

CA Report

Object Oriented Programming 4



November 15, 2015

James McGarr

T00126681

# Introduction

For this report I have learned about the binary tree structure with the Java programming language.

# Code for Height Method

I had some trouble getting this method to work initially. I first added a count integer parameter to heightSub thinking that I needed to send over the current count, but later realised that this would be done in the opposite direction when the heightSub methods return a value. I could further simplify the code, but for readability I left it as it is. I had the code handle different cases individually even though this isn’t always necessary as many cases would return the same value.



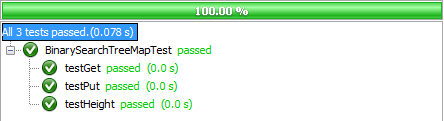
# Iterative or Recursive?

For many algorithms such as getting the Nth Fibonacci number, using an iterative approach would be better because N could be a very large number leading to a stack overflow exception in recursion. In the case of a binary tree however, this problem should only occur if the data being stored is extremely large or if the tree is very unbalanced. So as the height of a binary tree increases, assuming the tree will be balanced, the number of recursive calls needed would decrease per the amount of data being stored.

That being said, the height method in the previous section of this document can be implemented iteratively, but may not be as readable and clean as doing it recursively. A stack data structure such as Java’s LinkedList should be used as a suitable way of storing information in an iterative approach.

# Test Code for Height Method including a Test for Size

The size method did not exist so I added a size attribute to the binary search tree map class with an accessor method. The constructor initialises the size to 0. The put method increments the size attribute each time.



# Name and Source of File Used To Build Index

KingJamesBible – <http://www.gutenberg.org/cache/epub/10/pg10.txt>

# Profiling the Search Method of the Index Class

To profile the search method I created a JUnit test for its timing feature. Next I built up a list of words to search for and stored them in an array list. Then I built an index using the King James Bible which was named ‘pg10.txt’. Finally I had the program search Index for each of the words in ‘words.txt’ one thousand times.



|  |  |
| --- | --- |
| Index Based on: | Time taken in milliseconds(seconds to three decimal points) |
| BinarySearchTreeMap |  |
| TreeMap |  |

Obtained with **Intel i7-4510U CPU** processor, **2.6 GHz,** Java 8, Windows 8.1

# Conclusion from Profile Results

The Java TreeMap re-arranges its nodes to keep balance which will increase the efficiency of any search. This is called a Red-Black tree based navigable map implementation. (JavaTreeMap, 2015) Our Binary Search Tree Map does not maintain balance so its balance will depend on how it is populated. In the worst case scenario this map might be populated by a file whose elements are already ordered, such as the letters of the alphabet from ‘A’ to ‘Z’. In this case the benefit of using a TreeMap is lost.

The discrepancy in the amount of time it takes to search each binary tree is determined by how balanced they are.

# Bibliography

JavaTreeMap, 2015. *docs.oracle.com.* [Online]   
Available at: http://docs.oracle.com/javase/7/docs/api/java/util/TreeMap.html  
[Accessed 16 11 2015].