Splay Trees
Self-adjusting data structure — after structure on a query.
  e.g. move to front heuristic for a list
Note — we will do work ahead of time in hopes of saving later
  — the opposite approach from procrastinating (as in lazy
    Binomial Heaps)
Recall Dictionary
  — keys from totally ordered universe
  — operations: Insert, Delete, Search

Binary Search Tree
  e.g. search for 4

  6
 / \
3  7
 / \ /
1  5 9

Insert — make new node where search fails
Delete — easy for node w/ 0 or 1 child
  — else replace value by predecessor or successor
    and delete that node (which has 0 or 1 child)
All operations take \( O(h) \) \( h = \) height of tree
Balance — keep \( h \in O(\log n) \)
AVL, red-black etc. [also 2-3 but they are not binary]
  technique for rebalancing

rotation

must keep balance info and update as you rotate
Splay Tree [Sleator & Tarjan 1985]
- $\Theta(\log n)$ amortized cost per operation, $\Theta(n)$ worst case
- Easier to implement than AVL/Red-Black trees
- Don't keep balance info - the tree may become unbalanced
- Careful! Repeated searching for a deep node costs $\Theta(n)$ each time
  Fix - adjust the tree whenever a node is accessed.
  Hence "self-adjusting".

First idea: do [single] rotations to move the searched node to root. [Show this gives amortized search cost $\Omega(n)$]

Splay operation

Splay ($x$)

Repeat til $x$ is root

Case 1 "zig-zag"

Case 2 "zig-zig"

Case 3 "zig" At child of root, do single rotation.
Splay Tree Algorithm

Search - after finding x call Splay (x)
Insert - do standard insert then Splay new node
Delete - do standard delete of key,
      splay parent of node that gets deleted
      (the node with 0 or 1 child )

Examples (next slide)

Amortized Analysis of Splay Trees
Use Potential Method (reminder on slide)

Plan - use 3 steps:
1. Define potential and
   analyze Z-operations (zig-zag, zig-zig, zig)
2. Analyze Splay
3. Analyze Search, Insert, Delete

Intuition: if node x is far from
          root Splay (x) is expensive but
          potential will pay for it.
          (potential will be higher for tree with large height)
- Search: Find the node containing the key using the usual algorithm, or its predecessor or successor if the key is not present. Splay whichever node was found.

- Insert: Insert a new node using the usual algorithm, then splay that node.

- Delete: Find the node $x$ to be deleted, splay it, and then delete it. This splits the tree into two subtrees, one with keys less than $x$, the other with keys bigger than $x$. Find the node $w$ in the left subtree with the largest key (the inorder predecessor of $x$ in the original tree), splay it, and finally join it to the right subtree.

- Splaying a node. Irrelevant subtrees are omitted for clarity.
Reminders (repeat from Lecture 3)

Definition sequence of \( m \) operations

- total cost is \( T(m) \)

- then the amortized cost per operation is \( T(m)/m \)

\[ \text{cost}(i) = \text{true cost of } i^{th} \text{ operation} \]

\[ \text{charge}(i) = \text{[artificial] charge for } i^{th} \text{ operation} \]

\( \Phi_i \) - potential after \( i^{th} \) operation.

\[
\Phi_i = \Phi_{i-1} + \text{charge}(i) - \text{cost}(i)
\]

\[
\text{charge}(i) = \text{cost}(i) + \frac{\Phi_i - \Phi_{i-1}}{\Delta \Phi_i}
\]

Thm. If final potential \( \geq \) initial potential then amortized cost \( \leq \max \text{ charge} \).
Define \( D(x) \) = # descendants of \( x \) (including \( x \))
\[
    r(x) = \log D(x)
\]
\[
    \Phi(t) = \sum_{x} r(x)
\]

**Ex.** What is max and min potential of tree of \( n \) nodes?

Path: \( r = \log n \)       perfect binary tree: \( \Phi = \frac{n \log n}{h+1} \) number of nodes at height \( h \)

\[
    \Phi = \sum_{i=1}^{n} \log i = O(n \log n)
\]

\[
    = n \sum_{h=1}^{\log n} \frac{n}{2^h} - \text{constant}
\]

\[
    = O(n)
\]

Analyze \( z \) operation at node \( x \):

\( r(x) \) - current rank \( r'(x) \) - new rank

**Claim** amortized cost of one \( z \) operation on \( x \) is

\[
    \leq 3(r'(x) - r(x))
\]

