Other/Better Criteria?
- Expected case: Some keys more popular than others
- Self-adjusting: Tree adapts as popularity changes

How to design/analyze?
- Splay Tree: A self-adjusting binary search tree
  - No rules! (yay anarchy!)
  - No balance factors
  - No limits on tree height
  - No colors/levels/priorities
  - Amortized efficiency:
    - Any single op - slow
    - Long series - efficient on avg.

Intuition: Let T be an unbalanced BST, suppose we access its deepest key.

Splay Trees I

Recap: Lots of search trees
- Unbalanced BSTs
- AVL Trees
- 2-3, Red-black, AA Trees
- Treaps + Skip lists

Focus: Worst-case or randomized expected case

Lesson: Different combinations of rotations can:
- bring given node to root
- significantly change (improve) tree structure.

Focus: Worst-case or randomized expected case

SPLAY Tree:
- A self-adjusting binary search tree
- How to design/analyze?

Trees' height has reduced by ~ half!

Idea I: Rotate "a" to top
(Future accesses to "a" fast)

Idea II: Rotate 2 at a time - upper + lower

Amortized efficiency:
- Any single op - slow
- Long series - efficient on avg.

Intuition: Let T be an unbalanced BST, suppose we access its deepest key.

...final result:
Splay Trees I

Node \( p \leftarrow \text{find} \; x \) by standard BST search
while \( (p \neq \text{root}) \) do

- \((p = \text{child of root}) \): \( \text{zig}(p) \)
- \(\ast\; p \) has grand parent \( \ast\)\:
  - \((p \text{ is LL or RR grand child}) \): \( \text{zig-zig}(p) \)
  - \((p \text{ is LR or RL grand child}) \): \( \text{zig-zag}(p) \)

Example:

- \( \text{splay}(3) \):
  - \( \text{LR zig-zag} \)
- \( \text{insert}(x) \):
  - \( \text{add node} \; q \leftarrow \text{splay}(x) \)
  - \( \text{if} \; (p, \text{key} = x) \) \text{Error!!}
  - \( q \leftarrow \text{new Node}(x) \)
  - \( \text{if} \; (p, \text{key} < x) \)
    - \( q, \text{left} = p, \text{right} = q, \text{right} = \text{p.right} \)
  - \( \text{else} \) \( \text{null} \)
  - \( \text{root} = q \) (symmetrical)

Subtrees \( A, C \) move up ↑

\( \text{Zig-zig}(p): \) [LL case]

\( \text{Zig-zag}(p): \) [LR case]

\( \text{Subtrees} \; C, E \; \text{of} \; p \; \text{move up} ↑ \)

\( \text{Zig}(p): \) [L case]

Subtree \( A \) moves up ↑

\( \text{C} \) unchanged
**Dynamic Finger Theorem:**

Keys: $x_1, \ldots, x_n$. We perform accesses $x_{i_1}, x_{i_2}, \ldots, x_{i_m}$.

Let $\Delta_j = i_j - i_{j-1}$, distance between consecutive items.

**Thm:** Total access time is $O(m + n \log n + \sum_{j=1}^{m} (1 + \log \Delta_j))$.

---

**Splay Trees III**

- **Static Optimality:**
  - Suppose key $x_i$ is accessed with prob $p_i$. ($\sum p_i = 1$)
  - Information Theory: Best possible binary search tree answers queries in expected time $O(H)$ where $H = \sum p_i \log \frac{1}{p_i}$ = Entropy

- **Static Optimality Theorem:**
  - Given a seq. of $m$ ops on splay tree with keys $x_1, \ldots, x_n$, where $x_i$ is accessed $q_i$ times. Let $p_i = q_i / m$. Then total time is $O(m \sum p_i \log \frac{1}{p_i})$.

---

**Analysis:**

- Amortized analysis
  - Any one op might take $O(n)$
  - Over a long sequence, average time is $O(\log n)$ each
  - Amortized analysis is based on a sophisticated potential argument

- Potential: A function of the tree's structure
  - Balanced $\Rightarrow$ Low potential
  - Unbalanced $\Rightarrow$ High potential
  - Every operation tends to reduce the potential

**Balance Theorem:** Starting with an empty dictionary, any sequence of $m$ accesses takes total time $O(m \log n + n \log n)$ where $n = \text{max. entries at any time}$.
What does entropy have to do with search times?

Keys: a b c d e f g

Access Probability: \[ \frac{1}{9} \frac{1}{9} \frac{1}{9} \frac{1}{9} \frac{1}{9} \frac{1}{2} \frac{1}{2} \Rightarrow \sum p_i = 1 \]

= \frac{1}{2^2} \frac{1}{2^4} \ldots \frac{1}{2^5} \rightarrow \text{The higher the exponent, the deeper the key should be.}

Depth:

- Extended Binary Search Tree (keys stored in leaves)
- "Ordered" Huffman tree

Entrophy:

\[ H = \sum p_i \lg \frac{1}{p_i} \]

The "ideal" depth is (access probability +1)