PROPOSAL
to
Analyzing and Comparing Performances of Self-Balanced Binary Search Trees

Kadir Emre Oto - 150140032
İdil Uğurnal - 150150017
Çağla Kaya – 150150117

Contents

1. Introduction

The following is a proposal to discuss our solution for improvement to the current caching system used by the company, the mentioned solution contains an experimental analysis of self-balanced binary search trees. Currently, map structure in Standard Template Library (STL), which is implemented as a red-black tree, is used for the clients' IP caching system, however, the current system is required to perform look up intensive tasks that require less alteration in storage and more search operations, therefore, the current solution is inadequate. The proposal will mention the areas that specific instances of self-balanced binary search trees are effectively used and give information about the type of trees that have this property. Our purpose is to demonstrate which self-balanced binary search tree, among AVL trees, red-black trees and splay trees should be preferred in specific cases that require storing and accessing data efficiently.

2. Background

Binary Search Trees are mainly utilized in database systems, cache systems, 3D video games, operating system kernels and many similar areas that require keeping and accessing data. There are two types of these trees which are naïve and self-balanced binary search trees. The first type, naïve approach of binary search trees, are not commonly preferred in the industry due to possibility of having high time complexities. Therefore, we will be discussing the self-balanced approaches of binary search trees. Main examples of such trees are; AVL trees, red-black trees, splay trees, treap, b-trees etc. However, due to increased usage in the engineering sector and their popularity, AVL trees and red-black trees and splay trees are candidates for this research [1].

We will use python to create datasets for conducting experiments and measure the performances of AVL and red-black trees.

2.1 AVL Trees

One of binary search trees is AVL tree (named as initials of Soviet mathematicians Adelson-Velsky and Landis) which is height balanced [2]. For AVL trees, the difference of the left and right subtrees' height of any node cannot be bigger than 1. So, every node of AVL trees includes height information. When inserting to AVL trees, firstly, new node is added according to binary search tree order and then tree rebalances itself with rotations [3].

2.2 Red-Black Trees

Red-black tree is a data structure and is among the type of self-balanced binary search trees. Every node that belongs to the red-black tree also stores a color information. Only if the following clauses are satisfied by a binary search tree, it is to be considered as a red-black tree [4].

1. Every node is either red or black.
2. The root is black.
3. Every leaf is black.
4. If a node is red, then both its children are black.
5. For every single node, all paths from the node to the leaves have the same number of black nodes.

2.3 Splay Trees

Splay tree is one of the self-balancing binary search trees which aims faster lookup operation by reconstructing the tree after each access, and the accessed element is moved closer to the root at reconstructing step which is called *splaying*. There are three cases in order to relocate the accessed item: zig, zig-zig, and zig-zag. These steps are applied until the root of the tree is the accessed element. [5]

3. Proposal Summary

Python programming language will be used to create datasets and getting performance outcomes for respective datasets. The datasets will contain sorted, unsorted and random data instances. We will demonstrate experimental outcomes on tables and graphs.
The experiments will include addition, deletion and search function experiments on different datasets for AVL trees, red-black trees and splay trees.
After conducting this experiment, we will see which data structure among AVL, red-black and splay trees will be more useful for current cache system task.

4. Outcome

The outcomes of this experiment will include running times of some specific tasks performed by the respective data structure type (red-black tree, AVL tree and splay tree). The functions tested will be insertion, deletion and lookup functions. The results will be presented on tables.
After, the most appropriate data structure to be used given the needs of the company, will be informed.

5. Value

Determining the most suitable data structure for the certain operations and different datasets, is crucial to optimize the performance of the code. The wrong choice of structure can cause higher runtime cost. Therefore, it is important to reconstruct the caching system using right self-balanced binary search tree to store and access data in most efficient way.

6. Methods

First of all, AVL trees, red-black trees and splay trees will be compared via certain experimental functions such as insertion, deletion and lookups. This will be performed with different type of data sets such as ordered, unordered and random datasets. After, performance based on each functionality will be demonstrated on tables and charts.

Python and C++ programming languages are going to be used for implementation. Python libraries will also be utilized throughout the experiment.

7. Schedule and Cost

There are three main stages that the project is separated:

7.1 Research

All data structures which is mentioned in background sections will be researched while first 5 weeks of the project. In order to create meaningful datasets, and implement the binary search trees, this step is vital. All team members are responsible for different trees:

Emre	AVL Tree
İdil	RB Tree
Çağla	Splay Tree

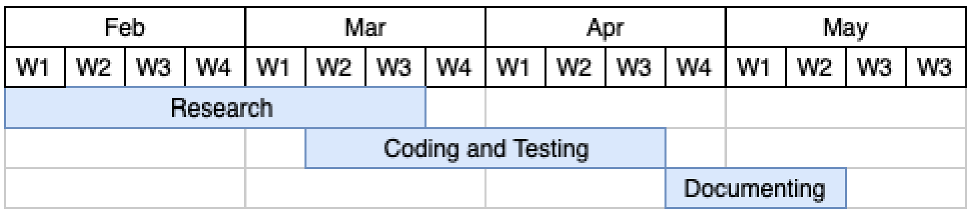
7.2 Coding and Testing

After the research, the team members will be able to create meaningful datasets, implement the assigned data structures, and test it with different cases. Each test will be consisting of three different operation: insertion, deletion, and lookup.

7.3 Documentation

Finally, all test results and outcomes will be documented. In order to visualize the runtime analysis, the final report will be enhanced with graphs and tables. All members are responsible for this step.

7.4 GANTT Chart



7.5 RACI Analysis

	Research	Implementation of AVL Tree	Implementation of RB Tree	Implementation of Splay Tree	Runtime Comparison for AVL Tree	Runtime Comparison for RB Tree	Runtime Comparison for Splay Tree
Emre	RA	RA	CI	CI	RA	CI	CI
İdil	RA	CI	RA	CI	CI	RA	CI
Çağla	RA	CI	CI	RA	CI	CI	RA

8. Conclusion

As a conclusion, to find a more suitable implementation for the caching system of the company, AVL trees, red-black trees and splay trees are going to be compared with each other considering their time cost for the storage and search operations for different datasets. The necessary datasets, which are going to include sorted, unsorted and random data samples, are going to be built by us. After the performances of the mentioned self-balanced binary search trees for the required datasets are measured using Python programming language, more favorable structure for the different cases is going to be specified in the final report referring to the analyzes of the measurements.

9. References

[1] Bounif Lynda and Zegour Djamel. Eddine, *AVL and Red-Black Tree as a Single Balanced Tree*, March 2016.
[2] Baer, J.-L and Schwab, B., *A comparison of tree-balancing algorithms*, Communications of the ACM, 1977, v. 20, n. 5, pp. 322-330
[3] Cormen, Thomas, Charles Leiserson, Ronald Rivest, and Clifford Stein, *Introduction to Algorithms*, 2nd ed. MITPress, 2002, pp. 296
[4] Jeff Edmonds, *How to think about algorithms,* Cambridge University Press, 2008, USA
[5] Dominics Sleator and Endre Tarjan, *Self-Adjusting Binary Search Trees,* Journal of the Association for Computer Machinery, Vol 32, No: 3, July 1985, pp. 625-628