Advanced Data Structures

Splay Trees
Recap

- Sub to finish next Friday
- HW due Friday

Questions?
Today: Back to binary trees

Rotations: (Single rotation)

Note: If in an unbalanced tree, "rotate up to root" can take $O(n)$ time

Next: 2 kinds of double rotations
01  Roller coaster  "zig zag"
(Just what it sounds like)

Z
→
Y
→
X

T1 T2

T3 T4

How: rotate (y)
rotate (x)

Why?

rotate (y)

both same direction
2. Zig-Zag:

How:
- rotate \((x)\)

Why?
- rotate \((x)\)
Note: Each double rotation
- affects \( x \)'s depth: \(-2\)
- \( x \)'s parent's depth (\( y \) or \( w \)) unchanged
- \( x \)'s grandparent: \( z \)
  \( +1 \) or \(+2\)

Runtime: \( O(1) \)
\( \leq 14 \) pointer updates
(plus ifs to check cases)
Splay \((x)\):

while \((x \neq \text{root})\) or \((\text{parent}(x) \neq \text{root})\)

double rotation \((x)\)

If \(x \neq \text{root}\)

rotate \((x)\)

Runtime: \(\frac{1}{2} \cdot \text{depth}(x) \cdot \mathcal{O}(1)\)

= \(\mathcal{O}(\text{depth}(x))\)

(Data structure doesn't track height/depth)
Splay Tree

A (more or less) balanced binary tree where we splay to balance (* mostly!)

High level idea:
Any time a node is accessed (search/insert/delete), splay it to the root.

Why??

Amortization!
If you splay, other things balance—works out to $O(\log n)$ amortized
time per operation.
More concretely:

**Search** \( (x) \):

\[
\text{node} \leftarrow \text{BSTFind} (x) \\
(\text{assume this returns } x, \text{ or pred/succ if } x \text{ is not in tree}) \\
\text{splay} (\text{node})
\]

**Insert** \( (x) \):

\[
\text{node} \leftarrow \text{BSTinsert} (x) \\
(\text{assume this returns } x's \text{ node in tree}) \\
\text{splay} (\text{node})
\]
Delete \( (x) \):
\[
\text{xnode} \leftarrow \text{BSTFind} (x)
\]
if \( \text{xnode}.value = x \):
\[
\text{splay} (\text{xnode})
\]
left \leftarrow (\text{xnode}.left) 
right \leftarrow (\text{xnode}.right) 
\text{delete} (\text{xnode}) 
\]
\[
\text{l} \leftarrow \text{FindLargest} (\text{left})
\]
\[
\text{splay} (\text{l})
\]
\[
l.left \rightarrow \text{right}
\]
Note: Each of these has a constant # of the following:
- walk down to some node
- splay that node to root

Cost (walk) ≤ Cost (splay)

Why? 1 per level ≤ 20 ops per level

= \( O\left(\text{depth}(T)\right) \)
What does it cost to splay?

Worst case: $O(n)$

To get amortized, need a potential function:

Let $w(v) =$ weight of $v$'s subtree

$$S(v) = w(v) + S(v_{left}) + S(v_{right})$$

Set $S(null) = 0$

Let $rank(v) = g(S(v))$

Note: if $S(v) = 1$ for all $v$. 

Diagram of a splay tree.
Potential function:
\[ \Phi(T) = \sum_{v \in T} r(v) \]

\[ = \sum_{v \in T} \log(s(v)) \]

(Useful later...)

Amortized cost:
\[ \text{time} + \Phi' - \Phi \]
Access Lemma:

Amortized time to splay a binary tree $T$ with root $t$ at $x$ is

$$\leq 3(r(t) - r(x)) + 1$$

$$= O\left(\frac{\log s(t)}{\log s(x)}\right)$$

Restate:

Let $r(x) =$ rank before splay

$r'(x) =$ rank after splay

Rotation:

Single

double