1 AVL: Adelson-Velskii, Landis

For all nodes `z`,

$$|\ \texttt{height(z.left)} - \texttt{height(z.right)}\ | \leq 1$$

Then,

$$\text{height} \in O(\log(n))$$

Assume: every node stores **height of its subtree** (and update it doing `insert`)

1.1 AVL-tree: Insert

- Insert as for a BST
- Go back to the root, update heights and check balances.
- If some node z is now unbalanced, let y and x be the descendants on insert-path apply an appropriate rotation at z, y, x. (Bring the middle-value of $\{x, y, z\}$ to the root.)

Runtime: $O(\log(n))$. Constant time per rotations. We have at most $O(\log(n))$ rotations, namely, one per level. It is true that there are at most $O(\log(n))$ rotations, but actually one rotation is enough!

Claim: If AVL-tree after insert is unbalanced at z, then this rotation makes subtree of z balanced and **reduces its height**.

Summary: AVL-trees have worst-case height $O(\log(n))$. Insert takes worst-case time $O(\log(n))$, and one rotation. Delete takes worst-case time $O(\log(n))$, but possibly many($\leq \log(n)$) rotations.

But not good in practice!

2 Scapegoat(α)-Tree

Idea: BST with $O(\log(n))$ amortized insert-tree that use no rotations.

α is a constant such that $\frac{1}{2} < \alpha < 1$. (Choose α initially, and keep it fixed.)

Structural Property: Height is at most $\log_{1/\alpha}(n)$.

To achieve this during insert, every node v stores $n_v = \#$ of nodes in subtree of v.

Insert:

- Insert as for a BST-tree.
- On insert-path, compute heights.
- On each node of the insert path, deposit a "token".
- If now the height is $> \log_{1/\alpha}(n)$, completely rebuild same subtree of the tree.

Completely Rebuild at p:

- Extract the descendants of p in *inorder* (sorted)
- Build them into a perfectly balanced tree.

  (Perfectly Balanced: |size(left) - size(right)| ≤ 1.)
- This can be done in $O(n_p)$ time.
- Remove all tokens at p (and all its descendants).

Claim: If the insertion path has height $> \log_{1/\alpha}(n)$, then some node v on that path satisfies $n_v > \alpha \cdot n_{\texttt{parent(v)}}$.

Proof. We prove the contrapositive.

Assume $n_v \leq \alpha \cdot n_{\texttt{parent(v)}}$ for all v.

- $n_{\texttt{root}} = n$
- $n_{\texttt{child(root)}} < \alpha \cdot n$
- $n_{\texttt{grandchild(root)}} \leq \alpha^2$
- $\cdots$
- $1 \leq n_{\texttt{leaf}} \leq \alpha^h \cdot n$

So,

$$\begin{aligned} 1 = n_{\texttt{leaf}} &\leq \alpha^h \cdot n \\ \left(\frac{1}{\alpha}\right)^h &\leq n \\ h &\leq \log_{1/\alpha}(n) \end{aligned}$$

□

Back to insert: Let v be the highest node with $n_v > n_{\texttt{parent(v)}}$. Rebuild the tree at `p:=parent(v)`. Afterwards, $n_z \leq \alpha \cdot n_{\texttt{parent(z)}}$ for all nodes on path.

2.1 Amortized Analysis

Claim: The amortized tree for insert is $O(\log(n))$.

Proof. Assume constant c so big that

- rebuild takes time $\leq c \cdot n_p$,
- the height is $\leq c \cdot \log(n)$,
- insert without rebuild takes time $\leq c \cdot \log(n)$

Set $K = \frac{c}{2\alpha - 1}$. ($c, K$ are constants)

Define

$$\Phi(t) := K \cdot \ \#\text{ “tokens” at time t}$$

Insert without rebuilding

Amortized time of insert without rebuilding

$$\leq c \cdot \log(n) + K \cdot \underbrace{c \cdot \log(n)}_{\#\text{ of new tokens (=height)}} \in O(\log(n))$$

Insert with rebuilding

$$\begin{aligned} &= c \cdot \log(n) + \underbrace{c \cdot n_p}_{\text{rebuild}} + K \cdot c \cdot \log(n) - K \cdot (\#\text{ tokens at } p) \\ &= O(\log(n)) + c \cdot n_p - \underbrace{c \cdot \frac{1}{2\alpha - 1}}_{=K} (\#\text{tokens at } p) \end{aligned}$$

Claim: # of tokens at p is $\geq (2\alpha - 1) \cdot n_p$.

□