ID1021: Trees

Alexander Lundqvist

26-09-2022

1 Introduction

This report covers the 6th assignment in the course. The goal of the assignment is for the students to get an understanding in how one can apply the concept of linked data to create complex data structures such as trees. The main focus here is binary trees. A binary tree can be described with the following image.

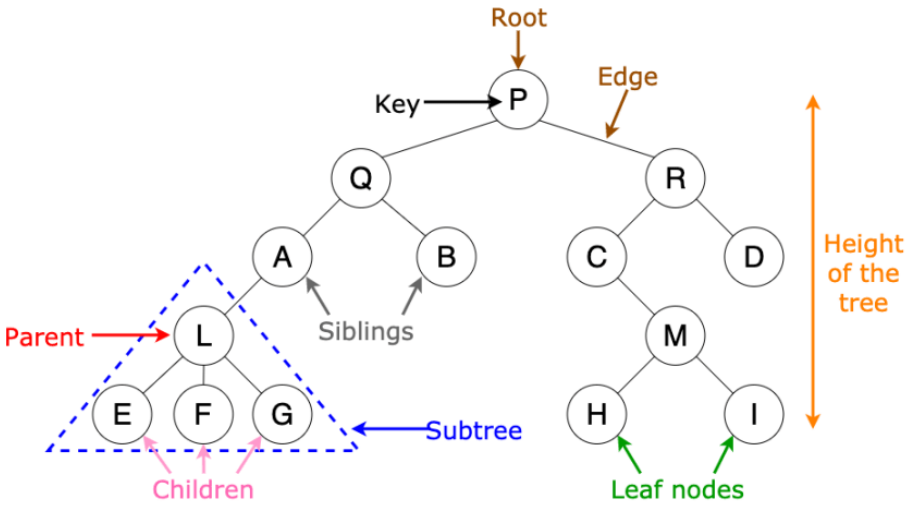

Figure 1: A tree structure with explanation of components.

2 Method

As a tree builds on the ideas of the linked list structure we did in the previous assignment, we can use it as a starting point. In the instructions we are provided a basic skeleton of the code where we are supposed to implement two basic functions, add and lookup. Furthermore we have to utilize a auxiliary node structure for the linking process.

The node class was written as a separate class to remove cluttering from the binary tree class. I decided to use String type objects for values instead of using Integer objects for both keys and values, as this makes testing and console output easier to understand.

To construct the binary tree I re-used code from one of my older projects where I constructed a more complex Binary Search Tree or BST. The assignment also calls for a benchmark so I used the same benchmark method used in previous assignments. The benchmark will also be used against the binary search method we did earlier so I modified my old code to use Integer objects as I found out that time measurements gets affected differently when using objects vs. primitive values.

We also modify the the node class with a print method that follows the depth first principle. This is only done to illustrate that we use javas built in stack when traversing the tree structure. We will implement our own iterator as the final task.

2.1 Iterator

To support the iterator method we will utilize a stack. I reused my old stack that I wrote for the calculator assignment. I modified it to accept generics, as I might want to use the code for later assignments. The iterator template looks like this(I omit the code that we didn't have to implement):

```
private Node next;
private Stack<Node> stack;

public TreeIterator() {
}

@Override
public boolean hasNext() {
}

@Override
public Integer next() {
}
```

As we can see we have three methods to implement. We start with the

constructor method. To start of I changed the name next to current since I found it easier to grasp the code if I only focus on the current node. As the assignment tells us to travel down the leftmost part in the tree, I initialize it with a while loop that travels down and sets the current node to the leftmost node while simultaneously pushing the nodes to the stack.

For the HasNext method I first thought I had to check if the following next left or right node was null, but I did some reading on how an iterator should function and realised it should check whether we have anything left to "return" from the tree. This was done simply by checking if the stack had had any elements left. As my stack have private attributes (which it should) I had to implement a public function to return a boolean value if the stack was empty or not.

The next method was the trickiest part. I knew that when the iterator was initialized we would already have a pointer to the first value that we wanted to return. I pop the current value from the stack and keep its reference as a temporary variable. I then want to see if the node has any right node and if so, set the current to the right node and do the same travel as I did in the initialization.

3 Result

Here we present the result of the execution time of the lookup method. Figure 2 illustrates the binary tree.

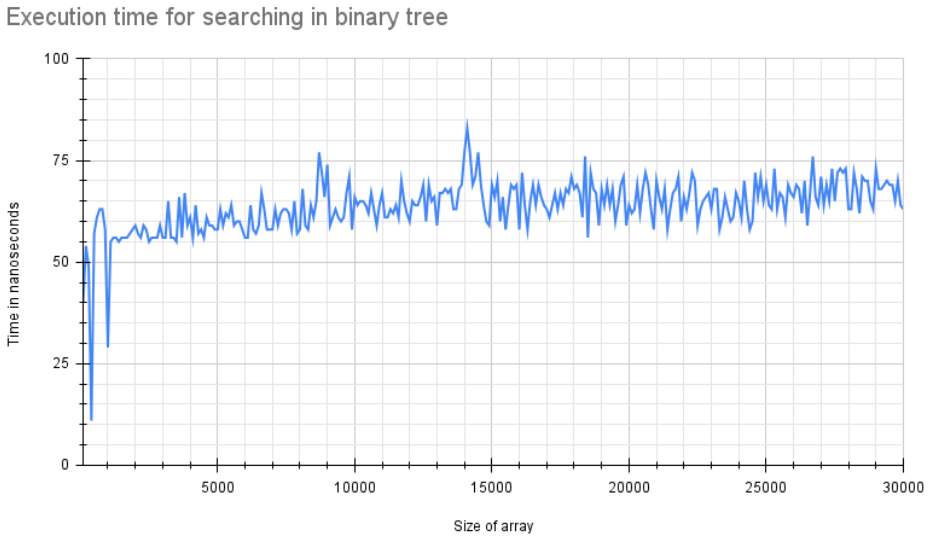

Figure 2: Execution time for search in binary tree.

