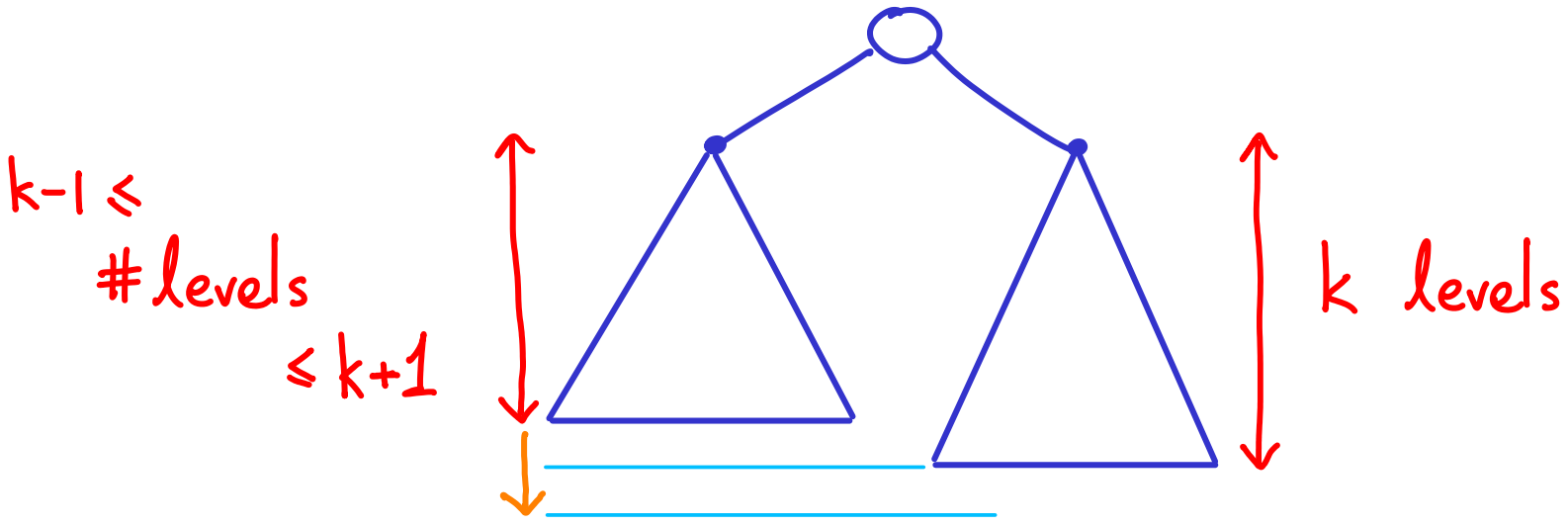
AVL TREES (Adelson-Velsky & Landis, 1962)

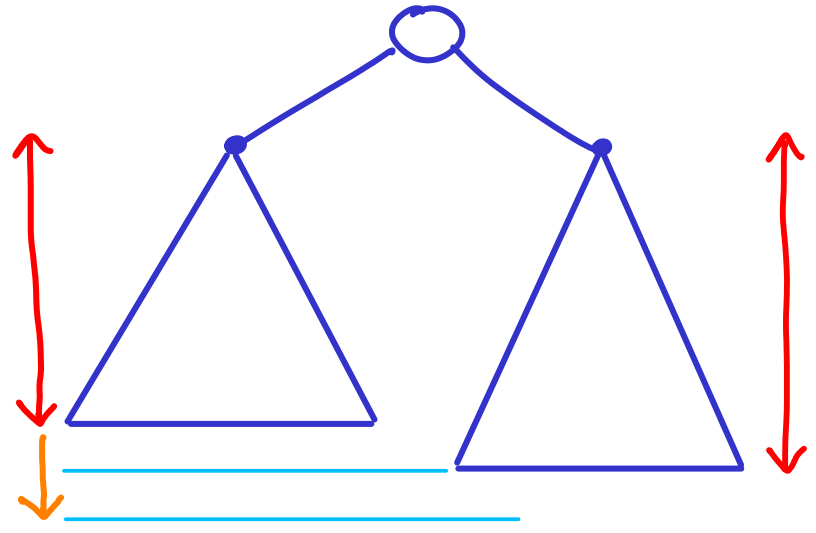
AVL TREES (Adelson-Velsky & Landis, 1962)

