Algorithms and Data Structures for Data Science
Trees 4

CS 277
Brad Solomon

October 20, 2021
Exam 2 October 29th - October 31st

Exam will be combination multiple choice and coding

Covers all content up until October 20th

Sign up for registration as early as October 14th

Coding question will use new workspace setup.
Learning Objectives

Discuss efficiency of BSTs

Identify best, worst, and average case for BSTs

Calculate height balance

**NOT ON EXAM:** Introduce tree rotations
A **BST** is a binary tree $T = \text{treeNode}(\text{val}, T_L, T_r)$ such that:

$\forall n \in T_L, \ n.\text{val} < T.\text{val}$

$\forall n \in T_R, \ n.\text{val} > T.\text{val}$
# BST Analysis – Running Time

<table>
<thead>
<tr>
<th>Operation</th>
<th>BST Worst Case</th>
</tr>
</thead>
<tbody>
<tr>
<td>find</td>
<td></td>
</tr>
<tr>
<td>insert</td>
<td></td>
</tr>
<tr>
<td>remove</td>
<td></td>
</tr>
<tr>
<td>traverse</td>
<td></td>
</tr>
</tbody>
</table>
def find(self, key):
    n = self.find_helper(self.root, key)
    if n:
        return n.val
    else:
        return None

def find_helper(self, node, key):
    nkey = node.key
    if nkey > key:
        if node.left:
            return self.find_helper(node.left, key)
        else:
            return None
    elif nkey < key:
        if node.right:
            return self.find_helper(node.right, key)
        else:
            return None
    else:
        return node
BST Insert

Base Case

Recursive Step

Combining
def remove_helper(self, node, key):
    if node == None:
        return None
    if node.key > key:
        node.left = self.remove_helper(node.left, key)
    elif node.key < key:
        node.right = self.remove_helper(node.right, key)
    else:
        if node.left == None:
            temp = node.right
            return temp
        if node.right == None:
            temp = node.left
            return temp
        iop = self.findIOP(node)
        node.key = iop.key
        node.val = iop.val
        node.left = self.remove_helper(node.left, iop.key)
    return node

def remove(self, key):
    self.root = self.remove_helper(self.root, key)
BST Analysis

Every operation on a BST depends on the height of the tree.

… how do we relate $O(h)$ to $n$, the size of our dataset?
BST Analysis

What is the max number of nodes in a tree of height $h$?
BST Analysis

What is the min number of nodes in a tree of height $h$?
BST Analysis

For all BSTs:

Lower-bound:

Upper-bound:
Tree Property: “Full”

A tree $F$ is \textbf{full} if (and only if) for each node $n$:

- $n$ has zero children
- OR
- $n$ has two children
Tree Property: “Perfect”

A **perfect** tree $P_h$ is a tree of height $h$ where:

$$P_{h-1} = \text{None}$$

AND

$$P_h = \{\text{key, value, } P_{h-1}, P_{h-1}\}$$
Tree Property: “Complete”

A complete tree $C_h$ is a tree of height $h$ where:

$C_{-1} = \text{None}$

AND

$C_h = \{\text{key, value, } C_{h-1}, P_{h-2}\}$

OR

$C_h = \{\text{key, value, } P_{h-1}, P_{h-1}\}$
Which tree properties are “good”?
BST Analysis

The height of a BST depends on the order in which the data was inserted

**Insert Order:** [1, 3, 2, 4, 5, 6, 7]

**Insert Order:** [4, 2, 3, 6, 7, 1, 5]
BST Analysis

How many different ways are there to insert $n$ keys into a BST?

**Claim:** The average height of all arrangements is $O(\log n)$
Height-Balanced Tree

What tree is better?

**Height balance:** \( b = \text{height}(T_R) - \text{height}(T_L) \)

A tree is “balanced” if:
Recursive balance calculations
Modifying insert (for height)

```
insert(98)
```
# BST Analysis — Running Time ($n$)

<table>
<thead>
<tr>
<th>Operation</th>
<th>BST Average case</th>
<th>BST Worst case</th>
<th>Sorted array</th>
</tr>
</thead>
<tbody>
<tr>
<td>find</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>insert</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>remove</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>traverse</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
End of Exam 2 Material!

- Sorting
- Search
- Recursion
- Trees
Sorting
Recursion
Trees
What will the tree structure look like if we remove node 16 using IOS?
THIS MATERIAL IS NOT ON EXAM!
BST Rotations

We can adjust the BST structure by performing rotations.
BST Rotations
BST Rotations
BST Rotations
BST Rotations

A binary search tree with the following structure:

- Node 38
  - Node 13
    - Node 10
    - Node 25
    - Node 37
  - Node 51
BST Rotations

Next week we will define four kinds of rotations (L, R, LR, RL)

We will see that:

1. All rotations are local (subtrees are not impacted)
2. The running time of rotations are constant
3. The rotations maintain BST property

Motivation for rotations:

We call these trees: