COS 423 Lecture 4
Balanced Binary Search Trees

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?

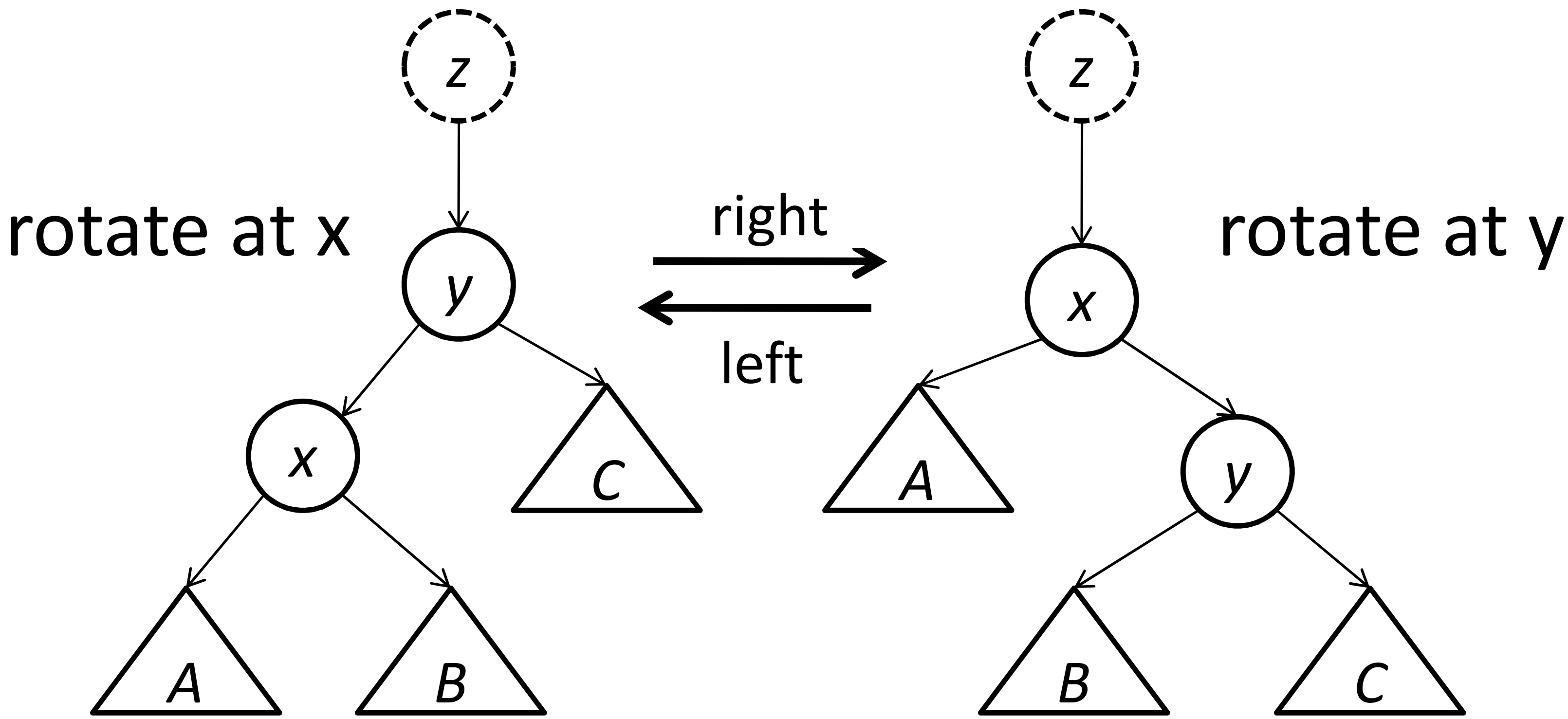
Restructuring primitive: *Rotation*

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
z
y
x
A
B
C
right
left
rotate at y
z
x
y
A
B
C

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: $\Delta r(\mathrm{x}) = r(p(x)) - r(x)$

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*)

$$\rightarrow h(x) = r(x)$$

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*)

$$\rightarrow r(x) \le h(x) \le 2\mathrm{r}(\mathrm{x}) + 1$$

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

$$\rightarrow r(x) \le h(x) \le 2\mathrm{r}(\mathrm{x}) + 1$$

Rank-balanced trees (Sen and Tarjan 2009):

Δr's are 1 or 2; leaves are 1,1

$\rightarrow r(x)/2 \leq h(x) \leq r(x)$

Relaxed AVL (ravl) trees (Sen and Tarjan 2010):

Δr's are positive

$\rightarrow h(x) \leq r(x)$

Many others, notably **weight-balanced trees**:

balance given by size ratio not rank difference.

AVL Trees

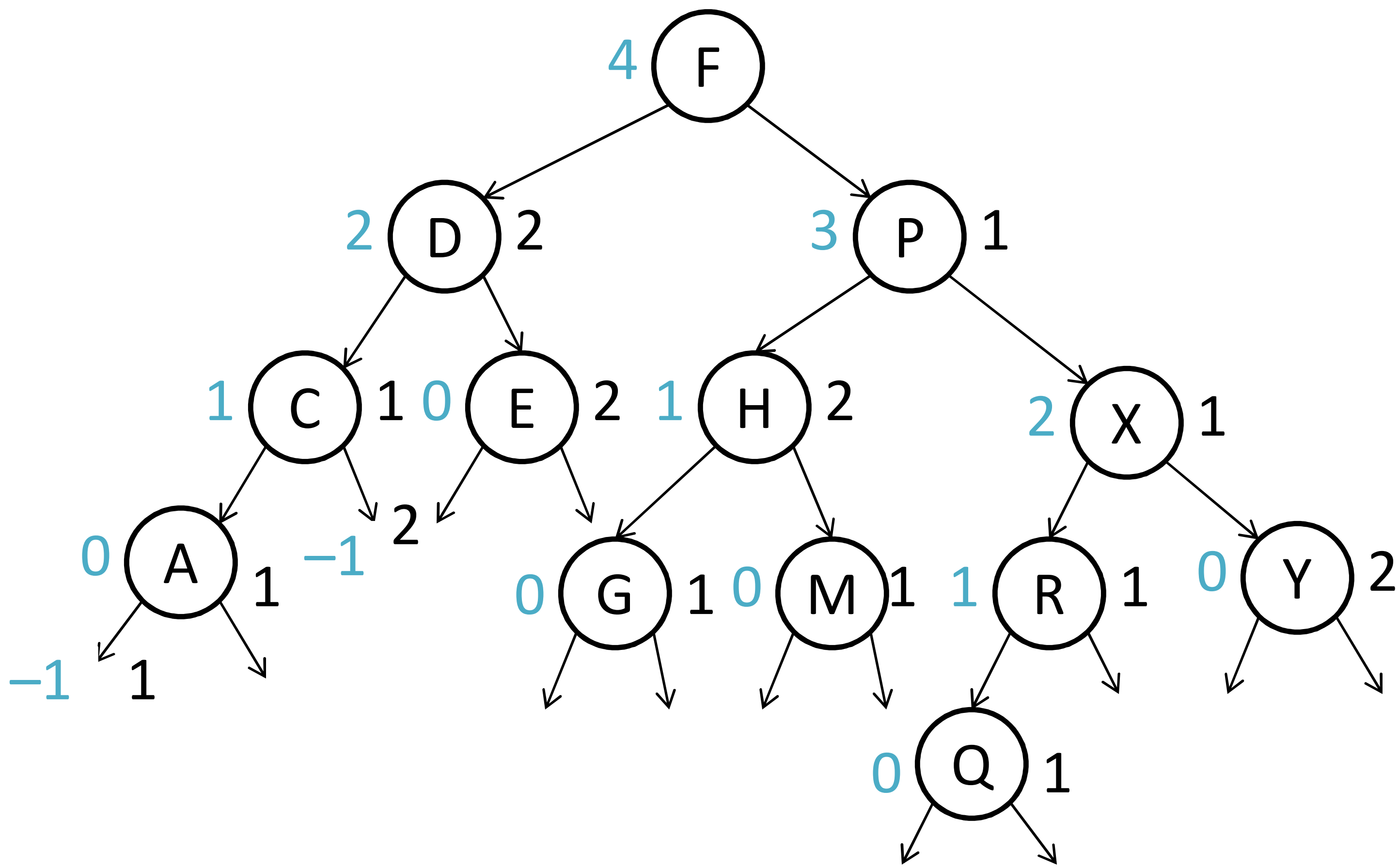
Each node is 1,1 or 1,2

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

Rank differences stored, not ranks: one bit per node, indicating whether Δr is 1 or 2. r's are computable from Δr'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 Δ*r*'s (stored). Null nodes have *r* = –1 and Δ*r* = 1 or 2 (*r* and Δ*r* shown for two null nodes)



AVL-tree height bound

Fibonacci numbers

$F_0 = 0$, $F_1 = 1$, $F_k = F_{k-1} + F_{k-2}$ for $k > 1$

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55,…

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):

$s(\text{null}) + 1 = 0 + 1 = F_2$

$s(\text{leaf}) + 1 = 1 + 1 \geq F_3$

$r(x) > 0$: $s(x) + 1 = s(\mathit{left}(x)) + 1 + s(\mathit{right}(x)) + 1 \geq$

$F_{r(x) + 2} + F_{r(x) + 1} = F_{r(x) + 3}$ since x is 1,1 or 1,2

$n + 1 \geq F_{h + 3} \geq \varphi^{h + 1} \rightarrow$

$\boldsymbol{h \leq \lg_{\varphi}(n + 1) - 1 \leq \lg_{\varphi} n < 1.44043 \lg n}$

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 ≤2 rotations.