Dynamic (height)-balanced binary search tree



AVL TREES (Adelson-Velsky & Landis, 1962)

Dynamic (height)-balanced binary search tree



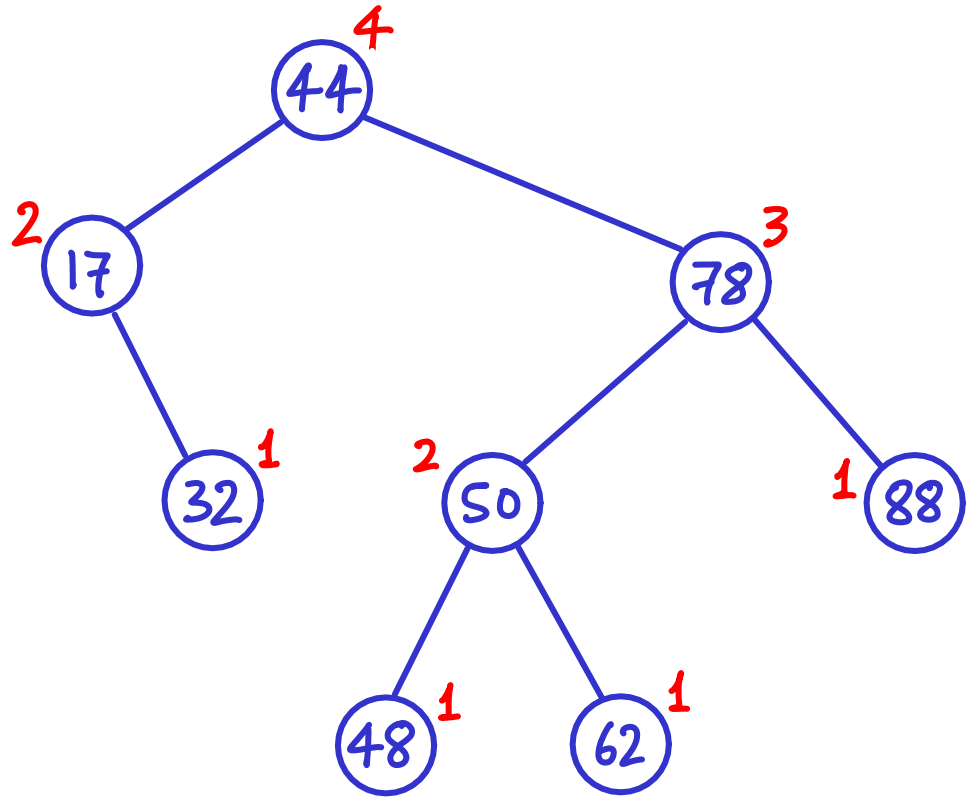
$k-1 \leq$ #levels $\leq k+1$

k levels

(recursively)

