Balanced tree: depth is $O(\lg n)$

Want update time as well as search time to be $O(\lg n)$.

Can’t keep all leaves within 1 in depth. Need more flexibility.

How to define balance?

How to restore balance after an insertion or deletion?
Restructuring primitive: \textit{Rotation}

Preserves symmetric order (searchability).
Changes some depths.
Complete: can transform any tree into any other tree on the same set of items.
Local: takes $O(1)$ time.
rotate at x

rotate at y

right

left
Balance

Each node \( x \) has an integer rank \( r(x) \). \( r(\text{null}) = -1 \).

Rank is a proxy for height.

*rank difference* of a child \( x \): \( \Delta r(x) = r(p(x)) - r(x) \)

**Balance: restriction on rank differences**

Notation: *i-child*: rank difference is \( i \).

*node is i,j*: rank differences of children are \( i, j \) (order unimportant)
**AVL trees** (Adelson-Velsky and Landis 1962): nodes are 1,1 or 1,2 (*not original dfn., but equivalent*)

\[ h(x) = r(x) \]

**Red-black trees** (Bayer 1972 via Guibas and Sedgewick 1978): nodes are 1,1 or 0,1 or 0,0; leaves are 1,1; if \( x \) is a 0-child, \( p(x) \) is not a 0-child (0-children *red*, other nodes *black*)

\[ r(x) \leq h(x) \leq 2r(x) + 1 \]

**Left-leaning red-black trees** (Bayer 1971 via Andersson 1993): red-black and each 0-child is a left child (no 0,0 nodes)

\[ r(x) \leq h(x) \leq 2r(x) + 1 \]
Rank-balanced trees (Sen and Tarjan 2009):
Δr’s are 1 or 2; leaves are 1,1
→ \( r(x)/2 \leq h(x) \leq r(x) \)

Relaxed AVL (ravl) trees (Sen and Tarjan 2010):
Δr’s are positive
→ \( h(x) \leq r(x) \)

Many others, notably weight-balanced trees:
balance given by size ratio not rank difference.
AVL Trees

Each node is 1,1 or 1,2

→ $r(x) = h(x)$, $r(\text{leaf}) = 0$, $r(\text{unary node}) = 1$

Rank differences stored, not ranks: one bit per node, indicating whether $\Delta r$ is 1 or 2. $r$’s are computable from $\Delta r$’s.

(vs original representation: store one of three states in each node: both subtrees have equal height, left subtree is higher by 1, or right subtree is higher by 1)
An AVL Tree. Numbers left of nodes are $r$’s (not stored). Numbers right of nodes are $\Delta r$’s (stored). Null nodes have $r = -1$ and $\Delta r = 1$ or 2 ($r$ and $\Delta r$ shown for two null nodes)
AVL-tree height bound

Fibonacci numbers

\[ F_0 = 0, \ F_1 = 1, \ F_k = F_{k-1} + F_{k-2} \text{ for } k > 1 \]
\[ 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, \ldots \]

Golden ratio \( \varphi = (1 + \sqrt{5})/2 \)

\[ \varphi^2 = \varphi + 1 \]
\[ \varphi^k \leq F_{k+2} \]
For any node \( x \), \( s(x) + 1 \geq F_{r(x)} + 3 \)

**Proof** by induction on \( r(x) \):

- \( s(\text{null}) + 1 = 0 + 1 = F_2 \)
- \( s(\text{leaf}) + 1 = 1 + 1 \geq F_3 \)

\( r(x) > 0: s(x) + 1 = s(\text{left}(x)) + 1 + s(\text{right}(x)) + 1 \geq F_{r(x)} + 2 + F_{r(x)} + 1 = F_{r(x)} + 3 \) since \( x \) is 1,1 or 1,2

\[
n + 1 \geq F_{h+3} \geq \varphi^{h+1} \rightarrow h \leq \log_\varphi(n + 1) - 1 \leq \log_\varphi n < 1.44043 \log n
\]
Balanced tree insertion

**Bottom-up rebalancing**: after an insert, restore balance by walking back up the search path, doing rank changes and rotations as needed.

