6.006 Lecture 4: Balanced BSTs

- Balance $\Rightarrow h = \Theta(\lg n) \Rightarrow \Theta(\lg n)$ operations
- AVL trees: definition, balance height, insert, rotations, insert
- "Big picture" on data structures
- CLRS 13.1 & 13.2, (red-black trees, not AVL trees)

Review

- BST property

- Operations (find, insert, delete, ...) take $\Theta(h)$ time where $h$ is the height of the tree (# edges downward to a leaf).

perfectly balanced
$h = \Theta(\lg n)$
$\Rightarrow \Theta(\lg n)$ operations

same $n$, poorly balanced
$\Theta(n)$ time operations
Balanced BST Strategy

- Augment every node with some property
- Define a local invariant on property
- Show (prove) that invariant guarantees $\Theta(\lg n)$ height
- Design algorithms to maintain property & to fix up the invariant.

AVL trees (Adel'son-Vel'skii - Landis 1962)

- Property: height, # edged to most distant leaf
- Invariant: heights of left & right children differ by at most $\pm 1$
  (define height of None as -1)

- Thm: Height of AVL tree $\leq 2\lg n$
  Proof: Let $N_h$ be min # nodes in height-$h$ AVL t.

\[ N_h = N_{h-2} + N_{h-1} + 1 \quad \text{(picture above)} \]

\[ > 2N_{h-2} \]

\[ N_0 = 1 \Rightarrow N_h > 2^{h/2} \Rightarrow \frac{h}{2} < \lg N_h \Rightarrow h < 2\lg N_h \]

For a given tree with height $h$ and $N$ nodes, $N > N_h$, so $h < 2\lg N$.\[ ... \]
- A tighter analysis (side note)

\[ h \quad 0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8 \quad 9 \quad 10 \]

\[ 1 + N_{h-2} \cdot N_{h-1} = N_h \]

\[ F_{h-2} + F_{h-1} = F_h \]

\[ N_h = F_{h+3} - 1 \]

\[ F_h = \frac{\phi^h}{\sqrt{5}} \quad \phi = \frac{1 + \sqrt{5}}{2} \approx 1.618 \quad \text{golden ratio} \]

\[ h = \log \phi F_h \quad \log \phi (N+1) + \log \sqrt{5} \]

\[ h + 3 = \log \phi (N+1) \pm \text{const} \]

\[ h = \log \phi (N+1) \pm \text{const} \approx \frac{\log(N)}{\log(\phi)} \pm \text{const} \approx 1.44 \cdot \log(N) \pm \text{const} \]
- When you insert (delete, etc), the invariant may get violated.

**Example: Insert(23)**

```
  41
 /   \
20    65
 /   /    \
11  29    50
|    |      |
26   23   29
```

**WHAT CAN WE DO?**

```
  3
 /   \
20  65
 /   /    \
11  29  50
|    |      |
23   50   29
```

A single rotation on 29

**Example: Insert(55)**

```
  41
 /   \
20    65
 /   /    \
11  29    50
|    |      |
26   23   55
```

```
  41
 /   \
20    65
 /   /    \
11  26    55
|    |      |
23   50   65
```

A double rotation

```
  65
 /   \
50  55
```

```
  55
 /   \
50  65
```

```
  55
 /   \
50  65
```

A double rotation
**General Rotation Rules**

**Case 1:**

\[ \begin{array}{c}
  \text{X} \\
  \text{Y} \\
  \text{A} \\
  \text{B} \\
  \text{C}
\end{array} \xrightarrow{k-1} \begin{array}{c}
  \text{K} \\
  \text{X} \\
  \text{Y} \\
  \text{A} \\
  \text{B}
\end{array} \xrightarrow{k} \begin{array}{c}
  \text{K} \\
  \text{X} \\
  \text{C} \\
  \text{A} \\
  \text{B}
\end{array} \]

**Case 2:**

\[ \begin{array}{c}
  \text{X} \\
  \text{Y} \\
  \text{A} \\
  \text{B} \\
  \text{C}
\end{array} \xrightarrow{k+1} \begin{array}{c}
  \text{K} \\
  \text{X} \\
  \text{Y} \\
  \text{A} \\
  \text{B}
\end{array} \xrightarrow{k-1} \begin{array}{c}
  \text{K} \\
  \text{X} \\
  \text{C} \\
  \text{A} \\
  \text{B}
\end{array} \]

**Case 3:**

\[ \begin{array}{c}
  \text{X} \\
  \text{Y} \\
  \text{A} \\
  \text{B} \\
  \text{C}
\end{array} \xrightarrow{k+1} \begin{array}{c}
  \text{K} \\
  \text{X} \\
  \text{Y} \\
  \text{A} \\
  \text{B}
\end{array} \xrightarrow{k-1} \begin{array}{c}
  \text{K} \\
  \text{X} \\
  \text{C} \\
  \text{A} \\
  \text{B}
\end{array} \]

\[ \text{violates our property; not a useful rotation on this tree} \]

**Case 3:** (again)

\[ \begin{array}{c}
  \text{X} \\
  \text{Y} \\
  \text{A} \\
  \text{B}_1 \\
  \text{B}_2
\end{array} \xrightarrow{k+1} \begin{array}{c}
  \text{K} \\
  \text{X} \\
  \text{Y} \\
  \text{A} \\
  \text{B}_1 \\
  \text{B}_2
\end{array} \xrightarrow{k-1} \begin{array}{c}
  \text{K} \\
  \text{X} \\
  \text{C} \\
  \text{A} \\
  \text{B}_1 \\
  \text{B}_2
\end{array} \]

- Here we assumed that right subtree of \( x \) is higher other case is symmetric.
- We also assumed that \( x \) is off-balance by 1.
- During insertions, this is always the case & we can fix up the balance going up the tree in \( \Theta(\ln n) \) time.
- In general, you can use rotations to bring in any shape tree to any other shape using rotations.
Other search trees

- AVL trees
- 2-3 trees $\Rightarrow$ B-trees
- weight-balanced trees a.k.a. BB[$\alpha$]
- red-black trees
- Splay trees
- Scapegoat trees
  - invented by Galperin
  - amortized insert, delete

Why did people invent so many trees?

- B-trees: very shallow, very little random access
- BB[$\alpha$]: tune insert/delete vs. search
- red-black: only 1 extra bit per node
- splay, scapegoat: no extra data in nodes, lower constants, but only amortized guarantees

(Insertions into lists in Python take constant amortized time; once in a while, the list is copied to a larger array to make room for more insertions; when do you trigger this?)

- van Emde Boas trees: $O(\log \log u)$ operations for integer key 1...u
The big picture: Abstract Data Types (ADT)

- Data-structure problem defines what operations to support
  - e.g.: Priority Queue
    - create empty queue
    - insert (x)
    - retrieve & delete minimum element

- Choice of DS driven by ADT requirements but not determined by them: BSTs are OK for priority queues but so are heaps.

- Another example: Union-Find
  - create a collection of sets \{S_1, S_2, \ldots, S_n\}
  - Union (i, j): merge the sets that contain i and j
  - Find (i): return a representative of the set containing i

\[
\begin{align*}
\text{init (4)} \\
\text{find}(3) &\rightarrow 3 \\
\text{union}(4, 2) \\
\text{find}(3) &\rightarrow 2 \\
\text{find}(2) &\rightarrow 2 \\
\ldots
\end{align*}
\]

- We specified the ADT completely but did not describe a DS; see CLRS Ch. 2.