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
- Find("a")
- Tree restructures itself

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

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

Focus: Worst-case or randomized expected case

Splay Trees I

Final
- Tree's 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

Intuition: Let \( T \) be an unbalanced BST, suppose we access its deepest key
- Find("a")
- Ugh!
- Tree restructures itself

Amortized efficiency:
- Any single op - slow
- Long series - efficient on avg.
ZigZig(p): [LL case]

Splay(Key x):
Node p ← find x by standard BST search
while (p ≠ root) {
    if (p == child of root) zig(p)
    else /* p has grand parent */
        if (p is LL or RR grand child) zigzag(p)
        else /* p is LR or RL gr. child */ zigzag(p)
    } /* end while loop */

Subtrees A, C move up↑

ZigZag(p): [LR case]

Splay Trees II

Example:

Insert(x):
Node p ← splay(x)
if (p.key == x) Error!!
q ← newNode(x)
if (p.key < x)
p.left ← p
p.right ← q
else /* (symmetrical) */
    root ← q

Subtrees C, E of p move up↑

Zig(p): [L case]

Subtree A moves up↑ C unchanged

find(x):
root ← splay(x)
if (root.key == x) return root.value
else return null

Subtrees A, C move up↑
splay(X)

null because y is x's inorder successor

splay(x)

[259x359]splay(x) [x now at root]

p = root

if (p.key ≠ x) error!

splay(x) in p's right subtree

q = p.right [q's key is x's successor]

q.left = p.left

root = q

delete(x):

Dynamic Finger Theorem:

Keys: x₁, ..., xᵢ. We perform accesses xᵢ₁, xᵢ₂, ..., xᵢₘ.

Let Δᵢ = jᵢ₂ - jᵢ₁ = distance between consecutive items

Thm: Total access time is

O(m + n log n + ∑ᵢ (1 + log Δᵢ))

Static Optimality:

- Suppose key xᵢ is accessed with prob pᵢ. (∑ᵢ pᵢ = 1)

- Information Theory:

Best possible binary search tree answers queries in expected time O(H) where

H = ∑ᵢ pᵢ log ᵃᵢ = Entropy

Static Optimality Theorem:

Given a seq. of m ops. on splay tree with keys x₁, ..., xₙ, where

xᵢ is accessed qᵢ times. Let

pᵢ = qᵢ/m. Then total time is

O(m ∑ᵢ pᵢ log ᵃᵢ)

Splay Trees are Amazingly Adaptive!

Balance Theorem: Starting with an empty dictionary, any sequence of m accesses takes total time

O(m log n + m log n)

where n = max. entries at any time.

Splay Trees III

- 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 ⇒ Low potential
  Unbalanced ⇒ High potential

- Every operation tends to reduce the potential

Analysis:
Ideal Skip List:
- Organize list in levels
  - Level 0: Everything
  - Level 1: Every other
  - Level 2: Every fourth
  - Level 3: Every $2^i$

Sorted linked lists:
- Easy to code
- Easy to insert/delete
- Slow to search \( O(n) \)

Idea: Add extra links to skip

How to generalize?

Example:

Node Structure: (Variable sized)

```java
class SkipNode {
    Key key
    Value value
    SkipNode[] next
}
```

Value `find(Key x)`

```java
i = topmost level
SkipNode p = head
while (i > 0) {
    if (p.next[i].key < x) p = p.next[i]
    else i--
} // we are at base level
if (p.key == x) return p.value
else return null
```

Too rigid \( \rightarrow \) Randomize!

To determine level - toss a coin and count no. of consec. heads:

Example:

Bill Pugh

Your rose...

Kirill
Delete:
- Start at top
- Search each level saving last node < key
- On reaching node at level 0, remove it and unlink from saved pointers

Example: find(75)

Insert: (Similar to linked lists)
- Start at top level
- At each level:
  - Advance to last node ≤ key
  - Save node + drop level
- At level 0:
  - Create new node (flip coin to determine height)
  - Link into each saved node

Insert (24)

Obs: When you first hit a node, land on top level.

Delete (12)

Analysis: All operations run in time ~ find \(\Theta(\log n)\) expected

Note: Variation in running times due to randomness only—not sequence
\(\Rightarrow\) User cannot force poor performance.
Announcements 4/6

- Prog assignment 2
  - Deadline extended to Fri, 11:59pm

- Homework 3
  - Due Tue, Apr 11 in class

- Midterm 2
  - Thu, Apr 13
    - Closed book / closed notes
    - 2 cheat sheets / front + back
    - Coverage through this week - kd-trees, splay, scapegoat, skip-lists