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 hope 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

Insert — make new node where search fails
Delete — easy for node w/ 0 or 1 child
- else replace value by predecessor or successor

All operations take $O(h)$ $h =$ height of tree

Balance — $h \in O(\log n)$

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 $O(n)$]

Splay operation

Splay ($x$)

repeat til $x$ is root

Case 1 "zig-zag"

Note that this is 2 rotations on $x$

Case 2 "zig-zig"

Note: rotate $y$ then rotate $x$

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 then Splay parent of node that gets deleted (w 0 or 1 child)

Examples (next slide)

Amortized Analysis of Splay Trees

Plan - use 3 steps:

① Analyze Z-operations (zig-zag, zig-zig, zig)
② Analyze Splay (Ge)
③ Analyze Search, Insert, Delete

Intuition for ②: if node x is far from root, Splay(x) is expensive but Potential will pay for it.
(Potential is higher for a 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.

\[ \text{Figures 2, 3, 4, 5} \]

from Jeff Erickson's notes
Define \( D(x) = \# \text{ descendants of } x \) (including \( x \))
\[ r(x) = \log D(x) \]
\[ \Phi(x) = \sum_{x} r(x) \]

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

Path \( r = \log n \)

\[ \log(n-1) \]

\[ \log 1 \]

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

\[ \Phi = \frac{n}{2} \cdot \log \left( \frac{n}{2} \right) - \text{constant} \]

\[ = O(n) \]

1. Analyze \( z \) operation at node \( x \):
   - \( r(x) \) — current rank
   - \( r'(x) \) — new rank

   **Claim** amortized cost of one \( z \) operation on \( x \) is
 \[ \leq 3 (r'(x) - r(x)) \]

   - for zig-zig, zig-zag
   - +1 for zig

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

**Proof**

- Zig (warm up)

\[ \begin{array}{c}
\text{zig} \\
\text{(warm up)}
\end{array} \]

\[ \begin{array}{c}
x \\
\downarrow \\
1 \\
\downarrow \\
2 \\
\downarrow \\
3 \\
\end{array} \]

\[ \begin{array}{c}
y \\
\downarrow \\
1 \\
\downarrow \\
2 \\
\downarrow \\
3 \\
\end{array} \]

\[ \begin{array}{c}
x \\
\downarrow \\
1 \\
\downarrow \\
2 \\
\downarrow \\
3 \\
\end{array} \]

\[ \text{since } D'(x) = D(y) \]

\[ r'(x) = r(y) \]

Recall \( \text{charge} = \text{cost} + \frac{\Delta \Phi}{\text{change in potential}} \)
amortized cost = 1 + \[ r'(x) + r'(y) - r(x) - r(y) \]

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

because y is child of x

zig-zig

\[ \frac{r'(x)}{2} - \frac{r'(z)}{2} \]

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

amortized cost = \[ \frac{2}{2 \text{ rotations}} + \frac{r'(x) + r'(y) + r'(z)}{2} - r(x) - r(y) - r(z) \]

cancel

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

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

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

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

i.e., \[ \log D(x) + \log D'(z) \leq 2 \log D'(x) - 2 \]

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 \( \log x + \log y \) is
max'd 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 \( D(x) + D'(x) \leq D'(x) \)

so \( \log D(x) + \log D'(x) \leq 2 \log D'(x) - 2 \)

i.e., \( r(x) + r'(x) \leq 2 r'(x) - 2 \) which gives \( \Omega \)

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

\[ \leq 3 (r(t) - r(x)) + 1 \leq 3 r(t) + 1 \in O(\log n) \]

Pf. Add up the amortized cost of each step of \( \text{splay} \) -

telelescoping sum:

Let \( r_i = r(x) \) after ith step of \( \text{splay} \). So \( r_0 = r(x) \), \( r_k = r(t) \)

where \( k \) is last step.

amortized cost is \( \sum_{i=1}^{k} 3(r_i - r_{i-1}) + 1 \) by claim above.

= \( 3(r_k - r_0) + 1 \)
Amortized cost of Insert, Search, Delete in splay tree is $O(\log n)$.

**Proof**

\[
\text{charge} = \text{charge (splay)} + \frac{\text{cost + } \Delta \Phi}{O(\log n)} \quad \text{by Lemma above}
\]

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)

\[
D'(v_i) = D(v_i) + 1 \quad D'(v_{i+1}) \leq D(v_{i+1})
\]

\[
r'(v_i) \leq r(v_{i+1})
\]

Change in potential is

\[
\sum_{i=0}^{k} r'(v_i) - \sum_{i=1}^{k} r(v_i)
\]

\[
\leq r'(v_k) \leq \log n
\]
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 \( O(\text{OPT}) \). 

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 CS341.