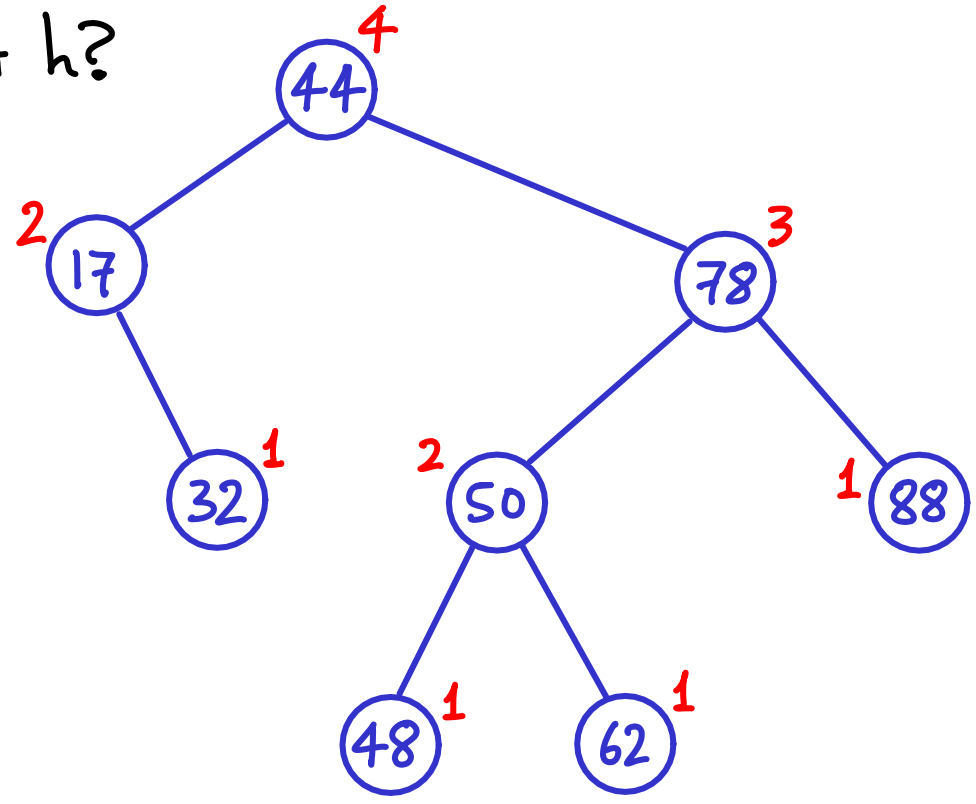
more strict than RB-trees

Augment with heights



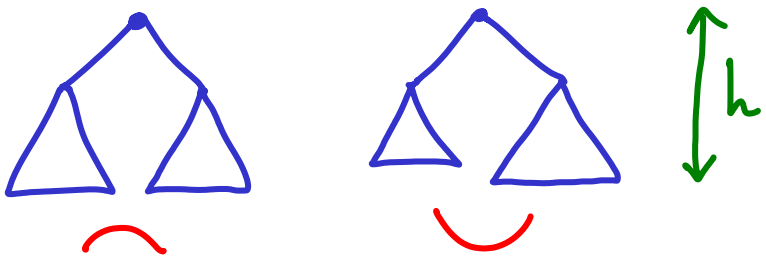
How many nodes are required to reach height h?

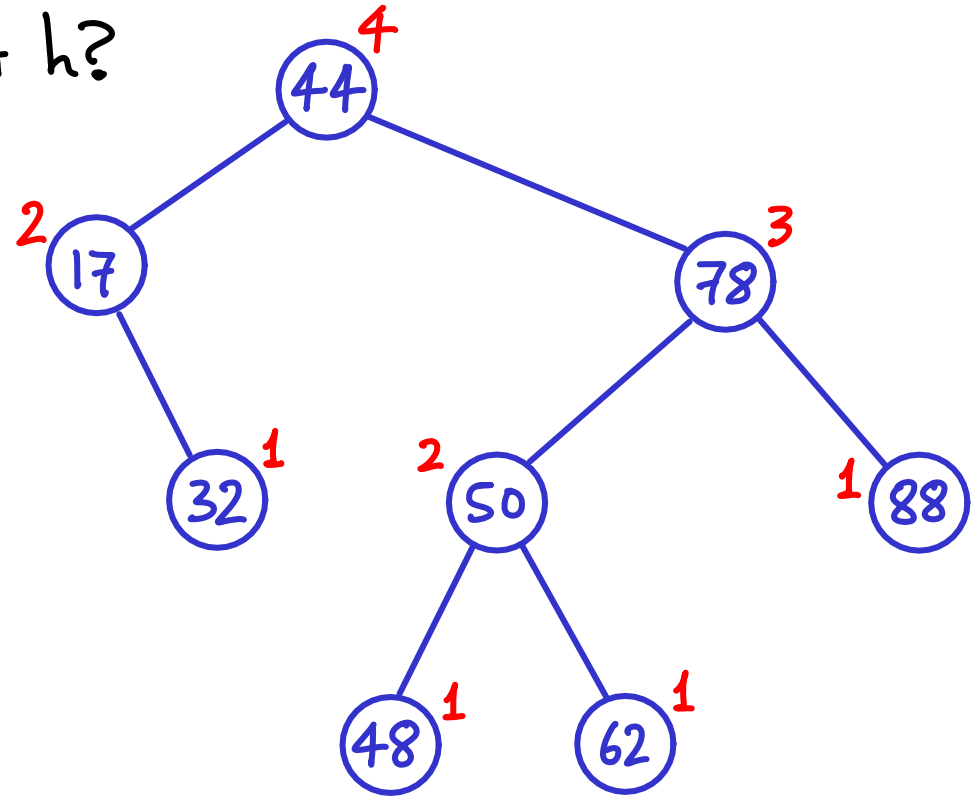
↳ N(h)



How many nodes are required to reach height h?

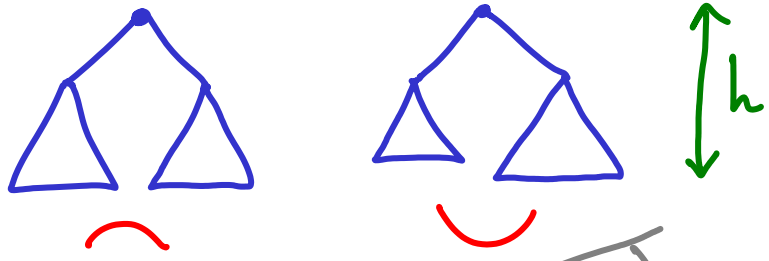
↳ N(h) ↓ minimize



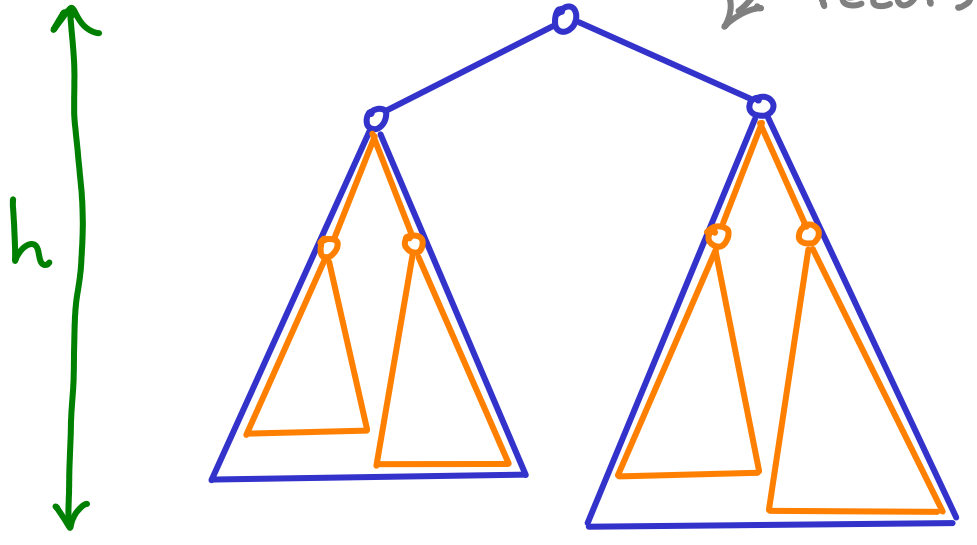


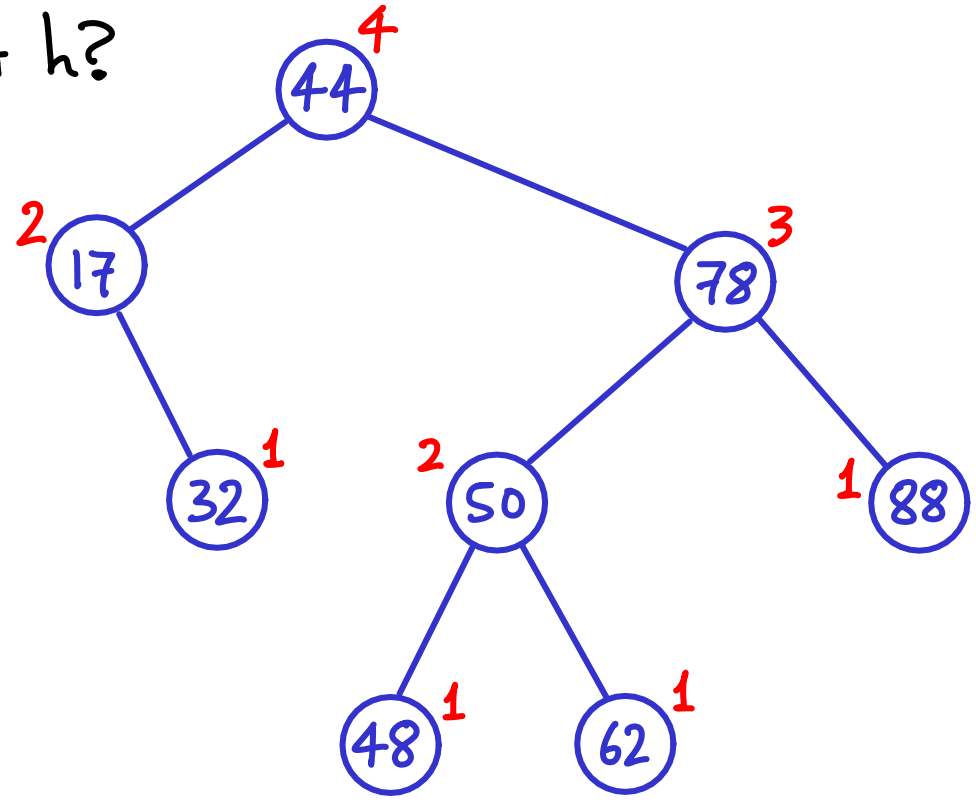