AVL-tree insertion

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

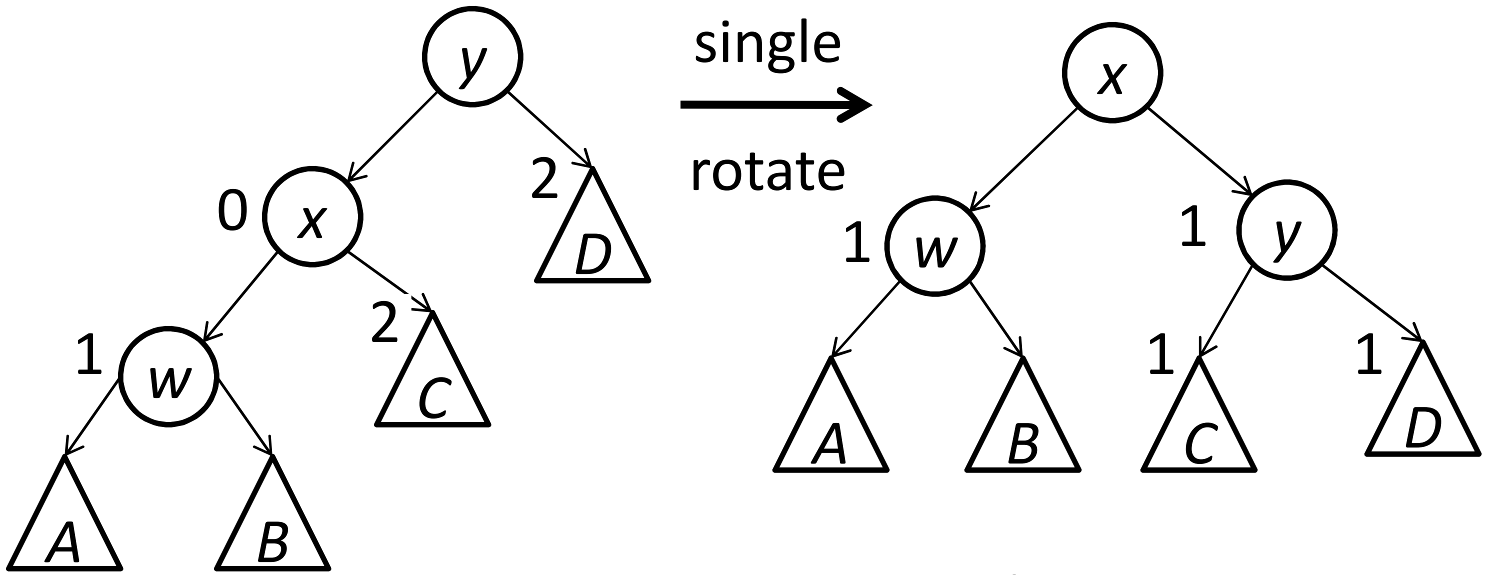
To restore balance:

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

$\{x \leftarrow p(x); \mathrm{r}(\mathrm{x}) \leftarrow \mathrm{r}(\mathrm{x}) + 1\}$

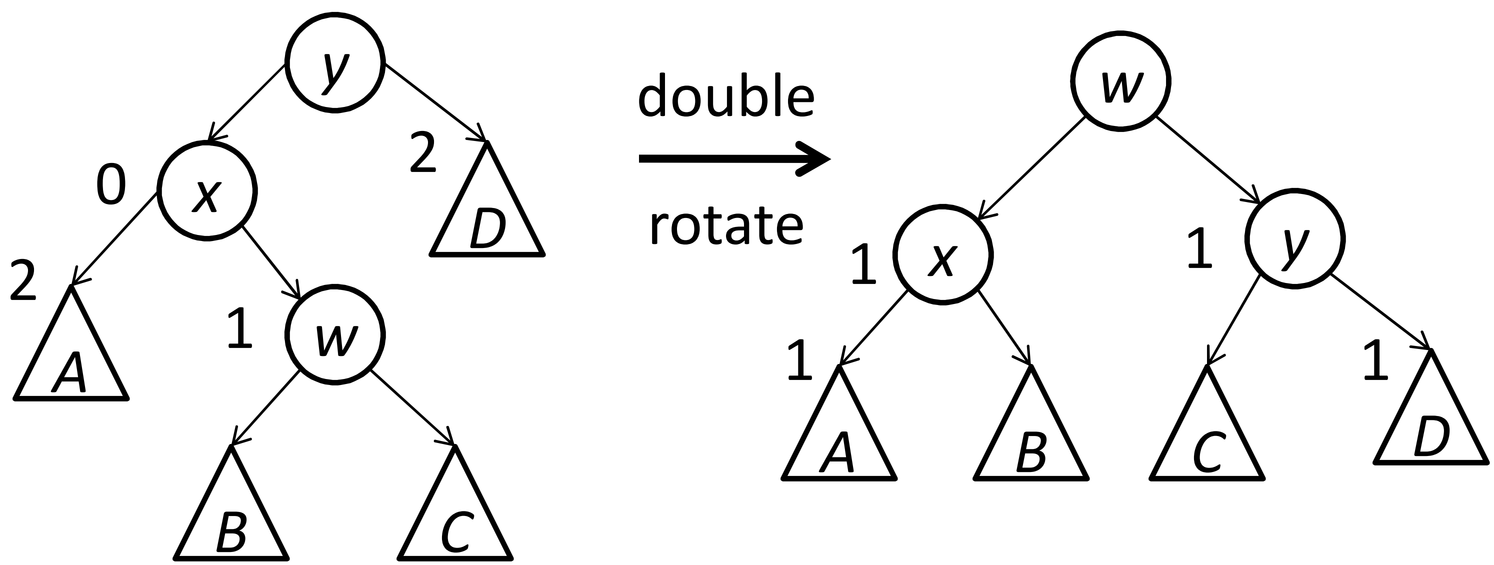
(Increase of $r(x)$ changes x from 0,1 to 1,2 but may make x a 0-child.)

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):



