Chapter 5, Binary search trees:

- Binary search trees; the balanced BST problem (fall-break eve; finishing Today)
- AVL trees (Today and Wednesday)
- Traditional red-black trees (Friday)
- Left-leaning red-black trees (next week Monday)
- “Wrap-up” BST (next week Wednesday)

Today and Wednesday:

- Review BST basics and code base
- BST performance and the balanced BST problem
- Rotations; overview of solutions
- AVL tree definition
- AVL tree cases
- AVL tree performance
Coming up:

- Catch up on older projects
- Do **BST rotations** project (due Wed, Mar 13)
- Do **AVL trees** project (due Mon, Mar 18)

*Due Tues, Mar 12* (end-of-day)
Read Section 5.(1 & 2)
Do Exercises 5.(2 & 6)
Take quiz (BSTs)

*Due Thurs, Mar 14* (end of day)
Read Section 5.3
Do Exercises 5.(7 & 8)
Take quiz (AVL trees)

*Due Tues, Mar 19* (end of day)—but spread it out
Read Sections 5.(4-6)
Take quiz (red-black trees)
A **binary search tree** (BST) over some ordered key type is either

- empty, or
- a node augmented with a key $k$ together with two children, designated *left* and *right*, such that
  - *left* is a binary search tree such that all of the keys in that tree are less than or equal to $k$, and
  - *right* is a binary search tree such that all of the keys in that tree are greater than or equal to $k$.

<table>
<thead>
<tr>
<th></th>
<th>Unsorted</th>
<th>Sorted</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Array</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Find</td>
<td>$\Theta(n)$</td>
<td>$\Theta(lg\ n)$</td>
</tr>
<tr>
<td>Insert</td>
<td>$\Theta(1)$ expected, $\Theta(n)$ worst</td>
<td>$\Theta(n)$</td>
</tr>
<tr>
<td>Delete</td>
<td>$\Theta(n)$</td>
<td>$\Theta(n)$</td>
</tr>
<tr>
<td><strong>Linked structure</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Find</td>
<td>$\Theta(n)$</td>
<td>$\Theta(n)$</td>
</tr>
<tr>
<td>Insert</td>
<td>$\Theta(1)$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td>Delete</td>
<td>$\Theta(1)$</td>
<td>$\Theta(1)$</td>
</tr>
</tbody>
</table>
The *height* of a node (or (sub)tree) is the number of nodes on any path from that node to any leaf, inclusive.

\[
height(c) = \begin{cases} 
0 & \text{if } c \text{ is null} \\
\max(\text{height}(c.\ell) + \text{height}(c.r)) + 1 & \text{otherwise}
\end{cases}
\]

The *balance* of a node is the difference between the heights of its left and right children. In an AVL tree, each node’s subtrees’ heights must differ by at most 1:

\[
\forall x \in \text{nodes}, |\text{height}(x.\text{left}) - \text{height}(x.\text{right})| \leq 1
\]

A node that has balance 1 or -1 has a *bias*. A node that (temporarily) has balance 2 or -2 is in *violation*.

(A balance less than -2 or greater than 2 shouldn’t happen even temporarily.)
The diagram illustrates the process of inserting and rotating a node in a tree.

1. Initially, we have a tree with nodes labeled A, B, C, D, E, F, G, and I.
2. An insert operation is performed, which results in the tree structure shown.
3. A rotate operation is then applied, which results in the final tree structure shown.

The nodes are labeled with values that represent their position in the tree.

- Insertion results in a tree with nodes A, B, C, D, E, F, G, and I.
- Rotation results in a tree with nodes D, E, F, G, H, I, J, and K.

The diagram also includes a note indicating that the rotation was performed incorrectly.
Right–Left:

\[
\begin{array}{c}
\text{A} \\
\text{(h) } \alpha \\
\text{[−2]} \\
\text{C} \\
\text{[−1 or 0 or 1]} \\
\text{B} \\
\text{β γ δ} \\
\end{array}
\]

rotate

\[
\begin{array}{c}
\text{A} \\
\text{(h) } \alpha \\
\text{B} \\
\text{β} \\
\text{C} \\
\text{γ δ} \\
\end{array}
\]

fall through

\[
\begin{array}{c}
\text{A} \\
\text{(h) } \alpha \\
\text{[−2]} \\
\text{B} \\
\text{β γ δ} \\
\end{array}
\]

Right–Right:

\[
\begin{array}{c}
\text{A} \\
\text{(h) } \alpha \\
\text{B} \\
\text{β γ δ} \\
\text{[−2 or −1 or 0]} \\
\text{C} \\
\text{γ δ} \\
\end{array}
\]

rotate

\[
\begin{array}{c}
\text{A} \\
\text{β} \\
\text{C} \\
\text{γ δ} \\
\end{array}
\]
Invariant 30 (Postconditions of `RealNode.put()` with AVLBalancer.)
Let $x$ be the root of a subtree on which `put()` is called and $y$ be the node returned, that is, the root of the resulting subtree. The subtree rooted at $y$ has no violations and the height of the subtree rooted at $y$ is equal to or one greater than the original height of the subtree rooted at $x$.

