Union-Find DS, Binary Search Trees

$S_1, \dots, S_k \subseteq [n]$

Def: Union-Find DS represents a partition of $[n]$ into sets

Find(x) for $x \in S_j$ will return a representative of S_j

$S_j = \{a, b, c\}$ $\quad$ Find(a) = Find(b) = Find(c)

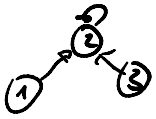
Union(x,y) for $x \in S_i$, $y \in S_j$ modifies the partition by replacing S_i and S_j with $S_i \cup S_j$

Implementation with trees:

- Begin with $S_1 = \{1\}, S_2 = \{2\}, \dots S_n = \{n\}$

Union(1,2)

Union(1,3)



Find(x): return Find(parent(x)) (+base case)

∴ complexity of Find is $O(\text{depth})$ $\quad$ depth = max depth of all the trees

Rank(x) = depth of subtree rooted at x

Union(x,y):
- r_x = Find(x) ... root of tree containing x
- r_y = Find(y)

$O(\text{depth})$

if rank(r_x) > rank(r_y):

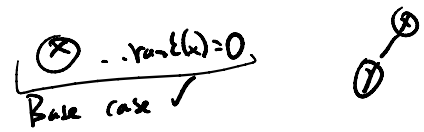
$O(1)$ parent(y) = r_x

else: parent(x) = r_y

Lemma: depth is $O(\log n)$

Proof: by induction, prove that if rank(x) = k, then subtree of x contains at least 2^k vertices

Subtree of x contains at least $2^{rank(x)}$ vertices

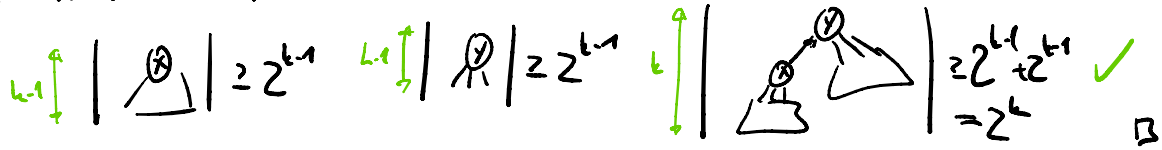


Inductive case: Union(x,y) .. By IH subtree of x has $\geq 2^{rank(x)}$ vert.
—"— y —"— $2^{rank(y)}$ verts.

If rank(y) < rank(x) →  → rank(x) did not increase, but #vertices in its subtree increased → good

if rank(y) > rank(x) → symmetric ✓

if rank(x) = rank(y) = k-1



✓ □

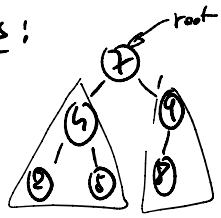
Getting the lemma: no tree has more than n vertices, so what is its rank?

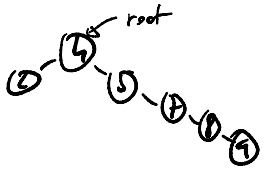
$k \to 2^k = n$ / log

$k = \log n$

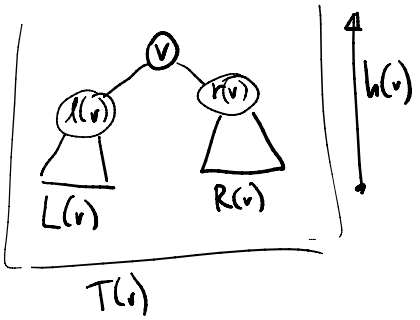
Binary Search Trees

Examples:





≤ 2 children
- ordered (left / right)



$L(v) = T(l(v))$

Definition: A Binary Search Tree (BST) is a binary tree where we associate to each vertex a unique key $k(v)$, and:
- if $a \in L(v)$ then $k(a) < k(v)$
- if $b \in R(v)$ then $k(b) > k(v)$

BSTShow(v):
if v = ∅ → return None
BSTShow(l(v))

BSTFind, BSTMin (go left as long as you can)

if $v = \emptyset$ → return nothing

BSTShow($\ell(v)$)

print($k(v)$)

BSTShow($r(v)$)

BSTInsert(v, x) — v: root of the BST; x: inserted value

(like Find, but instead of not finding the key we insert where it should've been

---

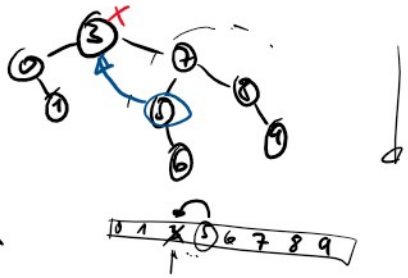
BSTDelete(v, x) ... three cases:

- x is a leaf → trivial, just delete
- x has one child

- x has two children

Deleting x with two children

- Find succ(x) ... next value after x
  - go to $R(x)$, keep going left as long as possible, when you cannot continue, you found successor of x
- succ(x) doesn't have a left child
  - ⇝ succ(x) has ≤ 1 child



Then: – replace x with succ(x)

– delete succ(x) – we already know how

BSTShow ... $\Theta(n)$

BSTFind/Insert/Delete ... $\Theta(\text{depth})$

Def: BST is perfectly balanced if $\forall v \in V: \big||L(v)| - |R(v)|\big| \leq 1$.

