McMaster University

Department of Computing and Science

COMPSCI 2XC3
Lab-6-7 Report

Suhaas Parcha (Mac-ID : Parchas)
Yu Chang (Mac-ID : changy72)
Yuvraj Singh Sandhu (Mac-ID : sandhy1)
Group 20-L01

Professor: Dr. Vincent Maccio

Focuses on implementations of advanced data structures and also on implementing self-balancing binary trees. Also Empirically analyse a formally defined tree structure to try and show its height it O(logn) where n is the number of nodes.

March 19, 2023

Contents

List of plots

Abstract

The lab report focuses on implementing advanced data-structures. That's, the first part focuses on implementing self-balancing binary trees: "a complete implementation of Red Black Trees (RBTs)" and an analysis of how much better if at all RBTs are over simple BSTs. And second half of the lab report we empirically analyse a formally defined tree structure to try and show its height it O(logn) where n is the number of nodes.

Part - 1

0.1 Experiment 1

Description:

Trying to compare the use cases of both BST's and RBT's. We generate lists of various length, insert each of the values into both a BST and a RBT. Later we take the sum of the length of both after x number of iterations and compare them.

Procedure:

A random list of length 10000 was generated with the maximum value of each element in the list was 10000. The runs was set to 1000. These values are then inputted into the list, then the height of each tree is calculated and then divided. This results in which tree is bigger.

Later the same test was performed with the length of 10000 and the maximum value of the each element of the list was set to 1000, with the runs as 1000 Later the same test was performed with the length of 10000 and the maximum value of the each element of the list was set to 100, with the runs as 1000

Observation:

In the first experiment where the length of the list was **10000** and the maximum values of the list is set to **10000**.
The BST was on average larger than the RBT

In the second experiment we run a similar test where the length of the list is **10000** and the maximum values of the list is set to **1000**.
We observe that the RBT is larger than the BST

In the last experiment we run a similar test where the length of the list is **10000** and the maximum values of the list is set to **100**.
We observe that the RBT is larger than the BST on average.

Conclusion :

We can conclude that RBT is not always smaller than a BST , in our last 2 experiments we find that a RBT's height is more than that of a BST is because, **the lenght of the list is 10000 while the max value of the list is much smaller, due to which, there are a lot of duplicate values and the list is not sorted which causes the RBT to struggle while the BST is much better**.
So we can conclude that **BST is smaller than an RBT when there are duplicate values and the list is not sorted**. But, an RBT is almost always smaller than the BST when the **lenght of the list is smaller or equal to max value allowed per element of the list**

0.2 Experiment 2

Description:

Unlike Experiment1, we use sorted lists instead of normal list, while we increment the amount of swaps which makes our lists more unsorted over time. Later, we input the elements of this list into the trees and then calculate if RBTs are more desirable than BSTs.

For this experiment we take 100 runs, where the max number of swaps is 100 and size of the list is 100 and the maximum value is 100. If we graph the height of both trees and the number of swaps performed we get the following graph.

