Advanced Trees

Jakob Wyatt
19477143

October 18, 2019

1 Height Comparison

	Randomized (Size 9)	**Increasing (Size 8)**	**Decreasing (Size 11)**
Binary Search Tree	4	7	10
2-3-4 Tree	1	1	2
B-Tree (Degree 5)	1	1	1
Red-Black Tree	3	3	4

The binary search tree becomes degenerate for the increasing and decreasing datasets. The other trees avoid this, as they are self balancing. The 2-3-4 trees and B-Tree will often have a lower height than the red-black tree, as they contain more keys per node.

2 Complexity Analysis

All self balancing trees are given as average-case complexity.

	Insert	**Find**	**Delete**
Binary Search Tree (Average Case)	$O(\log_2 n)$	$O(\log_2 n)$	$O(\log_2 n)$
Binary Search Tree (Worst Case)	$O(n)$	$O(n)$	$O(n)$
2-3-4 Tree	$O(\log_4 n)$	$O(\log_4 n)$	$O(\log_4 n)$
B-Tree (Degree d)	$O(d \log_d n)$	$O(d \log_d n)$	$O(d \log_d n)$
Red-Black Tree	$O(\log_2 n)$	$O(\log_2 n)$	$O(\log_2 n)$

The worst case time complexities were not included for the self-balancing binary trees, as they do not degrade as badly as the binary search tree does. The time complexity of the 2-3-4 tree initially looks better than the red-black tree. However, due to the large constant factors contained in having 3 keys per node, it can often end up being slower in implementation. The self balancing trees will have much better performance than the BST on degenerate datasets, as the automatic balancing allows them to retain performance on the order of $O(\log n)$.

3 Difficulty of Implementation

I believe that the easiest tree to implement would be the 2-3-4 tree, as it has a fixed size and an easily understandable node splitting strategy. Next would be the B-Tree, as it is a more generalized version of the 2-3-4 tree, however still has an easily understandable splitting strategy / algorithm. I believe that the most difficult tree to implement would be the red-black tree, as the insertion algorithm has many special cases to consider, which adds more room for error.

4 In-Order Traversal

An in order traversal on a red-black tree is the same as the traversal of a normal binary search tree. An in order traversal of a B-tree could be implemented using recursion, using the example code below.

```
def inOrderBTree(node):
    # len(node.key) + 1 == len(node.children)
    for n, c in zip(node.key, node.children[:-1]):
        inOrderBTree(c)
        process(n)
    inOrderBTree(node.children[-1])
```

An in order traversal on a 2-3-4 tree is a special case of the traversal of a B-tree of order 4.