Obs: A perfectly balanced BST has depth $\lfloor \log_2 n \rfloor$

Theorem: For any implementation of Insert and Delete which maintains a perfectly balanced BST, it holds that Insert and Delete take $\Omega(n)$ time for infinitely many values of n,

Takeaway: Maintaining a perfectly balanced BST is too costly.

Depth-balanced trees: AVL trees

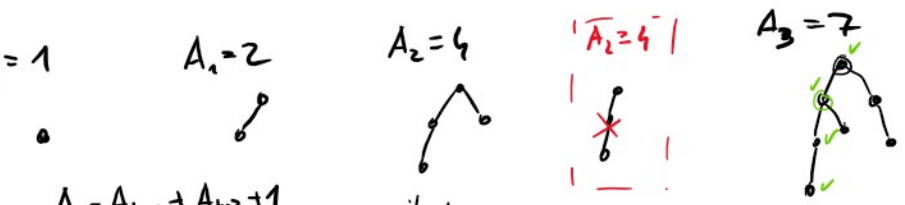
Def: A BST is depth-balanced (AVL) if $\forall v \in V: \big|h(\ell(v)) - h(r(v))\big| \leq 1$ (depth/height)

Def: A BST is depth-balanced if $\forall v \in V: |h(l(v)) - h(r(v))| \leq 1$ — "AVL"

Lemma: An AVL tree on n vertices has depth $\Theta(\log n)$

Proof: For $h \in \mathbb{N}$, let A_h be the smallest number of vertices of an AVL tree of height h.

$A_0 = 1$ $A_1 = 2$ $A_2 = 4$ $A_2 = 4$ $A_3 = 7$



$A_h = A_{h-1} + A_{h-2} + 1$ ... it holds that $A_h = F_{h+3} - 1$ ⇝ A_h grows exponentially

Instead, prove by induction that $A_h \geq 2^{h/2}$. $A_0 \geq 2^{0/2} = 2^0 = 1$ ✓ (base case)

$2 = A_1 \geq 2^{1/2} = \sqrt{2} \doteq 1.414$ ✓

Inductive step: $A_h = 1 + A_{h-1} + A_{h-2} \geq 2^{\frac{h-1}{2}} + 2^{\frac{h-2}{2}} = 2^{\frac{h}{2}}\left(2^{-\frac{1}{2}} + 2^{-1}\right) \geq 2^{\frac{h}{2}} \cdot 1.2 > 2^{\frac{h}{2}}$

(+1 dropped because I put 1 on rhs)

⇝ $A_h \geq c^h$ for $c = \sqrt{2}$

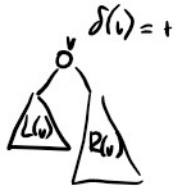
→ If an AVL tree has n vertices, it can have at most depth $\log_c n$ (otherwise it would contain more than $c^{\log_c n} = n$ vertices)

Maintaining AVL trees:

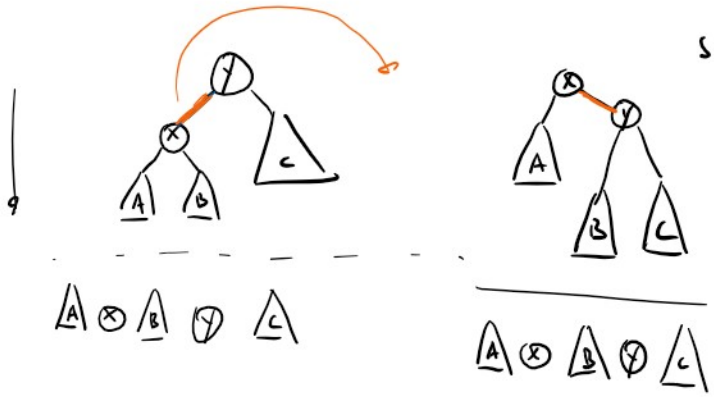
- each node gets a sign (signifying how balanced it is)

$\delta(v) = h(r(v)) - h(l(v))$

→ in an AVL tree $\delta(v) \in \{-1, 0, +1\}$ — $-$ / 0 / $+$



Single rotation



A X B Y C

A X B Y C

double rotation