\( +1 \) for zig-zig, zig-zag

**Note** for bounding Splay: this sum will telescope!

**Proof**

zig.
(warm up)

\[
    charge = \text{cost} + \Delta \Phi
\]

change in potential
amortized cost = \[
\frac{1}{2} \cdot \text{cost of rotation}
\] + \[
\text{change in potential}
\]

\[
= 1 + (r'(y) - r(x)) \\
\leq 1 + (r'(x) - r(x)) = 1 + 3(r'(x) - r(x))
\]

because y is child of x

\[
z'g - z'g
\]

D(x) + D'(z) \leq D'(x)

amortized cost = \[
\frac{2}{2} + \text{change in potential}
\]

\[
= 2 + (r'(x) + r'(y) - r(x) - r(y) - r(z)) \\
\leq 2 + r'(x) - 2r(x)
\]

We want this \leq 3(r'(x) - r(x))

i.e. want \[2 + r(x) + r'(z) \leq 2r'(x)\]

\[
\log D(x) + \log D'(z) \leq 2 \log D'(x) - 2
\]

given \[D(x) + D'(z) \leq D'(x)\]

For this, we need a property of logs
Fact: by convexity of log
if \( x, y > 0 \) \( x + y \leq 1 \) then the log fn \( \log \frac{x}{y} + \log \frac{y}{x} \) is
maxd at value -2 when \( x = y = \frac{1}{2} \)

Hence, for \( a, b > 0 \) \( a + b \leq c \)
\[
\log \frac{a}{c} + \log \frac{b}{c} \leq -2
\]
\( \log a + \log b \leq 2 \log c - 2 \)

Now \( D(x) + D'(z) \leq D'(x) \)
\( \log D(x) + \log D'(x) \leq 2 \log D'(x) - 2 \)
i.e. \( r(x) + r'(x) \leq 2 r'(x) - 2 \) which gives

Zig-Zag - similar (needs same log trick)

Part 2 Analyze Splay \((x)\)

Lemma Tree \( T \), root \( t \), node \( x \)

amortized cost of \( \text{splay}(x) \) is
\[
\leq 3 (r(t) - r(x)) + 1 \leq 3r(t) + 1 \in O(\log n)
\]

pf. Add up the amortized cost of each step of Splay - telescoping sum.

Let \( r_i = r(x) \) after \( i \)th step of splay. So \( r_0 = r(x) \) \( r_k = r(t) \)
where \( k \) is last step.

amortized cost is
\[
\sum_{i=0}^{k} (3(r_i - r_{i-1}) + 1)
\]
by claim above.
The amortized cost of Insert, Search, Delete in splay tree is $O(\log n)$.

Proof: \[
\text{charge} = \underbrace{\text{charge(splay)}} + \text{cost(operation)} + \Delta \Phi
\]

$O(\log n)$ by Lemma above.

Cost of each operation (walk down tree) $\leq$ cost of ensuing splay.

Only thing we haven’t taken into account is change in potential due to changes outside splay: Search - only splay changes potential.

Delete - removing a node decreases potential.

Insert - adding new node $v$ increases ranks of all ancestors of $v$ (might be $n$ of them).

\[
\begin{align*}
D'(v_i) &= 1 + D(v_i) \\ r'(v_i) &\leq r(v_{i+1}) \\
\text{change in potential is} &\ \sum_{i=0}^{k} r'(v_i) - \sum_{i=1}^{k} r(v_i) \\
&\leq r'(v_k) \leq \log n
\end{align*}
\]
Dynamic Optimality Conjecture

Conjecture [Sleator & Tarjan '85] Splay trees are optimal (within a constant) in a very strong sense:

Given a sequence of items to search for:

\[ x_1, x_2, \ldots, x_m \]

Let \( \text{OPT} \) be \( \min \) cost of doing these searches + any rotations you like on BST.

Charge 1 for following tree pointer (parent \( \rightarrow \) child or child \( \rightarrow \) parent)
Charge 1 for rotation

Conj: cost of splay tree is \( \Theta(\text{OPT}) \).

Note that for \( \text{OPT} \), you get to look at the seq. of searches first and plan ahead. "offline", in contrast with "online" which is usual for data structures.

We will cover online algorithms later in the course.

Note 2: \( \text{OPT} \) can adjust the tree so it's even better than the static optimal binary search trees you may have seen in CS 341.