CMPT 225: Data Structures & Programming
– Unit 25.5 –
Interlude on Advanced Trees

Dr. Jack Thomas

Simon Fraser University

Spring 2021

Today's Topics

- Tree Recap
- TreeMap and TreeSet
- Red-Black Trees
- The Last Word on Trees

Tree Recap – Treecap?

- This section of the course has introduced or expanded upon a variety of more **advanced Trees**:
  - **Binary Search Trees**
  - **AVL Trees**
  - **Multi-Way Trees**
  - **(2, 4) Trees**
- These key-based Tree structures are good candidates for building efficient versions of other data structures, like **Ordered Maps**.

Then Where Are The Classes and Interfaces?

- Despite their usefulness, these Trees have **very little support in the standard Java library**, which would be nice given how complicated they are to implement.
- This is because, once again, Java is not beholden to implement every theoretical data structure just because they exist – if two Trees have the same performance, implementing both of them because they achieve it differently is redundant.

What Java Offers: TreeMap and TreeSet

- Instead of offering a Tree directly, Java uses a self-balancing Search Tree to implement a Map class and a Set class (more on Sets to come), which benefit from the Tree to provide O(log n) run-time methods.
- These cover most practical use-cases for one of these Trees, any more niche application will probably require the programmer to customize their Tree implementation anyway.

Quick TreeMap Interlude (Double Interlude?)

- TreeMap works just like a Map implemented by a Tree, meaning that it has all the expected methods of a Map but with the run-time efficiency of a Tree.

```
TreeMap<Integer, String> exampleTreeMap = new TreeMap<>();
exampleTreeMap.put(1, "First TreeMap Entry");
exampleTreeMap.put(2, "Second TreeMap Entry");
exampleTreeMap.put(3, "Third TreeMap Entry");
System.out.println(exampleTreeMap.remove( key: 1));
System.out.println(exampleTreeMap.remove( key: 2));
System.out.println(exampleTreeMap.remove( key: 3));
```

```
First TreeMap Entry
Second TreeMap Entry
Third TreeMap Entry
```

TreeMap as an Ordered Map

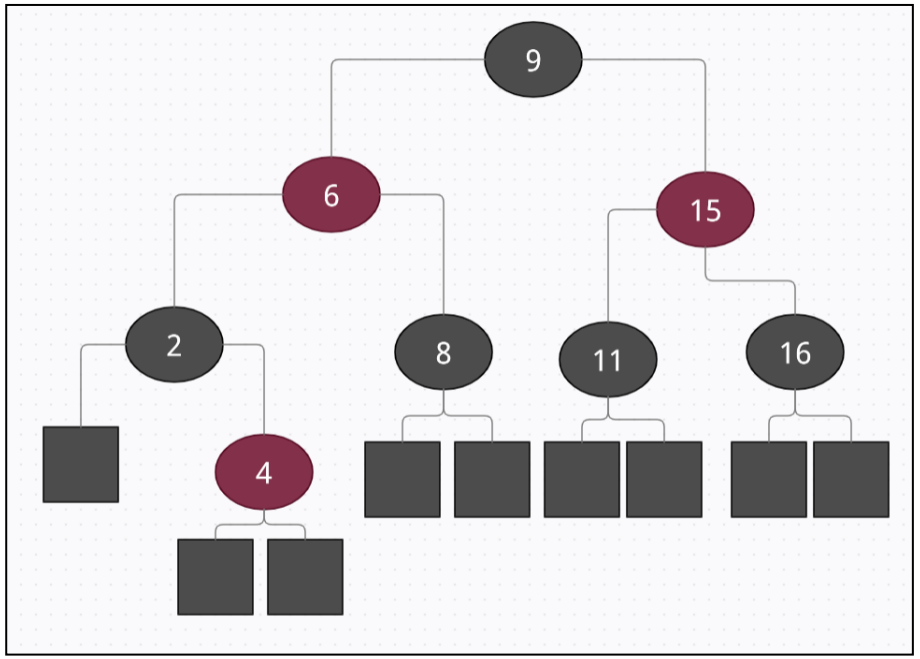
- This is actually slower than the near-constant-time performance of HashMap (the Hashtable implementation of Map), but TreeMap stores the entries in order.

```
exampleTreeMap.put(1, "First TreeMap Entry");
exampleTreeMap.put(2, "Second TreeMap Entry");
exampleTreeMap.put(3, "Third TreeMap Entry");
SortedMap<Integer, String> subMap = exampleTreeMap.subMap(2,3);
TreeMap<Integer, String> subTreeMap = new TreeMap<>(subMap);
System.out.println(subTreeMap.firstEntry().getValue());
```

```
Second TreeMap Entry
```

Okay, Then Which Tree Are Those Classes Based On?

- **Red-Black Trees.**
- They're yet another type of **self-balancing Binary Search Tree**, which achieves its balance through designating **every node as either red or black**, and maintaining rules about the alternating colour of parents and children and the number of black nodes on any one path from the root.



Why Aren't We Doing a Full Unit on Red-Black Trees, Then?

- Fair question.
- Since Java already covers the major use-cases for Trees with a Red-Black Tree, that kind of Tree is **already pretty well-represented in the Standard Library**.
- I can be confident you'll encounter and learn about Red-Black Trees over the course of your programming career, whether or not I introduce them to you now.
- Instead, spending our time introducing you to the wider world of Trees provides more theoretical background you wouldn't normally need to seek out on your own.

Still Seems Kinda Weak

- Yeah okay it kind of is.
- Actually we've just spent a whole lot of time on Trees and we've still got other topics to cover before the end of the course.
- There's only so much time I can spend showing you different ways to self-balance a Search Tree to get O(log n) time before we hit diminishing returns, so let's just skip ahead.

Also We Were Already A Little Behind And Then Your Dog Ate Chocolate Cake And You Cancelled a Lecture

- Shhshshshhhshshh
- She’s fine

Recap – I Already Said Treecap

- There are a number of ways to implement **self-balancing Search Trees** that bound a Tree's height to log n and search, insert, and delete to O(log n).
- Since they're **mostly just used to implement other structures**, and you really just need one of them for that, they're **not heavily supported in Java and other languages**.
- Nevertheless, they're useful to know about for your **Computer Science theory background**, and so you'll **recognize when they would apply**.