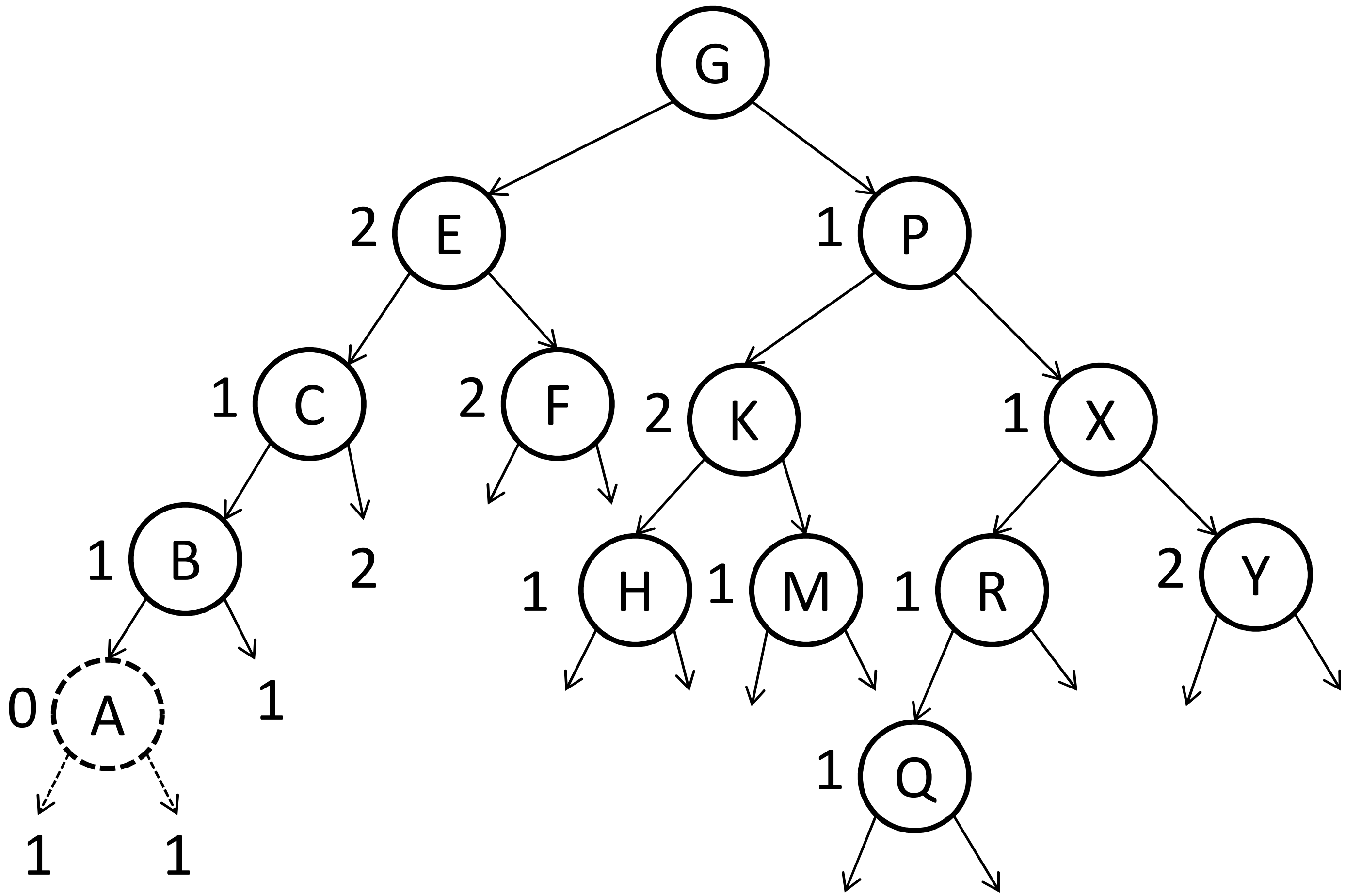
numbers are Δ*r*'s

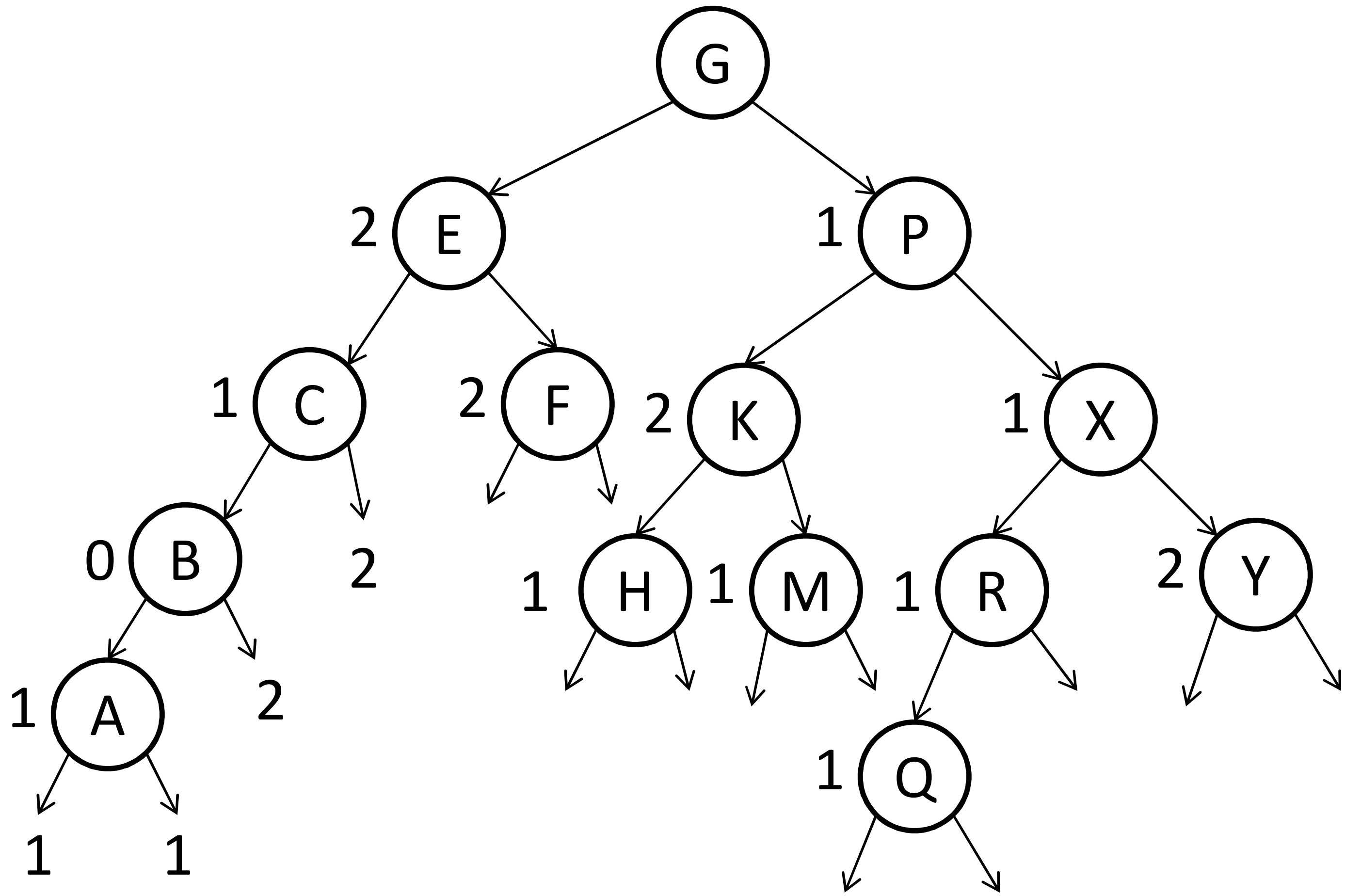
also two mirror-image cases

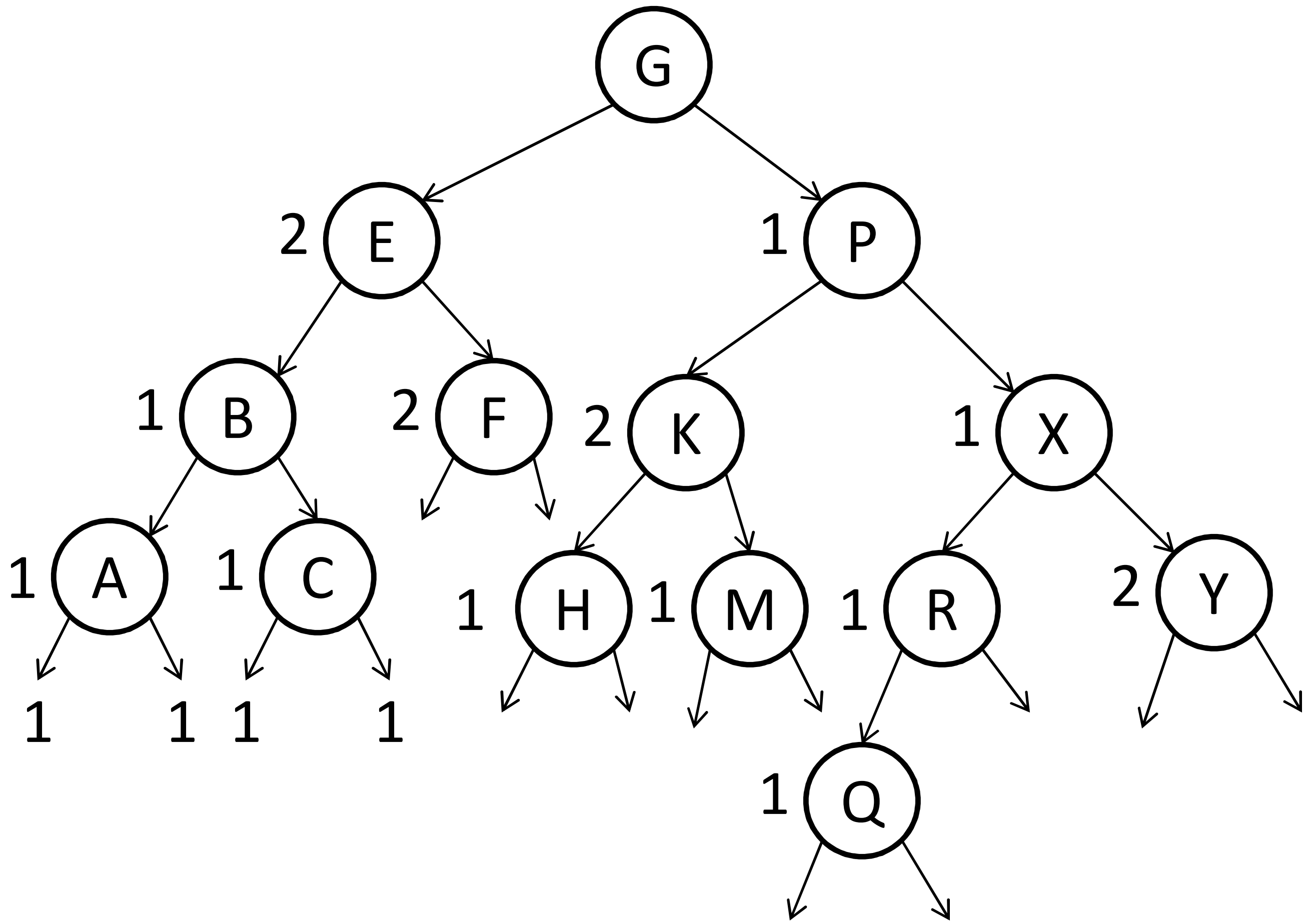


Insert A

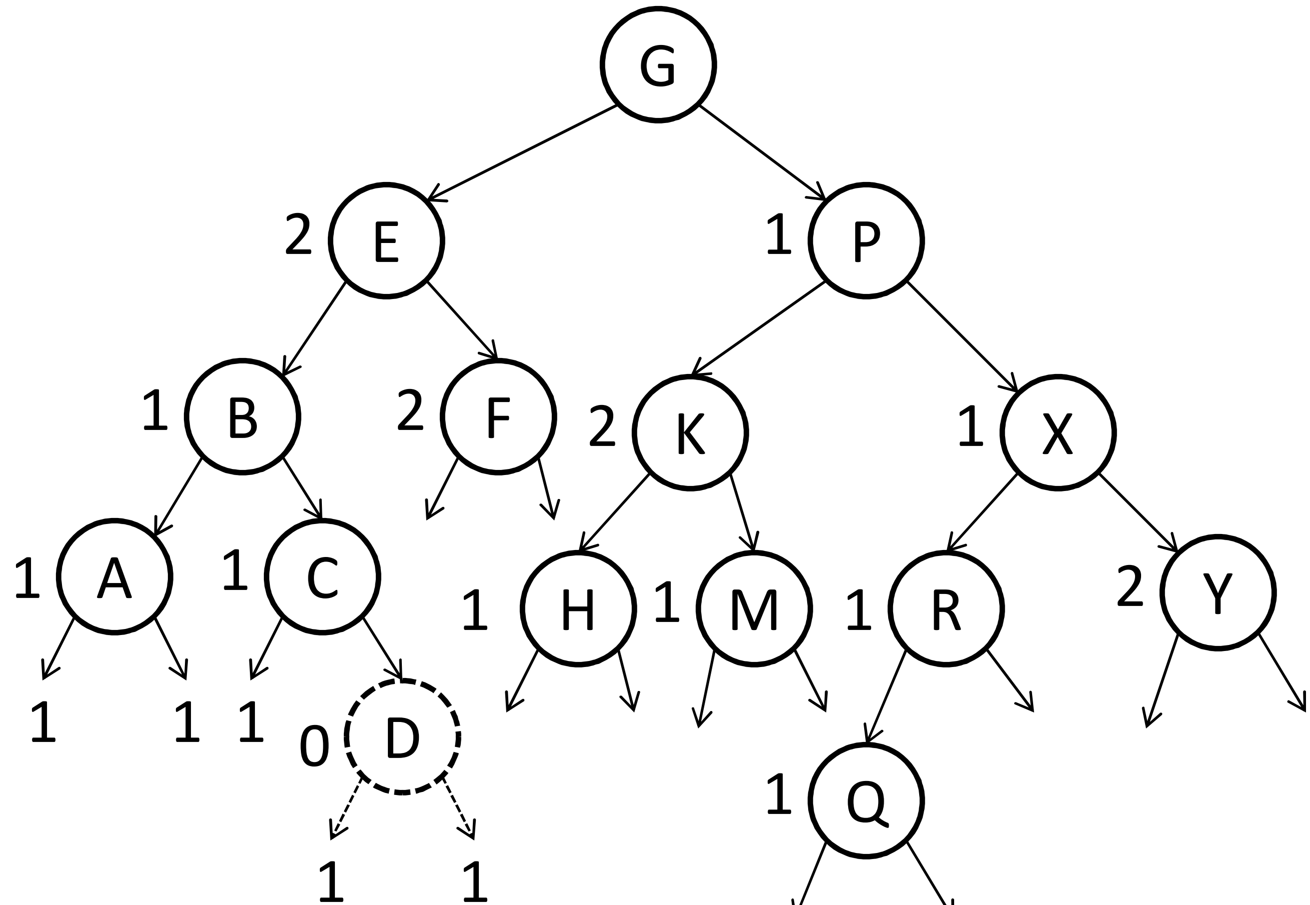
numbers are Δ*r*'s

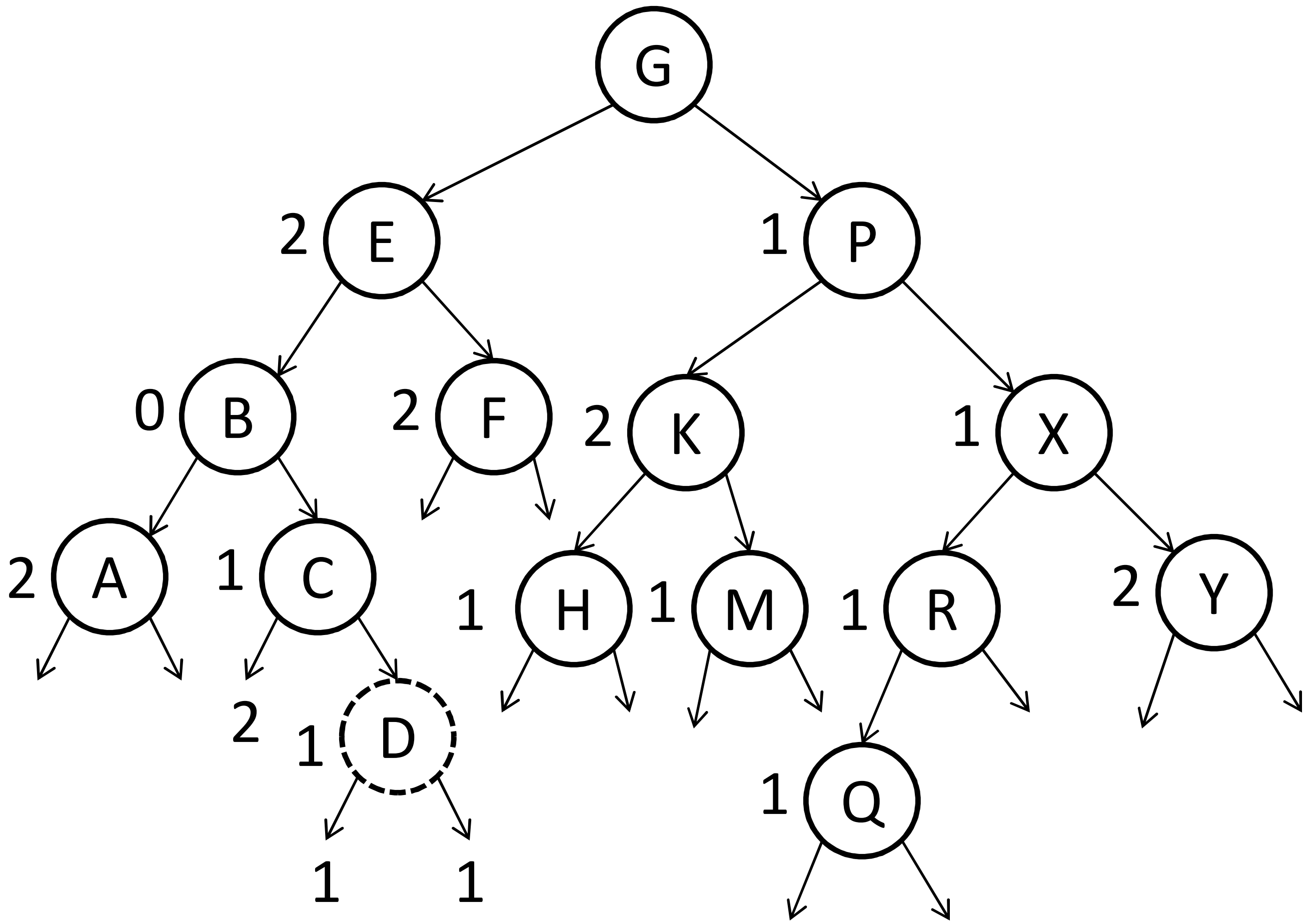


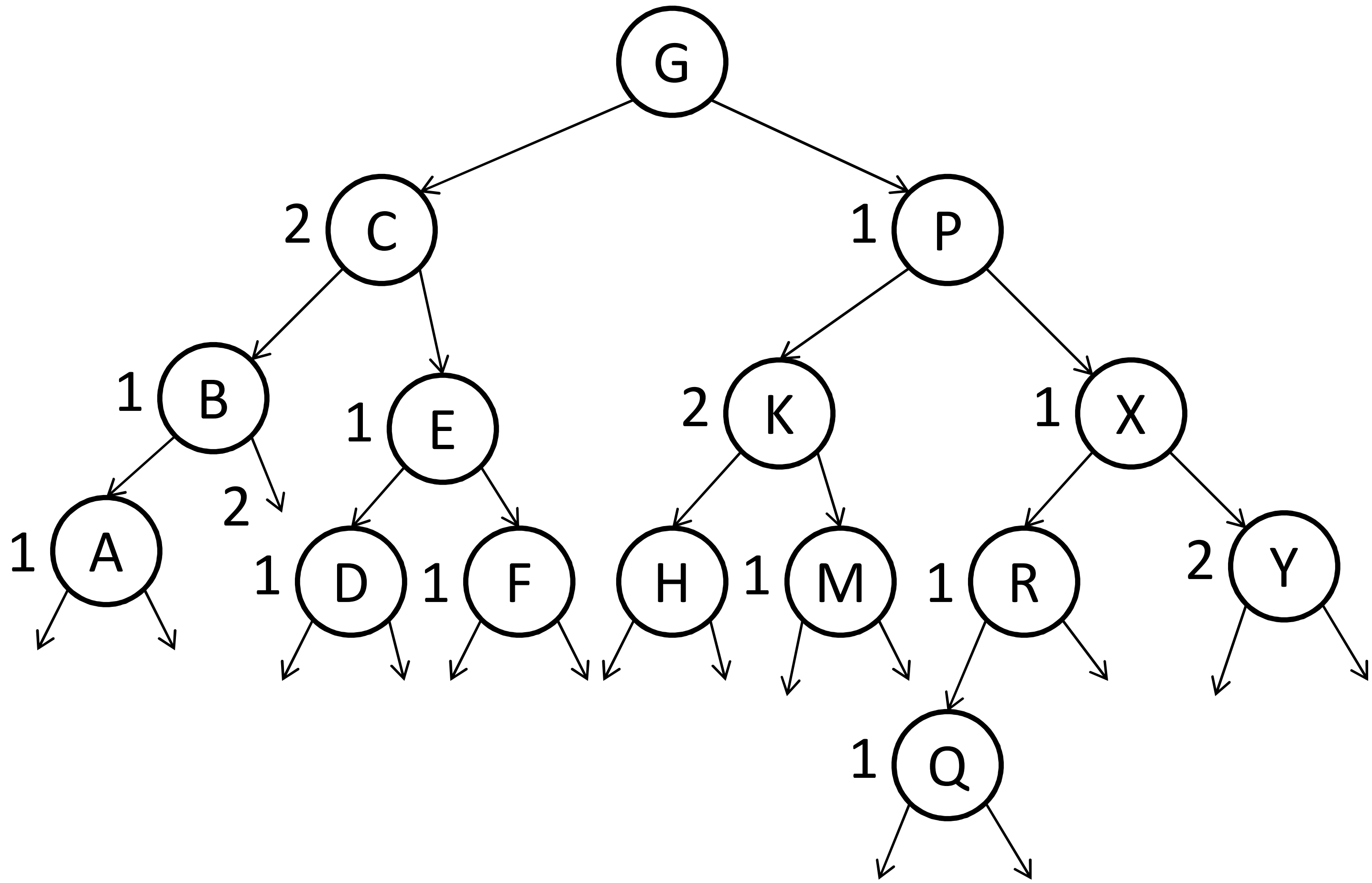

G
2
E
1
P
1
C
2
F
2
K
1
X
0
B
2
1
H
1
M
1
R
2
Y
1
A
2
1
Q
1
1


G
2
E
1
P
1
B
2
F
2
K
1
X
1
A
1
C
1
H
1
M
1
R
2
Y
1
1
1
1
1
Q

