COP 3503
Lecture Notes

Mark Williams

Contents

Chapter 0

Preface

These notes are from COP 3503 Spring 2023. I use these notes to take after class and prepare me for exams, midterms, and finals.

Note:-

Imports to Remember

```
import java.util.*;
import java.awt.Point;
```

Definition 0.1: Some definitions

BST: Binary Search Tree

Chapter 1

Exam 1

1.1 Runtime and Runtime Analysis

1.1.1 The formal definition of Big-Oh

Big-Oh is the way in which Computer Scientists mathematically question the running times of our programs. It allows a general understanding how an algorithm will work with large, or infinitely large datasets or inputs.

Definition 1.1: Big-Oh, Big-Ω and Big-Θ

Let c, N represent constants where c is some constant towards the functions, $c_1 > 0$, and N be where $cg(n)$ meets or exceeds $f(n)$.

Big-Oh:

$$f(n) = O(g(n)) \text{ iff } f(n) \leq c_1 g(n) \text{ for } n \geq N_0$$

Big-Ω:

$$f(n) = \Omega(g(n)) \text{ iff } f(n) \geq c_1 g(n) \text{ for } n \geq N_0$$

Big-Θ:

$$f(n) = \Theta(g(n)) \text{ iff: } \begin{cases} f(n) = O(g(n)) \\ \wedge \\ f(n) = \Omega(g(n)) \end{cases}$$

Big-Oh is used to find an **upper bound** of a function. Big-Ω is used to find a **lower bound** of a function.

1.2 Hash Table

1.3 Hash Containers Review

1.3.1 HashSet and HashMap

Both HashSet and HashMap are used to store items in a set or array such that they are accessible in constant time, however because of the issues that could arise from hashing, this isn't always possible. The two defined above, in their simplist methods have an upper bound of $O(n)$ and a lower bound of $\Omega(n)$.

HashSet: a HashSet is a set that implements the uses of hashing and hash codes. Alike a set, the functions are add(), remove(), contains(), and size().

HashMap: a HashMap is very similar however it houses key value pairs where the key is what is used as a hashcode and the value is what is stored. Both can be searched and queried with containsKey(), containsValue(). This class however changes their priorty functions of retrieval and inputting to get() and put(), however size() remains.

1.4 Trees

1.4.1 AVL Trees Review

1.4.2 2-4 Trees

Insertion: The basic premise of 2-4 Trees is that each node has either no children or between two and four, hence the name 2-4 trees. In 2-4 tree algorithms numbers are continuously inserted into a node with a array with a maximum of four elements. When that number rises above three, we split the node into two separate arrays at some split point. This splitting action is defined as "splitting" or "SQUISHING UP." The split point or squish up point is somewhat arbitrary and almost any number can be selected, however it is wise to use a number that would most evenly split the array, preferably at point two or three. For the scope of this class, that split point will be at point three.

When this squish up action occurs and there exists a head node, we must push this splitted node upwards into the head node.

Deletion: When a node is deleted that is *not* a leaf node, we must take the largest value on the left subtree or smallest value on the right subtree and replace it with the number that was deleted.

When a leaf node is deleted, we must take from a sibling that has more than one value and push it upwards so that the connected value from the parent is drop down to the missing leaf node. However, if no sibling has more than one node then we must fuse with that sibling and push the connected parent value down to create a node with an array of two values.

1.4.3 Treaps

Definition 1.2: Heaps

Heaps are data structures defined as a tree, however it has a structural property that forces to fill the tree in order of the minimum height. Heaps are also in sorted order.

Treaps are the combination of the data structures trees and heaps. It stores data like a tree, in logmarithmic time, however how that data is configured is based on a heap. Every time a node is inserted, it is given a random unique priority. If this priority doesn't fit the heap, then the node is rotated until it fits.

Note:-

Treaps do not follow the structural property that heaps have.

Insertion: When values are inserted into the tree, they are given a random, unique, priority. Insertion is the same as BST

Deletion: