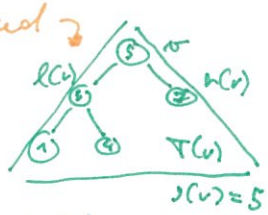
Lazy balanced BSTs – example for amortized complexity

idea: let the structure degrade, but rebuild it (locally) occasionally

notation: v node in BST — not perfectly balanced, balanced?



$T(v)$ ... subtree rooted at v (including v itself)

$s(v)$ ... size of v = # vertices in $T(v)$

$l(v), r(v)$ ... left, right child of v ($\emptyset$ if child missing)

Def: A BST is perfectly balanced if $|s(l(v)) - s(r(v))| \le 1$ for every v.

1:1

: sorted n items $\Rightarrow$ perfectly balanced BST in $\Theta(n)$ time.

Def: A node v is in balance if $s(c) \le 2/3 \cdot s(v)$ for each of its child c.

1:2 or 2:1

A BST is balanced if all its nodes are in balance.

Lemma: The height of a balanced tree with n nodes is $O(\log n)$.

Pf  $1 \le \left(\frac{2}{3}\right)^h n \Rightarrow h \le \log_{2/3}(1/n) = \log_{3/2} n = O(\log n)$

INSERT (lazy): update $s(v)$ in each v to root

if all in balance, end

else REBUILD $(T(v))$ at highest out-of-balance v

Thm: INSERT works in $O(\log n)$ amortized time.

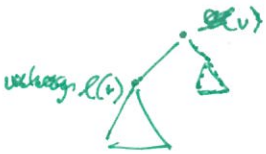
Pf: potential $\Phi := \sum_v \varphi(v)$ ← how far from perfectly balanced tree

$$\varphi(v) := \begin{cases} |s(l(v)) - s(r(v))| & \text{if at least 2} \\ 0 & \text{otherwise} \end{cases}$$

← so that p.b.t. has $\Phi = 0$.

a) no rebuild: amortized (real) = $O(\log n)$ + $O(\log n)$ ($\Delta\Phi$ (at most 2 each $\varphi(v)$ by ≤ 2)) $= O(\log n)$

b) rebuild at v



w.l.o.g. $s(l(v)) > 2/3\,s(v)$ $\Rightarrow$ $s(r(v)) < 1/3 \cdot s(v)$

$\Rightarrow$ $\varphi(v) \ge 1/3\, s(v)$ contribution of v to Φ

↳ drops to 0 in Φ $\Rightarrow$ $\Delta\Phi \le -\frac{1}{3}s(v)$

amortized cost = $\Theta(s(v))$ + $c \cdot \Delta\Phi \le 0$ for a suitable const. c.

of rebuild (real cost)