Insert D




G
2
E
1
P
0
B
2
F
2
K
1
X
2
A
1
C
1
H
1
M
1
R
2
Y
2
1
D
1
Q
1
1



G
2
C
1
P
1
B
1
E
2
K
1
X
2
1
A
1
D
1
F
H
1
M
1
R
2
Y
1
Q

AVL-tree insertion: ≤2 rotations per insertion, worst-case.

What about promotions(rank increases)?

O(lg*n*) worst-case but O(1) amortized:

Φ = #1,1-nodes + #0,1-nodes

Creation of a new node increases Φ by 1; each promotion decreases Φ by 1; last step increases Φ by at most 2: ≤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 → 0,1 → 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 ≤ 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, $s(x) + 1 = s(\mathit{left}(x)) + 1 + s(\mathit{right}(x)) + 1$

$\geq 2^{r(x)/2} + 2^{r(x)/2} = 2^{r(x)/2 + 1}$ 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)

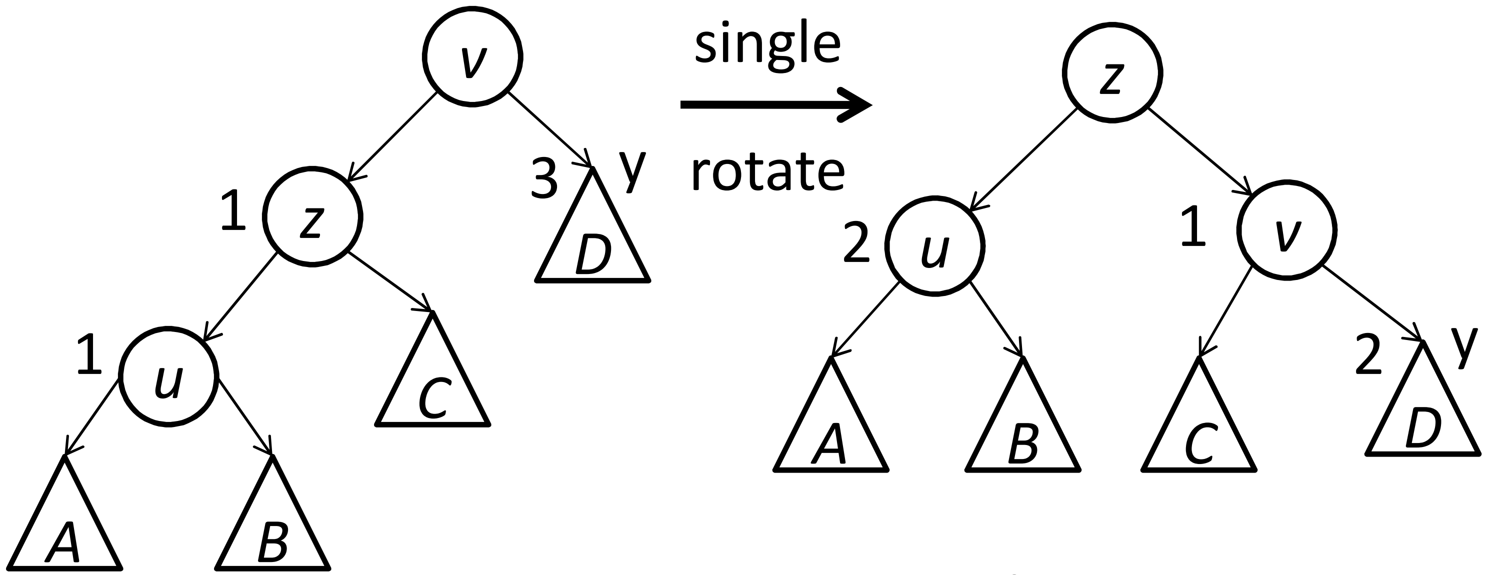
Insertion: same bottom-up rebalancing algorithm as AVL trees: no 2,2's created, but one can be destroyed.