Figure 3 illustrates the execution time for the binary search class. Note that the graphs looks quite similar. This will be discussed in the last section.

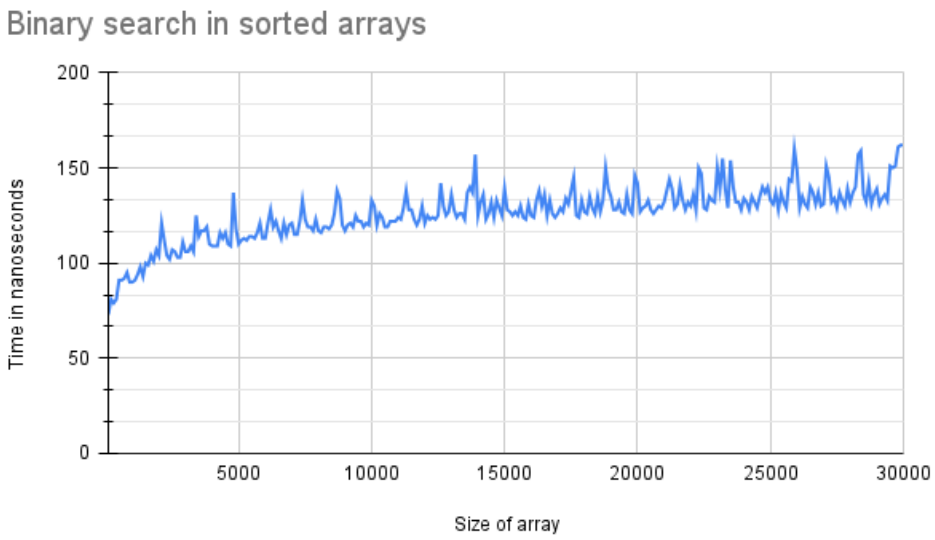

Figure 3: Execution time for binary search.

In figure 4 is also a sample printout of the depth first print method. The values of the nodes are named as they are input into the tree.

```
DFS.put(5,"Root");
DFS.put(2,"RootLeft");
DFS.put(1,"RootLeftLeft");
DFS.put(8,"RootRight");
DFS.put(6,"RootRightLeft");
DFS.put(3,"RootLeftRight");
```

```
run:
Key: 1  Value: RootLeftLeft
Key: 2  Value: RootLeft
Key: 3  Value: RootLeftRight
Key: 5  Value: Root
Key: 6  Value: RootRightLeft
Key: 8  Value: RootRight
BUILD SUCCESSFUL (total time: 1 second)
```

Figure 4: Depth first traversal print method.

3.1 Iterator

The final code for the iterator is presented here. The output is equal to the depth first print method.

```
public class TreeIterator implements Iterator<String> {
        private Node current;
        private Stack<Node> stack;

        public TreeIterator() {
            this.stack = new Stack<Node>();
            this.current = root;

            // Travel down left side
            while (this.current != null) {
                this.stack.push(this.current);
                this.current = this.current.left;
            }
        }
```

```
    @Override
    public boolean hasNext() {
        return !this.stack.isEmpty();
    }

    @Override
    public String next() {
        if (!hasNext()) throw new NoSuchElementException();

        Node temp = this.stack.pop();
        String value = temp.value;

        if (temp.right != null) {
            this.current = temp.right;
            while (this.current != null) {
                this.stack.push(this.current);
                this.current = this.current.left;
            }
        }

        return value;
    }
}
```

The entire code is available at this github repository.

4 Discussion

The assignment asks us the question what would happen if we would input an ordered sequence of keys. The answer is that it would essentially become just a linked list. We could absolutely do this but it makes makes the tree class redundant and we could just use a linked list structure. In technical view it is called a left or right skewed binary tree. There is something called self balancing binary trees that accounts for this problem but unfortunately we are not exploring that in this assignment.

4.1 Binary search vs. Binary tree

My initial testing showed that the binary search was much faster for some reason. As previously mentioned, I instead used the binary search on Integer objects. This is because I wanted the measurements to be equal since I have noted that time measurements gets affected by using primitive values. After the changes I got much better results.

At a first glance the graphs looks kind of similar but I did a scatter plot of the binary tree data and it looked more like a linear function. Unfortunately my laptop couldn't handle very large data sets so I had to make do with the data. I then did some research into this matter and it seems that searching in binary trees have $O(log(n))$ time complexity. As this is a recursive function and I don't have proficient knowledge in the master theorem I can't give the actual proof here. Further reading reveals that the matter is more complex as time complexities in tree structures are dependent on the height of the tree and amount of vertices/nodes.

4.2 Depth first traversal

We also briefly touch on the subject of traversal methods. As trees are a linked data structure it is not really possible to access any element by index like with arrays, where we can access with $O(1)$ time. In the assignment we explore the concept of depth first traversal with the node class print method that was given. Depth first traversal not interesting in itself, but when we start using it for searching it becomes something that we could benchmark against the Breadth first search principle. This is however something that we most likely will explore in the later graph assignment.

4.3 Iterator

The most difficult task was the implementation of the iterator. I have not implemented such methods before and I had only vague understanding of the idea. I actually had to draw the tree on paper to understand how the iterator should move through the structure. I forgot about the the task about what happens if i create the iterator, iterate then change the tree

structure and by the time I was done with the report I had no time left. I will gladly explain it in a completion assignment if necessary.