**Top-down rebalancing**: restore balance top-down as search proceeds. Only works for certain definitions of balance (red-black, rank-balanced): needs extra flexibility.

**AVL trees**: bottom-up rebalancing after an insertion takes $\leq 2$ rotations.
AVL-tree insertion

Give new node $x$ a rank of 0. $\Delta r(x) = 0$ (bad) or 1.

To restore balance:

**while** $x$ is a 0-child whose sibling is a 1-child **do**

$x \leftarrow p(x); \ r(x) \leftarrow r(x) + 1$

(Increase of $r(x)$ changes $x$ from 0,1 to 1,2 but may make $x$ a 0-child.)

**if** $x$ is a 0-child whose sibling is not a 1-child **then**

apply the appropriate one of the following two transformations (one or two rotations and some rank changes):
numbers are $\Delta r$'s
also two mirror-image cases
Insert A

numbers are Δr’s
Insert D
AVL-tree insertion: $\leq 2$ rotations per insertion, worst-case.

What about promotions (rank increases)? $O(\lg n)$ worst-case but $O(1)$ amortized:

$\Phi = \#1,1$-nodes + $\#0,1$-nodes

Creation of a new node increases $\Phi$ by 1; each promotion decreases $\Phi$ by 1; last step increases $\Phi$ by at most 2: $\leq 3$ promotions per insertion, amortized.
1,1-nodes are like 1 bits in binary addition:
   a 1 bit can cause a carry but becomes a 0;
   a 1,1-node can be promoted but becomes a 1,2-node: 1,1 \rightarrow 0,1 \rightarrow 1,2

Giving potential to bad nodes (0,1) as well as good ones clarifies the analysis.
Balanced tree deletion

Like insertion: rebalance either bottom-up after node deletion, or top-down during search (not always possible).

Generally more cases than insertion:
AVL trees: 8 cases (vs 6 for insert),
\[\Omega(\lg n)\] rotations (vs \(\leq 2\) for insert).
Rank-balanced trees

All nodes are 1,1, 1,2, or non-leaf 2,2

For any node \( x \), \( s(x) + 1 \geq 2^{r(x)/2} + 1 \)

**Proof** by induction on \( r(x) \):

- \( s(\text{leaf}) + 1 = 2; s(\text{unary}) + 1 = 3 > 2^{3/2} \)
- if \( x \) binary, \( s(x) + 1 = s(\text{left}(x)) + 1 + s(\text{right}(x)) + 1 \)
  \[ \geq 2^{r(x)/2} + 2^{r(x)/2} = 2^{r(x)/2} + 1 \] since
  - \( x \) is 1,1 or 1,2 or 2,2 (2,2 worst)

\[ n + 1 \geq 2^{h/2} + 1 \rightarrow \]

\[ h \leq 2\lg(n + 1) - 1 \leq 2\lg n \] (vs. 1.44043\lg n for AVL trees)
**Insertion**: same bottom-up rebalancing algorithm as AVL trees: no 2,2’s created, but one can be destroyed.

**Deletion** of $x$ (bottom-up): If $x$ binary, swap with successor. Let $y = \rho(x)$ (unless $x$ is root). If $x$ now leaf, delete $x$ and reduce $r(y)$ by one (*demote* $y$); otherwise ($x$ unary), replace $x$ by its child. Now $y$ may be a 3-child ($\Delta r$ too big).
To restore balance:

**while** *y* is a 3-child with sibling *z* a 2-child or 2,2

**do** { if *z* not a 2-child then *r(z)* ← *r(z) − 1*;

\[
y ← p(y);
\]

\[
r(y) ← r(y) − 1
\]

(one or two demotions; new *y* may be a 3-child)

**if** *y* is a 3-child with sibling not a 2-child and not 2,2 **then** apply the appropriate one of the following two transformations (one or two rotations and some rank changes):
single

rotate

numbers are $\Delta r$'s

also two mirror-image cases

double

rotate
4 cases for deletion including 2 non-terminating
demotion cases (rank decreases) vs. 3 for
insertion (×2 for mirror-image cases = 8 vs. 6
for insertion)

At most 2 rotations, worst-case

Number of promotions/demotions per
insertion/deletion is $O(1)$ amortized:

$$\Phi = #1,1 + 2 \times #2,2$$
Deletion without rebalancing: a better alternative?