Deletion of x (bottom-up): If x binary, swap with successor. Let $y = p(x)$ (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 (Δr too big).

To restore balance:

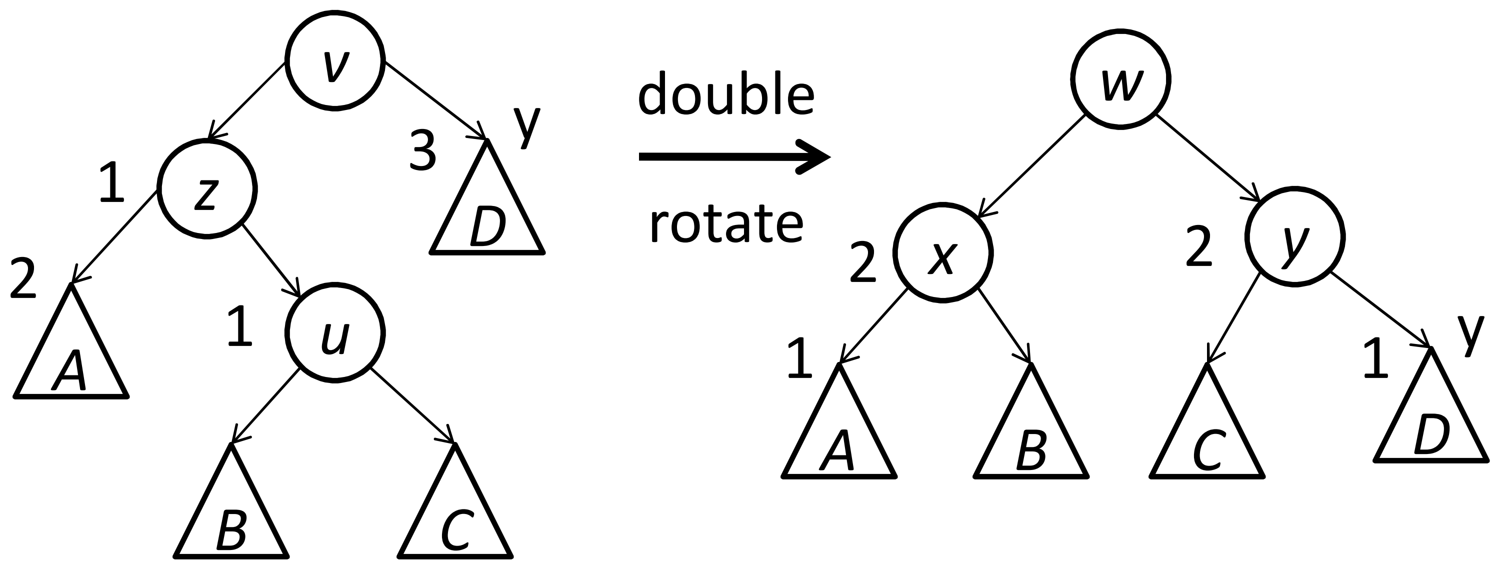
while *y* is a 3-child with sibling *z* a 2-child or 2,2
do { if *z* not a 2-child then r(*z*) $\leftarrow$ r(*z*) – 1;
y $\leftarrow$*p*(*y*); r(y) $\leftarrow$ r(y) – 1}
(one or two demotions; new y may be a 3-child)

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):



numbers are Δ*r*’s

also two mirror-image cases



4 cases for deletion including 2 non-terminating demotion cases (rank decreases) vs. 3 for insertion ($\times 2$ for mirror-imagecases = 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.

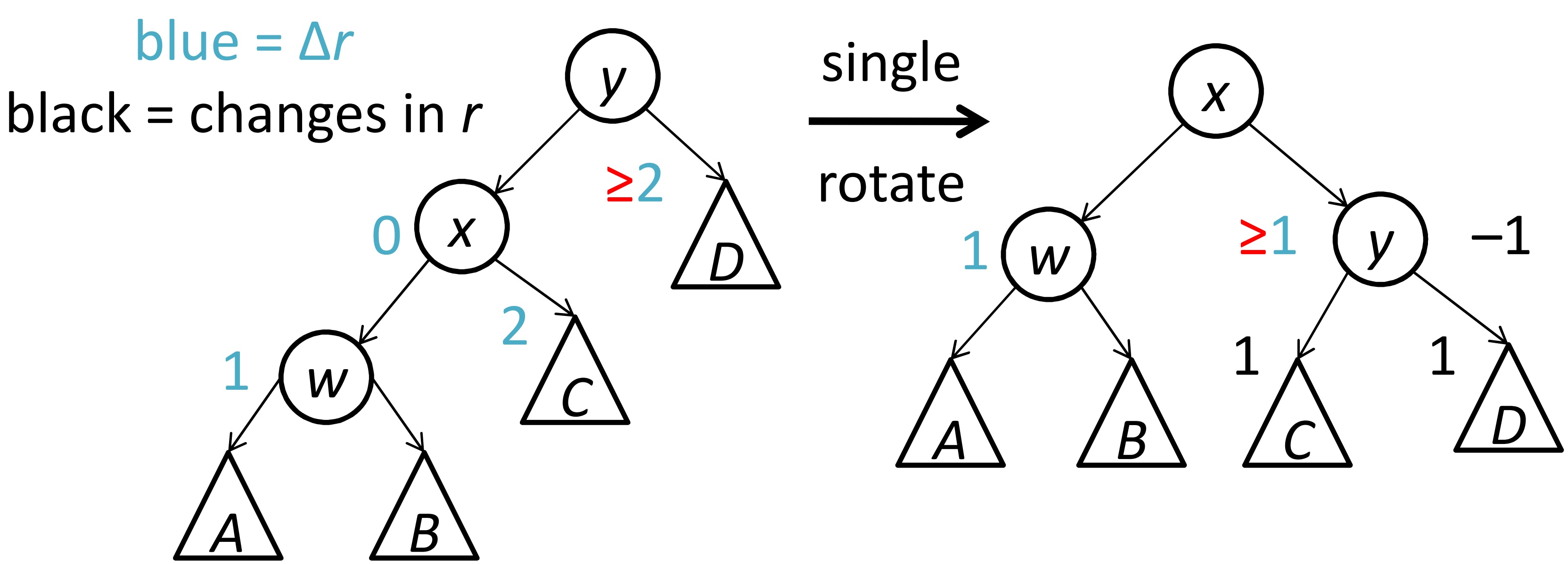
To restore balance:

