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:
  \[ x \leq \text{height of node} = \text{length (number of edges) of longest downward path to a leaf} \]

\[ \text{CLRS B.5} \]
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 $h = O(\lg n)$
  $\Rightarrow$ all operations run in $O(\lg n)$ time

vs.

path
AVL trees: [Adels'lon-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 AVL tree
  \( \Rightarrow N_h = N_{h-1} + N_{h-2} + 1 \)
  \( > 2 N_{h-2} \)
  \( \Rightarrow N_h > 2^{\frac{h}{2}} \)
  \( \Rightarrow h < 2 \lg N_h \)

Alternatively: \( N_h > F_h \) (\( n \)th Fibonacci number)
- in fact \( N_h = F_{n+2} - 1 \) (simple induction)
  \( F_h = \phi^h / \sqrt{5} \) rounded to nearest integer
  where \( \phi = \frac{1+\sqrt{5}}{2} \approx 1.618 \) (golden ratio)
  \( \Rightarrow \max. h \approx \log_\phi n \approx 1.440 \lg n \)
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, greatgrandpa,...
Example:

Insert(23)

\[ \xrightarrow{23} \]

\[ \xrightarrow{29} \]

\[ \xrightarrow{50} \]

\[ \xrightarrow{65} \]

\[ \xrightarrow{41} \]

\[ \xrightarrow{3} \]

\[ \xrightarrow{20} \]

\[ \xrightarrow{11} \]

\[ \xrightarrow{26} \]

\[ \xrightarrow{29} \]

\[ \xrightarrow{50} \]

\[ \xrightarrow{65} \]

\[ \xrightarrow{41} \]

\[ x=29: \text{left-left case} \]

Done.

Insert(55)

\[ \xrightarrow{55} \]

\[ \xrightarrow{65} \]

\[ \xrightarrow{41} \]

\[ \xrightarrow{3} \]

\[ \xrightarrow{20} \]

\[ \xrightarrow{11} \]

\[ \xrightarrow{26} \]

\[ \xrightarrow{50} \]

\[ \xrightarrow{65} \]

\[ \xrightarrow{41} \]

\[ \xrightarrow{3} \]

\[ x=65: \text{left-right case} \]

Done.

---

- in general may need several rotations before done with an Insert
- Delete(-min) is similar
AVL sort:
- insert each item into AVL tree
- in-order traversal

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

② = use random numbers to make decisions fast with high probability
① = “amortized”: adding up costs for several operations ⇒ 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.

vs. 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>DS</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>$\Theta(\log n)$</td>
<td>$\Theta(\log n)$</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>DS</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>
</tbody>
</table>
6.006 Introduction to Algorithms
Fall 2011

For information about citing these materials or our Terms of Use, visit: http://ocw.mit.edu/terms.