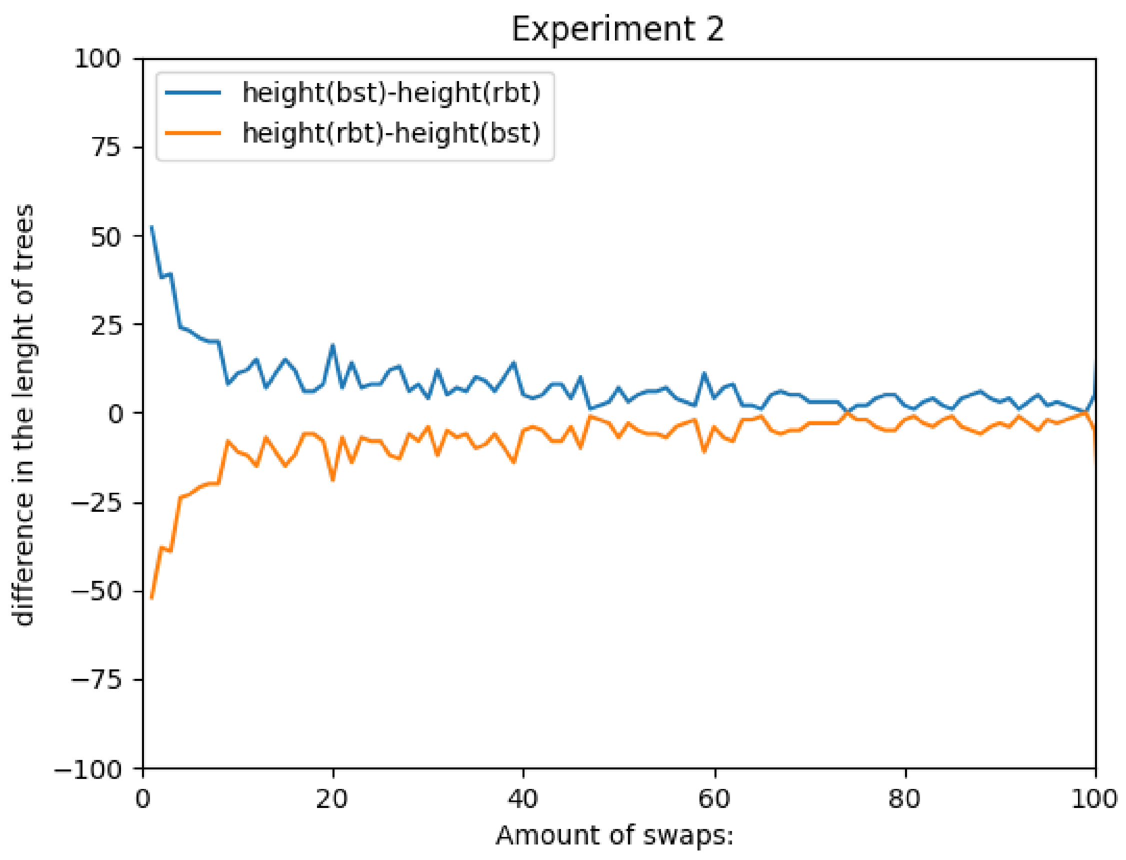

Figure 1: Difference in tree height vs swaps

Conclusion:

We can conclude that, as the number of swaps increase, from the graph, we can observe that the difference in both their heights approaches 0, this indicates that the heights become more similar and the height of the tree increases. Moreover, when the list is near sorted (0 swaps), we can see the value of rbt-bst is -50, this indicates that the size of rbt is much smaller when the list is near sorted. **Finally, we can conclude that as the list becomes more unsorted the size of the Red black tree increases.**

Part - 2

0.3 XC3 Tree Implementation and analysis

Description:

The goal of this part is implementing the formal structure: XC3 Tree and discuss the various properties of it.

Experiment3:
When I generated the XC3 Tree with the degree from 0 to 25, I got the number of heights: 1 2 2 3 3 4 4 5 5 6 6... 13 13 14. There's a pattern with the height that is: the height increases by one for every two increments in the degree.
This is due to the tree structure:
1. If the degree is 0, there are no children, and the height is 1.

2. For a node with a degree greater than 0, it has children with degrees from (i-2) to 0, where i is the degree of the parent node. 3. Each child of the root node of an XC3-Tree is also an XC3-Tree.

As the degree increases, the XC3 Tree will have more children due to the tree structure properties. When the degree of root increases by two, the number of height of the tree (levels of children) also increases by two (only if the degree is greater than 0).If the degree is 0 then adding one degree would make the height change from 1 to 2. For example: A tree with degree 1 has height "2" and if we increase the degree by 2, the height becomes 3. The XC3 tree is expanding mainly horizontally from the root and having 1 depth down every two degrees.

Experiment 4:

Number of nodes pattern

Yes I have definitely seen this pattern before, this is very similar to the Fibonacci sequence. From the third number, the number is the sum of the previous two numbers.

