The study of the efficiency and speed between differing non-linear data structures

By: Michael Baldwin

Written on: November 9th, 2019

For: EECS 2510

Contents

[Introduction 2](#_Toc24237418)

[No Tree Findings 2](#_Toc24237419)

[Red-Black Tree 2](#_Toc24237420)

[Initial thoughts and expectations 2](#_Toc24237421)

[Data Collected 2](#_Toc24237422)

[Conclusion 3](#_Toc24237423)

[AVL Tree 3](#_Toc24237424)

[Initial thoughts and expectations 3](#_Toc24237425)

[Data Collected 3](#_Toc24237426)

[Conclusion 3](#_Toc24237427)

[Binary Search Tree 3](#_Toc24237428)

[Initial thoughts and expectations 3](#_Toc24237429)

[Data Collected 4](#_Toc24237430)

[Conclusion 4](#_Toc24237431)

[Skip Lists 4](#_Toc24237432)

[Initial thoughts and expectations 4](#_Toc24237433)

[Data Collected 4](#_Toc24237434)

[Conclusion 5](#_Toc24237435)

[Concluding remarks 5](#_Toc24237436)

# Introduction

In this report we will be examining the difference between three tree data structures and a fourth linear/non-linear data structure. The three tree structures we will be examining, in order, are the Red-Black Tree, the AVL tree, and the Binary Search Tree. We will finally be looking at the linear/non-linear data structure, skip lists. The first part of each data structure report will cover my initial thoughts and expectations, including problems encountered with the implementation and initial guesses at time/work performed by the data structures. Following that will be the data collected for the data structure. And finally, the concluding remarks about each data structure.

# No Tree Findings

Initial Non-Tree Code run with Shakespeare took 0.412 seconds and 0.414 seconds.

Initial Non-Tree Code run with ALL.txt took 2.695 seconds and 2.753 seconds.

# Red-Black Tree

## Initial thoughts and expectations

I had originally thought that the Red-Black tree would outperform all the other data structures in at least time spent. I figured the work performed would be less than AVL but more than BST.

## Data Collected

In the initial run of 8200, 4 letter words, entered into the tree the time spent was around 6.2 seconds. The work performed, which was a combination of key comparisons made and node recoloring, was 570022.

**Shakespeare.txt**

Time: 0.6 – 0.414 = 0.186 seconds

Work: 18774182

**ALL.txt**

Time: 4.042 – 2.753 = 1.289 seconds

Work: 135519937

## Conclusion

My initial conclusion is that the tree performed great. In comparison to the other data structures that we will see further in this report, the tree was very well suited for data with lots of insertions.

# AVL Tree

## Initial thoughts and expectations

Having the majority of the code already present was initially a good thing. Having to figure out little bugs in the rotations was not. I initially though that AVL would perform better than BST, but worse than RBT and Skip Lists.

## Data Collected

In the initial run of 8200, 4 letter words, entered into the tree the time spent was around 2.513 seconds. The work performed, which was a combination of key comparisons made and branch factors set, was 262359.

**Shakespeare.txt**

Time: 0.624 – 0.414 = 0.21 seconds

Work: 18554518

**ALL.txt**

Time: 4.157 – 2.753 = 1.404 seconds

Work: 133925525

## Conclusion

My initial conclusion is that the tree performed rather well. The speed was almost on par with the red black tree and the work performed wasn’t very bad either.

# Binary Search Tree

## Initial thoughts and expectations

I had originally thought that the Binary Search tree would outperform all the other data structures in at least time spent, it needs no rotations. But I figured it would not perform well with searches as having data in order would make it extremely unbalanced.

## Data Collected

In the initial run of 8200, 4 letter words, entered into the tree the time spent was around 3.016 seconds. The work performed, which was a combination of key comparisons made and node links changed/updated, was 100855900.

**Shakespeare.txt**

Time: 0.636 – 0.414 = 0.222 seconds

Work: 24985513

**ALL.txt**

Time: 4.222 – 2.753 = seconds

Work: 170895251

## Conclusion

My initial conclusion is that the tree performed great with ordered data, just because no rotations were needed. But any searches would definitely prove slow.

# Skip Lists

## Initial thoughts and expectations

I had originally hoped that the skip list would outperform all the trees, but as you can see from the data, my implementation was lacking.

## Data Collected

In the initial run of 8200, 4 letter words, entered into the tree, the time spent was around 1.905 seconds. The work performed, which was a combination of key comparisons made, head tosses, and pointer (u, d, l, r) was 97439.

**Shakespeare.txt**

Time: 13.819 – 0.414 = 13.405 seconds

Work: 22908473

**ALL.txt**

Time: 531.686 – 2.753 = 578.933 seconds

Work: 161885793

## Conclusion

The skip list exceeded at data sent in order, and I expect that searches with it would perform quickly. Unfortunately, the txt files seemed to of causes it to slowly creep along. It might be my implementation of the skip list insertion. It would be fascinating to see if there is a better implementation out there than can outperform some of the other data structures we tested so far.

# Concluding remarks

My concluding remarks would have to be that this project, although not nearly as complex as the Huffman Encoding project, it was more difficult. Having to figure out the small bugs that cause rotation issues in the AVL tree 100 insertions later was not something I would like to do again. Overall, the project was definitely a intriguing one, but I vote that it should be named AVL rotations (with other trees included) for the next semesters classes!