black Trees for Insertion/Deletion:

As a red-black tree is balanced, the insert/delete operation takes O(log N) which is equal to the height of the tree. The second step is to color the new node red. This step is O(1) since it just requires setting the value of one node's color field. In the third step, we restore any violated red-black properties. Changing the colors of nodes during recoloring is O(1). However, we might then need to handle a double-red situation further up the path from the added node to the root. In the worst-case, we end up fixing a double-red situation along the entire path from the added node to the root. So, in the worst-case, the recoloring that is done during insert is O(logN) (= time for one recoloring * max number of recoloring done = O(1) * O(log N)).Thus, the third step (restoration of red-black properties) is O(log N) and the total time for insert is O(log N).

Time complexity of Red-Black Trees for search:

The complexity of the search operation is equal to the height of the tree. Different varieties of binary search trees differ in what guarantees on height of the tree they give, and in how exactly they maintain these guarantees. Red-black trees give you guaranteed O(log N) height, which is why search is **O(log N).**

B. AVL Trees

It is a self-height balancing binary search tree data structure. Each node has a balance factor, which is the height of its right subtree minus the height of its left subtree. A node with a balance factor of -1,0, or 1 is considered balanced. Nodes with different balance factors are considered unbalanced, and after different rotations performed on the tree makes it balanced. Search, insertion, and deletion all take O(log n) time in both the average and worst cases, where n is the number of nodes in the tree prior to the operation[2]. Insertions and deletions may require the tree to be rebalanced by one or more tree rotations.

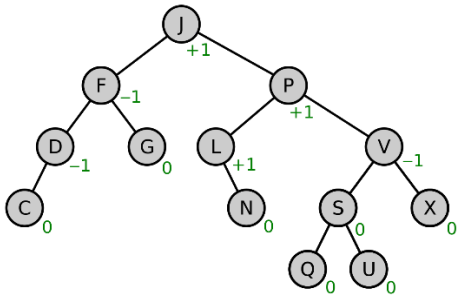

Figure 2: Example of AVL Tree

Time Complexity for Insertion and Deletion in AVL tree:
An AVL tree is balanced, so its height is O(log N) where n is the number of nodes. The rotation routines are all themselves O(1), so they don't significantly impact the insert operation or delete operation complexity, which is still O(k) where k is the height of the tree. But as noted before, this height is O(log N), so insertion or deletion into an AVL tree has a worst case O(log N).

C. Skip Lists

A skip list [3] consists of parallel sorted linked lists. A skip list is built in layers. The bottom layer is an ordinary ordered linked list. Each higher layer acts as an express lane for the lists below, where an element in layer appears in layer i+1 with some fixed probability p. A search for a target element begins at the head element in the top list, and proceeds horizontally until the current element is greater than or equal to the target. If the current element is equal to the target, it has been found. If the current element is greater than the target, or the search reaches the end of the linked list, the procedure is repeated after returning to the previous element and dropping down vertically to the next lower list.

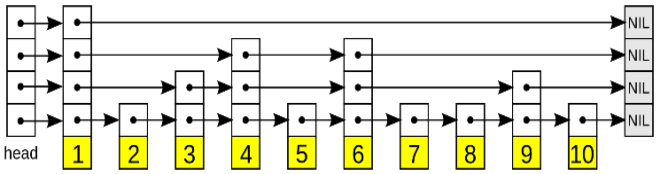

Figure 3: Example of Skip List

Time Complexity of search, insert and delete in Skip List:

The search time in a skip list is proportional to the number of drop-down steps, plus the number of scan forward steps. He drop-down steps are bounded by the height of the skip list and thus are O (log N) with high probability. To analyze the scan-forward steps, we use yet another probabilistic fact that is the expected number of coin tosses required in order to get tails is 2. When we scan forward in a list, the destination key does not belong to a higher list. A scan-forward forward step is associated with a former coin toss that gave tails. In each list the expected number of scan forward steps is 2 (Will visit at most 2 nodes per level). Thus, the expected number of scan forward steps is O (log N). O(log N) levels because you cut the no of items in half at each level Will visit at most 2 nodes per level. Therefore, total expected time is O(log N).

III. Software Design and Implementation

A. Software Design

We first started implementing our first algorithm, Red Black Trees and then moved on to its variation AVL followed by implementation of Skip List.

B. Implementation and Tools Used

The majority of our implementation for our algorithms is adapted from the book 'Introduction to Algorithms'[1]. However, due to the specified requirements of the dictionary, our adaptation of the red-black tree has ended up with various modifications and updates that differ from the CLRS description. We have used JAVA as our primary programming language.

C. Performance Evaluation

Red Black and AVL trees are alternatives to skip lists and can be used for the same type of problems. Both types of trees have performance bounds O(log N) of the same order. The comparison and contrast between these structures and skip lists are factors such as: difficulty of implementing the proposed algorithms and types of performance bounds.

Difficult Implementation [4]: For a majority of applications, the implementation of skip lists is much easier to do as compared to balanced tree algorithms. The primary reason for more difficulty in implementing balanced tree algorithms is the high demand of keeping balance. Keeping a tree balanced is a slow operation that consumes plenty of resources and must be done carefully. Red-black trees make less structural changes to balance themselves than AVL trees, which could make them potentially faster for insert/delete. For lookup intensive applications AVL trees are more efficient that Red black trees as AVL trees are more balanced than Red black trees.

Performance Bounds [5]: Balanced trees have worst-case time bounds and skip lists have probabilistic time bounds. For self-adjusting trees, it takes O(n) time, but the time bound always holds over a long sequence of operations. For skip lists, any operation or sequence of operations can take longer than expected, although the probability of operations taking much longer than expected is inconsequential. For some real-time productions, it's assured that an operation will complete within a certain time bound. For such applications, self-adjusting trees may be undesirable, since they run longer than expected. For example, an individual search can take O (n) time instead of O (log n) time.

ATTACHMENTS

https://github.com/gmore3/RedBlack_AVL_SkipList

REFERENCES

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein, Introduction to Algorithms, Third Edition, The MIT Press, 2009.

[2] Swift Data Structure and Algorithms by Mario Eguiluz Alebicto, Erik Azar.

[3] William Pugh. Skip lists: A probabilistic alternative to balanced trees. *Communications of the ACM*, 33(6):668–676, 1990.

[4] Jeff Edmonds, How to think about algorithms, Cambridge University Press,2008, USA

[5] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. Data Structures and Algorithms. Addison-Wesley, 1983.