Proof. Suppose `put()` is called on node $x$ in a BST using AVL balancing which has no violations. There are three cases: $x$ is `null`, $x$ is a `RealNode` containing the key being searched for, or $x$ is a `RealNode` with a different key. We use structural induction with the first two cases as base cases.
**Base case 1.** Suppose $x$ is `null`, which has height 0. Then the node $y$ returned is a new `RealNode` with `null` as both children, height 1, and balance 0. The subtree rooted at $y$ has no violations and height one greater than the original height of $x$.

**Base case 2.** Suppose $x$ is a `RealNode` whose key is equal to the key used for this `put()`. Then the value at node $x$ is overwritten but node $x$ itself is returned (so $y = x$ in this case) with the tree structure unchanged.

**Inductive case.** Suppose $x$ is a `RealNode` and, without loss of generality, the key used for this `put()` is greater than the key at $x$, and so `put()` is called on the right child of $x$. Let $h_0$ be the height of the tree rooted at $x$ before this call to `put()` on the right child, and let $z$ the root of the subtree that results from this call to `put()` on the right child. Our inductive hypothesis is that the subtree rooted at $z$ has no violations and that its height is equal to or one greater than the height of the original right child of $x$. 
Let $h_1$ be the height of the tree rooted at $x$ after the call to $\text{put()}$ on the right child but before the call to $\text{putFixup()}$ with $x$.

Since since at most the height of its right subtree has increased by one, either $h_1 = h_0$ or $h_1 = h_0 + 1$. By supposition, the balance of $x$ before the call to $\text{put()}$ was no less than $-1$, since we supposed the tree had no violations. Since at most the height of its right subtree has increased by one, the balance of $x$ is now no less than $-2$. We now have two subcases: Either the balance of $x$ is greater than $-2$ or it is equal to $-2$.

Suppose the balance of $x$ is greater than $-2$. Then the call to $\text{putFixup()}$ with $x$ returns $x$ unchanged, which is also returned as the result of $\text{put()}$ (again $y = x$), a tree with no violations and height $h_1$.

On the other hand, suppose the balance of $x$ is equal to $-2$. Then $y$ is a node other than $x$ returned by $\text{putFixup()}$. Let $h_2$ be the height of the subtree rooted at $y$ when $\text{putFixup()}$ returns. By inspection of the right-right and right-left subcases given above, the subtree rooted at $y$ has no violations and either $h_2 = h_1$ or $h_2 = h_1 - 1$. In either of those cases $h_2 = h_0$ or $h_2 = h_0 + 1$. \[\square\]
\[ B_h = \begin{cases} 
1 & \text{if } h = 1 \\
2 & \text{if } h = 2 \\
B_{h-2} + B_{h-1} + 1 & \text{otherwise} 
\end{cases} \]

\[ B_{h+1} = \begin{cases} 
2 & \text{if } h = 1 \\
3 & \text{if } h = 2 \\
(B_{h-2} + 1) + (B_{h-1} + 1) & \text{otherwise} 
\end{cases} \]

<table>
<thead>
<tr>
<th>( h )</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>( B_{h+1} )</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>8</td>
<td>13</td>
<td>21</td>
</tr>
<tr>
<td>( B_h )</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>7</td>
<td>12</td>
<td>20</td>
</tr>
</tbody>
</table>
\[ B_h + 1 > \frac{\phi^{h+2}}{\sqrt{5}} - 1 \]
\[ B_h + 2 > \frac{\phi^{h+2}}{\sqrt{5}} \]
\[ \sqrt{5}(B_h + 2) > \phi^{h+2} \]

\[ h + 2 < \log_\phi(\sqrt{5}B_h) \]
\[ h < \log_\phi(\sqrt{5}B_h) - 2 \]
\[ = \log_\phi B_h + \log_\phi \sqrt{5} - 2 \]
\[ = \frac{1}{\lg \phi} \lg B_h + \log_\phi \sqrt{5} - 2 \]
Coming up:

- Catch up on older projects
- Do **BST rotations** project (due Wed, Mar 13)
- Do **AVL trees** project (due Mon, Mar 18)

Due **Tues, Mar 12** (end-of-day)
Read Section 5.(1 & 2)
Do Exercises 5.(2 & 6)
Take quiz (BSTs)

Due **Thurs, Mar 14** (end of day)
Read Section 5.3
Do Exercises 5.(7 & 8)
Take quiz (AVL trees)

Due **Tues, Mar 19** (end of day)—but spread it out
Read Sections 5.(4-6)
Take quiz (red-black trees)