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

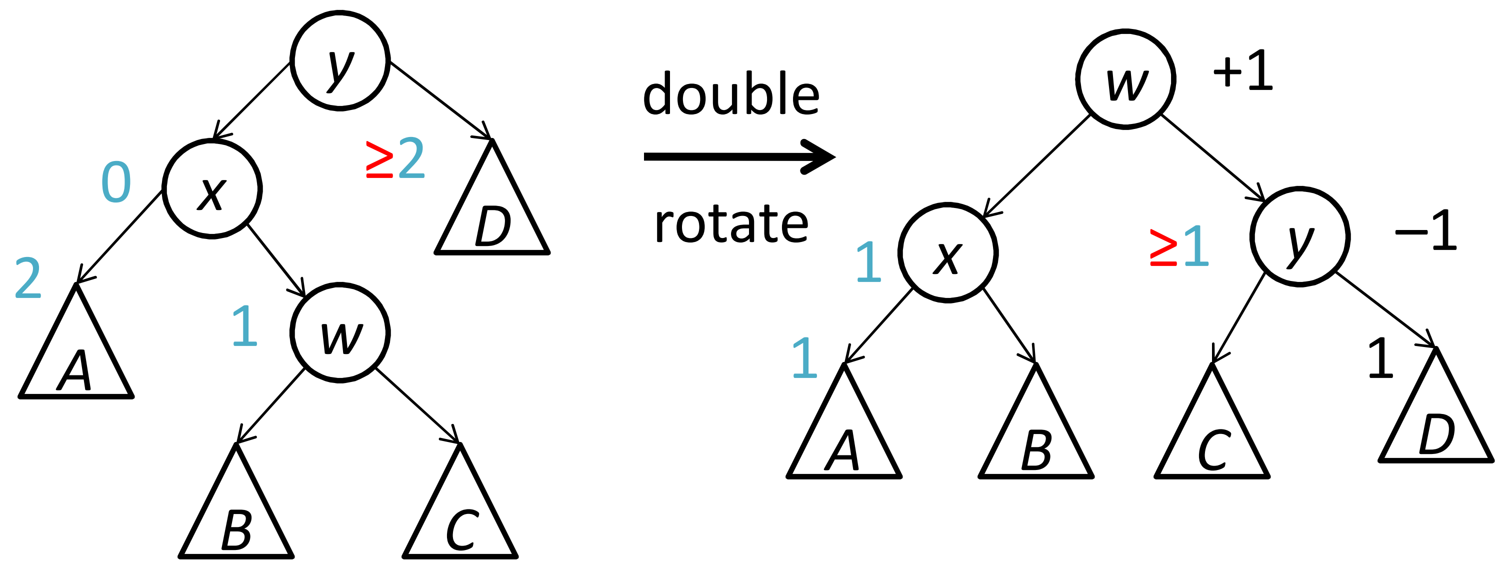
$\{x \leftarrow p(x);\ \mathrm{r(x)} \leftarrow \mathrm{r(x)} + 1\}$

(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:



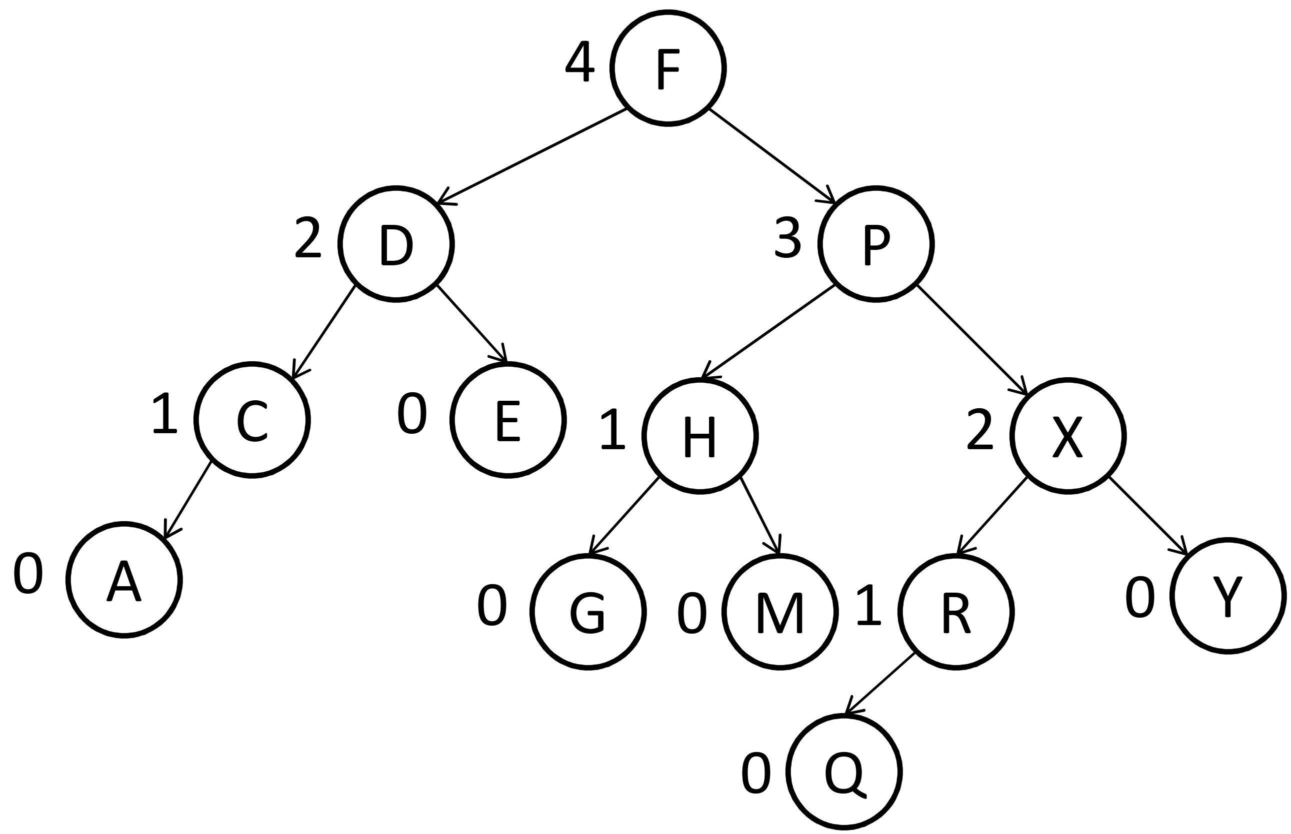
also two mirror-image cases



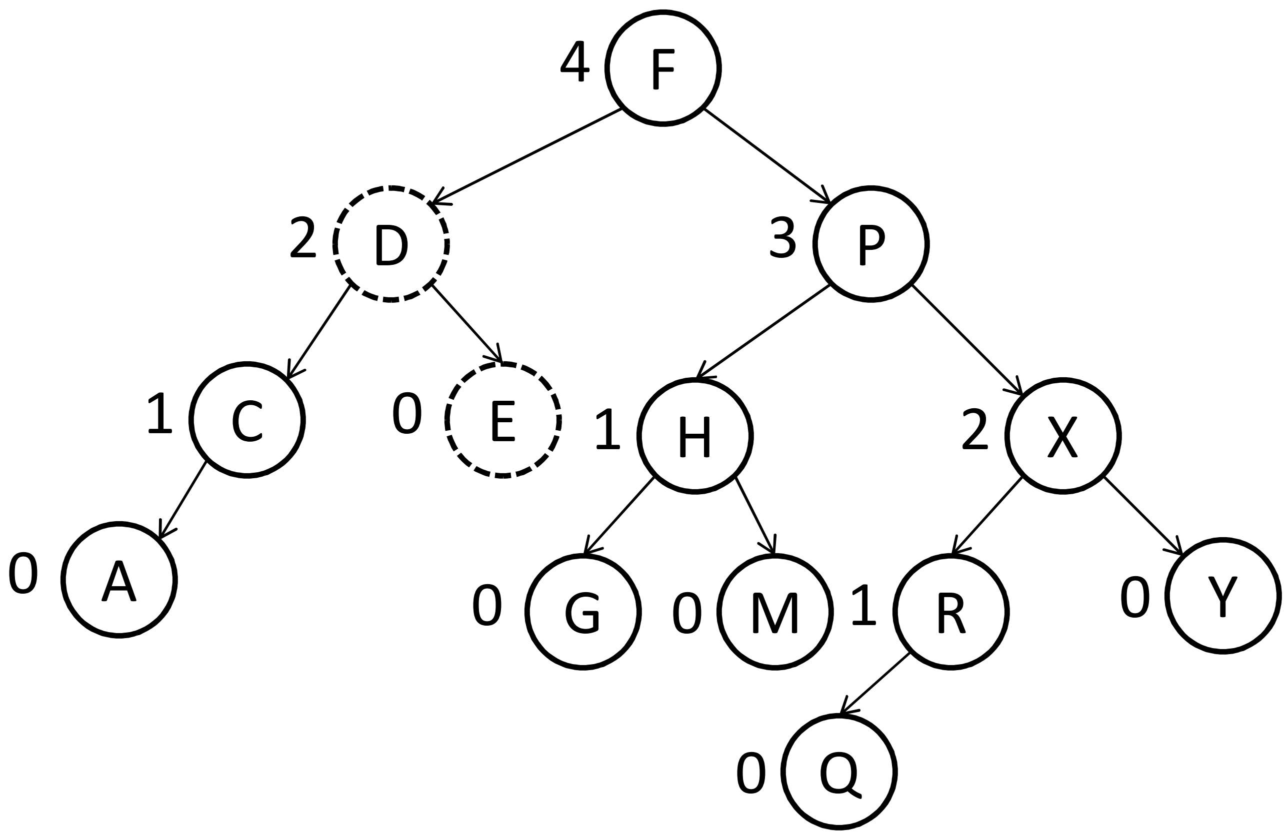
A ravl tree

numbers are ranks

lglg$n$ + O(1) bits per node

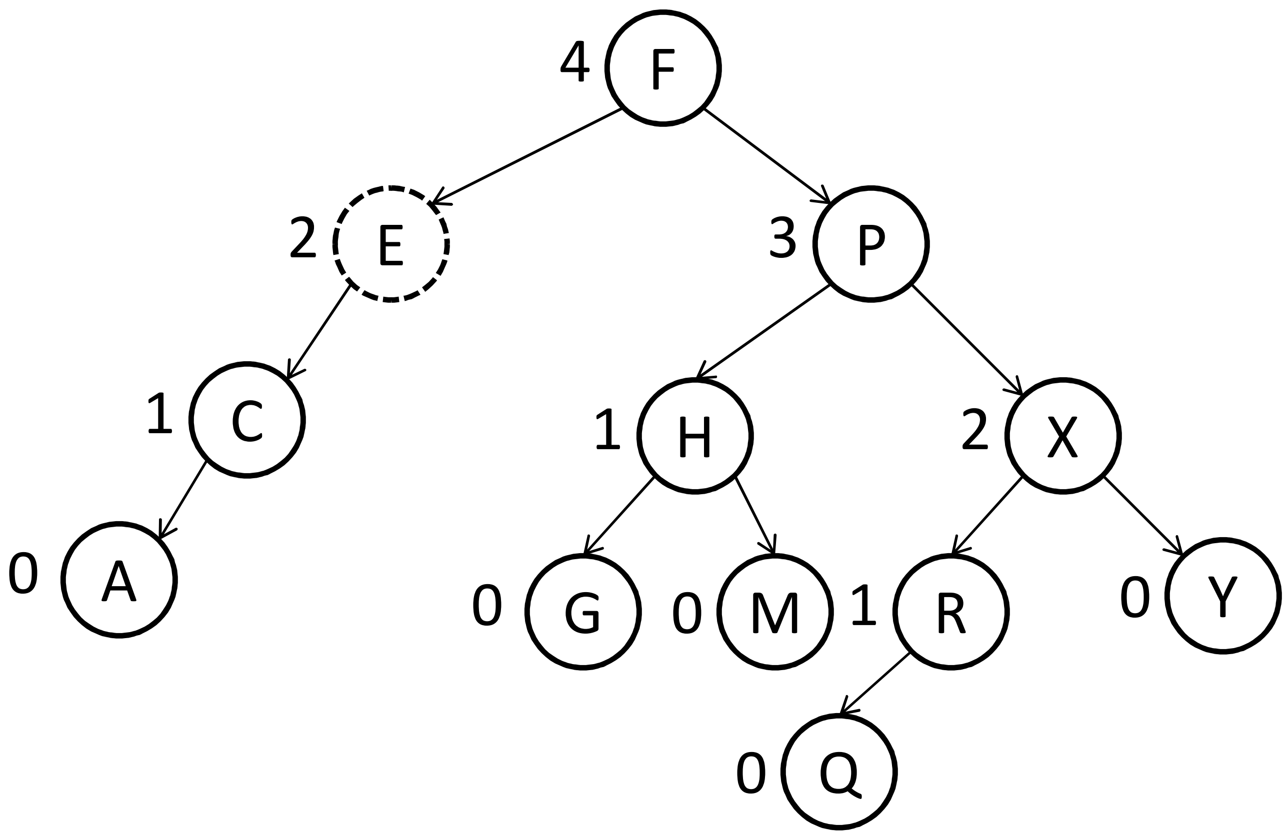


Delete D: swap with E, delete

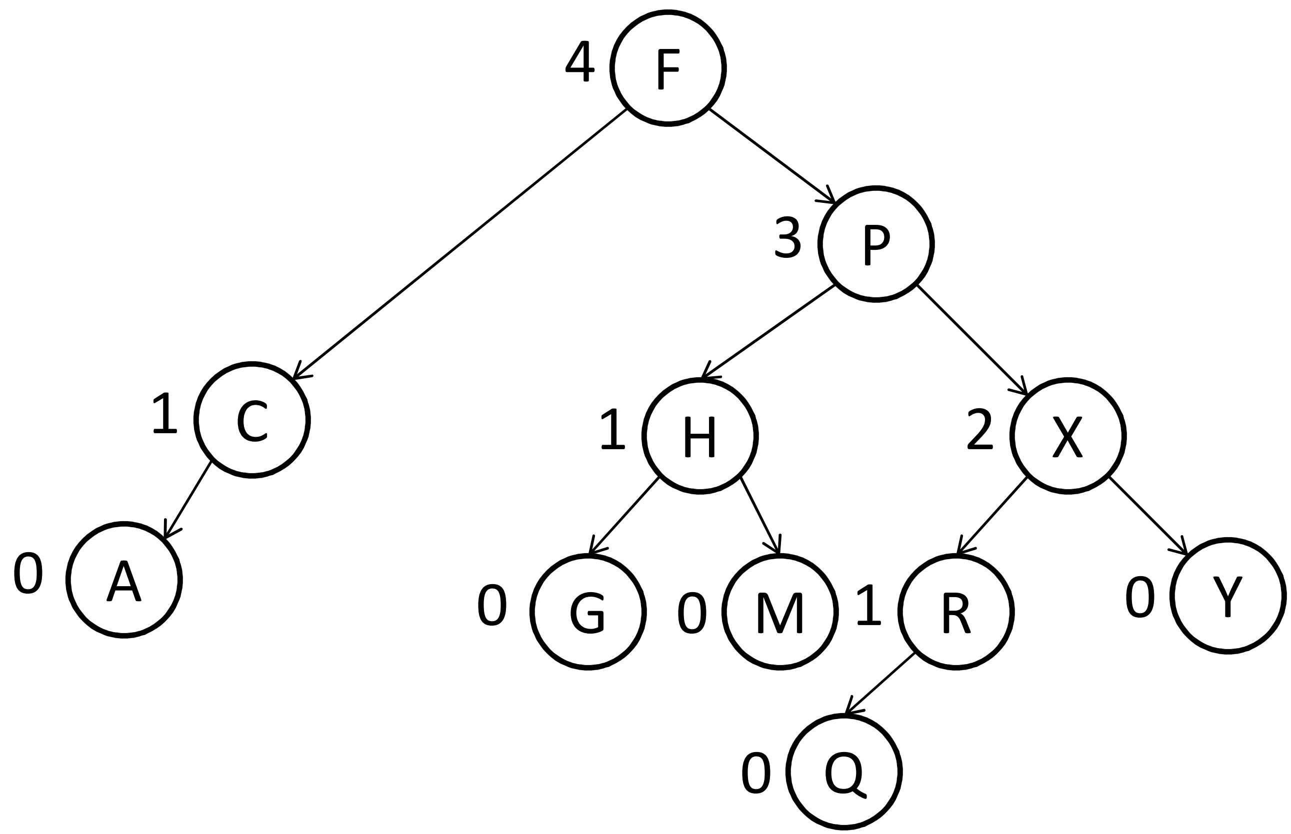


Delete D: swap with E, delete.

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 $\leq \lg_{\varphi} m$, where m = #insertions

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

$\Phi(x) = F_{k+2}$ if 0,1

F_{k+1} if 0,j for $j > 1$

F_k if 1,1

0 otherwise

$\Phi(T)$ = sum of node potentials

Deletion does not increase Φ.

Insertion creates a 1,1-node of rank 0 (Φ = 0), and changes the parent from 1,1 to 0,1 or 2,1 to 1,1 (ΔΦ = 1) or has no effect on Φ. Promotions and rotation cases cannot increase Φ (**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 Φ by F_k.

If root has rank *k*, decrease in Φ due to root promotions is at least

$$\Sigma\{F_{i+2} \,|\, 0 \le i < k\} = F_{k+3} - 1.$$

Φ increases by at most 1 per insertion, always ≥ 0, drops by $F_{k+3} - 1 \geq F_{k+2} > \varphi^k$ as a result of root promotions → $m > \varphi^k$.

In ravl trees, balancing steps are exponentially infrequent in rank.

Proof: truncate Φ (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).