Explanations on why my claim holds:

1. Obviously from the output everyone can see that it holds.

2. Because in our data structure XC3_Tree, each node has childrens and each child is a distinct XC3_tree with degree i-2 (i is the ith degree of its parent). If we want to make it more clear, we can list like below:

NOTE: Only degree $>$ 2 has the power to give birth to a node!

Degree(0) = 1 parent Node = 1 node
Degree(1) = 1 parent Node + 1 child (degree 0) = 2 node
Degree(2) = 1 parent Node + 1 child(degree 0) + 1 child(degree 1) = 3 node
Degree(3) = 1 parent Node + 1 child(degree 0) + 1 child(degree 1) + 1 child(degree 2 $->$ can produce children) + 1 child-child(degree0) = 5 nodes
. . .

As we keep generating XC3_Tree with higher degrees, the pattern continues showing up. Every time we add a child to a node, we are including a sub-tree of lower degree. That means the number of nodes in i-th degree is equal to the sum of the number of nodes in two XC3_Trees with degree i -2 and i-3, until it gets to 0. It's a recursive structure and has the property I just explained. Each node has children with decreasing degrees and the number of nodes is the sum of the previous two degrees!

Argument on height bound

Question: Why is the height of an XC3-Tree with n nodes is O(log(n)?

Solution:

Basically, if we can prove that the height of an XC3_Tree with n nodes is O(log_a (n)) for any constant a, then it is also true that the height is O(log(n)) because the base of the logarithm does not affect the complexity. We will also relate this to the golden ratio and other observations. By proving the height is O(log(n)), we consider from two perspectives:

1. The relationship between the number of nodes and the height in the tree(as we observed from the previous experiment:

As we have seen from the previous discussion, the number of nodes in the XC3 Tree performs like a Fibonacci pattern due to the recursive structure. From observing the height we got, we can get that the height increases at a comparably very slow rate than the number of nodes. Since each time we are actually adding a subtree with a smaller degree and thus the height increases at a very low rate than the number of nodes.

2. Relationship with golden ratio.

In Mathematics, two quantities are in the golden ratio if their ratio is the same as the ratio of their sum to the largest of the two quantities. [Wikipedia]
The golden ratio (φ) is approximately 1.618 and is known to be connected to the Fibonacci sequence. In fact, the ratio between consecutive Fibonacci numbers converges to φ as the sequence progresses. Since the number of nodes in the XC3 Tree follows a Fibonacci like pattern, it suggests that the height of the tree is proportional to the logarithm of the number of nodes with some base related to the golden ratio.

Due to how bases are changed in logarithms, to show that f(n) = O(log(n)), it is enough to show that f(n) = O(log_a(n)) for any constant a. In this case, the constant a is related to the golden ratio, so if we can show that the height of the tree is O(log_φ(n)), it will also hold that the height is O(log(n))!

Let's assume that the number of nodes in a random XC3 Tree of height h is Node(h) where Node(h) grows at a rate related to the golden ratio. Then, we can write:

$$N(h) \approx \varphi^h$$

To show that the height of an XC3 Tree with n nodes uses O(log(n)), it is enough to show that the height is O(log_a(n)) for any constant a. In our case, the constant a is related to the golden ratio. Due to how bases are changed in logarithms, changing the base from to any other constant "a" only introduces a constant factor. Big O notation is concerned with the growth rate and not constant factors, so both log_a(n) and log_φ(n) represent the same complexity class of O(log(n)).

Therefore, as we have shown that the height h of an XC3-Tree with N(h) nodes is approximately equal to log_φ(N(h)), we can conclude that the height of an XC3-Tree with n nodes is O(log(n)).

Appendix

0.4 How to navigate the code:

Part1: RBT:
The red-black tree implementation is in the "rbt.py" file. There are some test cases commented out in the file.
Part1: Experiment 1 and Experiment2:
Open the file "exp1and2.py" and use the functions experiment1(runs) for experiment 1 and experiment2(runs) for experiement2.
Part2: XC3 Tree:
Open the file name "XC3Tree.py", The implementation of XC3 Tree is implemented in the "XC3_tree". There're also exp3() and exp4() function there to show the increments of heights and the number of nodes.

End of Lab Report

Group 20, L01.