Simplifies deletion, but what happens to balance?

**Critical** idea: maintain and store ranks, **not** rank differences.

Storytime...
Relaxed AVL (ravl) trees

ravel: to clarify by separation into simpler pieces.

All rank differences are positive. Store with each node its rank, **not** its rank difference.

Ranks are defined by the operation sequence; *any* tree is possible!

Balanced?
**Deletion:** standard unbalanced deletion; node ranks do not change, but rank differences can.

**Insertion:** just like AVL-tree insertion:
Give new node $x$ a rank of 0. \( \Delta r(x) = 0 \) (bad) or 1.
To restore balance:

```plaintext
while $x$ is a 0-child whose sibling is a 1-child do
  \{ $x \leftarrow p(x); \ r(x) \leftarrow r(x) + 1$ \}
  (Increase of $r(x)$ changes $x$ from 0,1 to 1,2 but may make $x$ a 0-child.)
if $x$ is a 0-child whose sibling is not a 1-child then
  apply the appropriate one of the following two transformations:
```

blue = $\Delta r$
black = changes in $r$

also two mirror-image cases
A ravl tree
numbers are ranks
\( \lg \lg n + O(1) \) bits per node
Delete D: swap with E, delete
Delete D: swap with E, delete.
Delete E: replace by child. Child’s rank does not change, but its rank difference increases.
Delete E: replace by child. Child’s rank does not change, but rank difference increases.
Insertion bounds for AVL trees hold for ravl trees: ≤2 rotations per insertion worst-case, ≤3 promotions per insertion amortized, even with intermixed deletions.

Height?

Not logarithmic in $n$, current tree size: tree can evolve to have arbitrary structure!
But only slowly.
height ≤ \lg_\varphi m, where \( m = \#\text{insertions} \)

Proof: Use potential function. If \( r(x) = k \),

\[
\Phi(x) = \begin{cases} 
F_{k+2} & \text{if } 0,1 \\
F_{k+1} & \text{if } 0,j \text{ for } j > 1 \\
F_k & \text{if } 1,1 \\
0 & \text{otherwise}
\end{cases}
\]

\( \Phi(T) = \text{sum of node potentials} \)
Deletion does not increase $\Phi$.

Insertion creates a 1,1-node of rank 0 ($\Phi = 0$), and changes the parent from 1,1 to 0,1 or 2,1 to 1,1 ($\Delta \Phi = 1$) or has no effect on $\Phi$. Promotions and rotation cases cannot increase $\Phi$ (you check). Promotion of root of rank $k$ converts a 1,1-node of rank $k$ to a 1,2-node of rank $k + 1$, decreasing $\Phi$ by $F_k$. 
If root has rank $k$, decrease in $\Phi$ due to root promotions is at least

$$\sum_{0 \leq i < k} F_{i+2} = F_{k+3} - 1.$$ 

$\Phi$ increases by at most 1 per insertion, always $\geq 0$, drops by $F_{k+3} - 1 \geq F_{k+2} > \varphi^k$ as a result of root promotions $\rightarrow m > \varphi^k$. 
In ravl trees, balancing steps are exponentially infrequent in rank.

**Proof**: truncate $\Phi$ (0 above rank $k$).

Also true of rank-balanced trees.
Ravl trees with $O(\lg n)$ height bound?

Rebuild occasionally, either all at once or incrementally: e.g. run a background tree traversal that deletes successive items and inserts them into a new tree.

Sorted insertions into an AVL tree, or a ravl tree, produce a tree with height $\lg n + O(1)$. 