TODAY: Balanced BSTs
- The importance of being balanced
- AVL trees
  - definition & balance
  - rotations
  - insert
- Other balanced trees
- Data structures in general
- Lower bounds

Recall: Binary Search Trees (BSTs)
- rooted binary tree
- each node has
  - key
  - left pointer
  - right pointer
  - parent pointer
- BST property:
  \[
  \text{height of node} = \text{length (\# edges) of longest downward path to a leaf}
  \]

[Diagram of a balanced binary search tree]
The importance of being balanced:
- BSTs support insert, delete, min, max, next-larger, next-smaller, etc., in $O(h)$ time, where $h =$ height of tree (= height of root)
- $h$ is between $\lg n$ and $n$:
  - perfectly balanced
  - balanced BST maintains $h = O(\lg n)$
    ⇒ all operations run in $O(\lg n)$ time
AVL trees: [Adelson-Vel’skii & Landis 1962]
for every node, require heights
of left & right children
to differ by at most \( \pm 1 \)
- treat nil tree as height \(-1\)
- each node stores its height
  (DATA STRUCTURE AUGMENTATION) (like subtree size)
  (alternatively, can just store difference in heights)

Balance: worst when every node differs by 1
- let \( N_h = \text{(min.) \# nodes in height-}h \text{ AVL tree} \)
\[ \Rightarrow N_h = N_{h-1} + N_{h-2} + 1 \]
\[ > 2 \cdot N_{h-2} \]
\[ \Rightarrow N_h > 2^{h/2} \]
\[ \Rightarrow h < 2 \cdot \lg N_h \]

Alternatively: \( N_h > F_h \) (n-th Fibonacci number)
- in fact \( N_h = F_{h+2} - 1 \) (simple induction)
  \[ F_h = \phi^h / \sqrt{5} \text{, rounded to nearest integer} \]
  where \( \phi = \frac{1 + \sqrt{5}}{2} \approx 1.618 \) (golden ratio)
\[ \Rightarrow \text{max. } h \approx \log_\phi n \approx 1.440 \cdot \lg n \]
AVL insert:
1. insert as in simple BST
2. work your way up tree, restoring AVL property (and updating heights as you go)

Each step:
- suppose x is lowest node violating AVL
- assume x is right-heavy (left case symmetric)
- if x’s right child is right-heavy or balanced:

- else:

- then continue up to x’s grandparent, greatgp,...
Example:

Insert(23)  

$x = 29$: left-left case

Done.

Insert(55)

$x = 65$: left-right case

Done.

—in general may need several rotations before done with an Insert

—Delete(-min) is similar
AVL sort:
- insert each item into AVL tree
  \( \Theta(n \log n) \)
- in-order traversal
  \( \Theta(n) \)
  \( \Theta(n \log n) \)

Balanced search trees: there are many!
- AVL trees
- B-trees / 2-3-4 trees
- BB\([x] \) trees
- red-black trees
- splay trees
- skip lists
- scapegoat trees
- treaps

[Adelson-Velski & Landis 1962]
[Bayer & McCreight 1972]
[Nievergelt & Reingold 1973]
[CLRS ch. 13]
[Sleator & Tarjan 1985]
[Pugh 1989]
[Galperin & Rivest 1993]
[Seidel & Aragon 1996]

\( \mathbb{R} \) = use random numbers to make decisions fast with high probability
\( \mathbb{A} \) = “amortized”: adding up costs for several operations \( \Rightarrow \) fast on average

e.g. splay trees are a current research topic
  - see 6.854 (Advanced Algorithms)
  & 6.851 (Advanced Data Structures)
**Big picture:**

**Abstract Data Type (ADT):** interface spec.

**Data Structure (DS):** algorithm for each op.

- many possible DSs for one ADT
  - e.g. much later, "heap" priority queue

**Priority Queue ADT:**
- \( Q = \text{new-empty-queue}() \)
- \( Q.\text{insert}(x) \)
- \( x = Q.\text{deletemin}() \)
- \( x = Q.\text{findmin}() \)

<table>
<thead>
<tr>
<th></th>
<th>heap</th>
<th>AVL tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \Theta(1) )</td>
<td>( \Theta(1) )</td>
<td>( \Theta(1) )</td>
</tr>
<tr>
<td>( \Theta(\log n) )</td>
<td>( \Theta(\log n) )</td>
<td>( \Theta(\log n) )</td>
</tr>
<tr>
<td>( \Theta(1) )</td>
<td>( \Theta(1) )</td>
<td>( \Theta(1) )</td>
</tr>
</tbody>
</table>

**Predecessor/Successor ADT:**
- \( S = \text{new-empty}() \)
- \( S.\text{insert}(x) \)
- \( S.\text{delete}(x) \)
- \( y = S.\text{predecessor}(x) \)
- \( y = S.\text{successor}(x) \)

<table>
<thead>
<tr>
<th></th>
<th>heap</th>
<th>AVL tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \Theta(1) )</td>
<td>( \Theta(1) )</td>
<td>( \Theta(1) )</td>
</tr>
<tr>
<td>( \Theta(\log n) )</td>
<td>( \Theta(\log n) )</td>
<td>( \Theta(\log n) )</td>
</tr>
<tr>
<td>( \Theta(n) )</td>
<td>( \Theta(\log n) )</td>
<td>( \Theta(\log n) )</td>
</tr>
<tr>
<td>( \Theta(n) )</td>
<td>( \Theta(\log n) )</td>
<td>( \Theta(\log n) )</td>
</tr>
</tbody>
</table>