$\Rightarrow$ $O(\log n)$ amortized time for INSERT. □

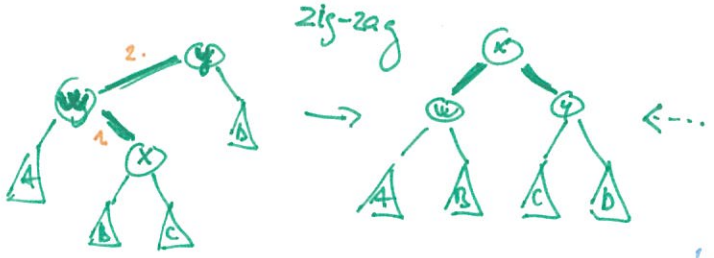
Splay trees

self-balancing BST with $O(\log n)$ amortized costs

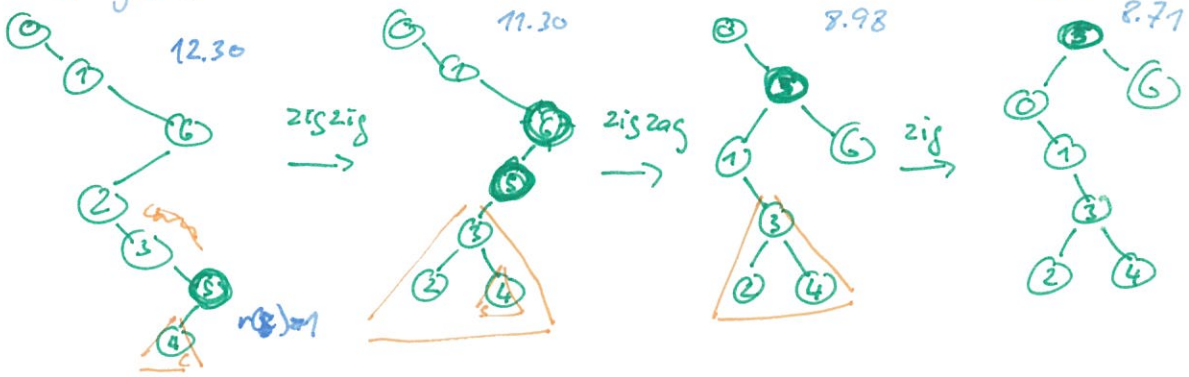
SPLAY(x) ... bring x to the root, preferring double rotations:

single rotation (of edge)

double rotations:



Ex. splay(5)



Notation: v ... node, n ... # nodes

$T(v)$ ... subtree rooted at v

$s(v) = |T(v)|$ (size) $\qquad 1 \le s(v) \le n$

$r(v) = \log_2 s(v)$ (rank) $\qquad 0 \le r(v) \le \log n$

$\Phi = \sum_v r(v)$ (potential) $\qquad 0 \le \Phi \le n \log n$

real cost of SPLAY(x) = # rotations (1 for zig, 2 for zigzig, zigzag)

Theorem: The amortized cost of SPLAY(x) is at most $3(r'(x) - r(x)) + 1$

where $r(x)$ and $r'(x)$ are ranks of x before and after.

Corollary: The amortized cost of SPLAY(x) is $O(\log n)$.

Corollary: A sequence of m SPLAYs runs in $O((n+m)\log n)$ time. (real cost = amortized cost + $\Delta\Phi$, drop of the potential $\leq n \log n$)

Operations in splay trees

Idea: always splay the lowest visited node

FIND (x) -- searching (real cost $\Theta(d)$) → worst-case $\Delta\Phi = 0$; d = depth where we stopped
SPLAY real cost $\Theta(d)$

amortized $O(\log n)$ for SPLAY $\Rightarrow$ amortized $O(\log n)$ FIND
(search accounted to splay, potential multiplied by some constant)

INSERT -- FIND x, adding a leaf (if unsuccessful) + SPLAY

$O(1)$ real cost but increases potential!

Lemma: Adding a leaf increases potential by $O(\log n)$.

$$\Delta\Phi = r'(v_{t+1}) + \sum_{i=1}^{t}\left(r'(v_i) - r(v_i)\right) \leq r'(v_1) - r(v_t) = O(\log n)$$

($r'(v_{t+1}) = 0$; telescoping)

$$r'(v_i) = \log(s(v_i)+1) \leq \log(s(v_{i-1}) = r'(v_{i-1}) \quad \forall i > 1 \qquad \square$$

$\Rightarrow$ INSERT $O(\log n)$ amortized cost

Lecture 3

DELETE ... FIND x, remove leaf / internal 1 child / min. in right subtree

decreases Φ

+ SPLAY (parent of removed node)

again, costs are accounted (offline) to SPLAY

$\Rightarrow$ DELETE $O(\log n)$ amortized cost

Theorem: A sequence of m operations FIND, INSERT, DELETE on initially empty splay trees takes time $O(m \log n)$ where n is the max. number of nodes during the sequence.

Pf: $\sum h_i = \sum A_i - \Delta\Phi = O(m \log n) - \Phi_m^{\geq 0} + \Phi_0^{\geq 0}$. $\square$