How many nodes are required to reach height h?

↳ N(h) ↓ minimize



do this recursively

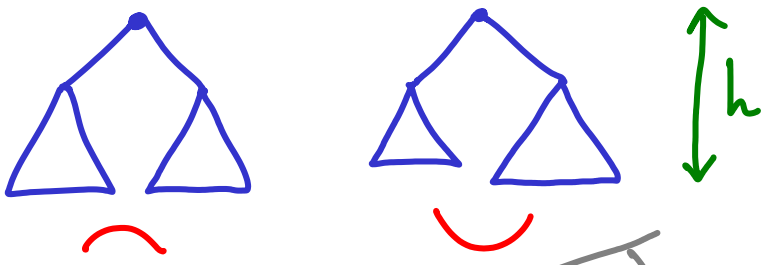




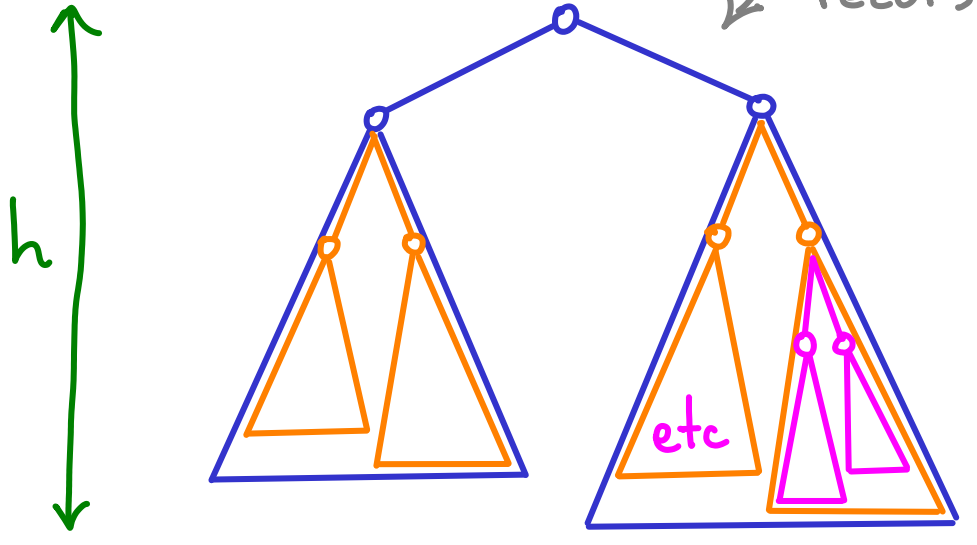
How many nodes are required to reach height h?

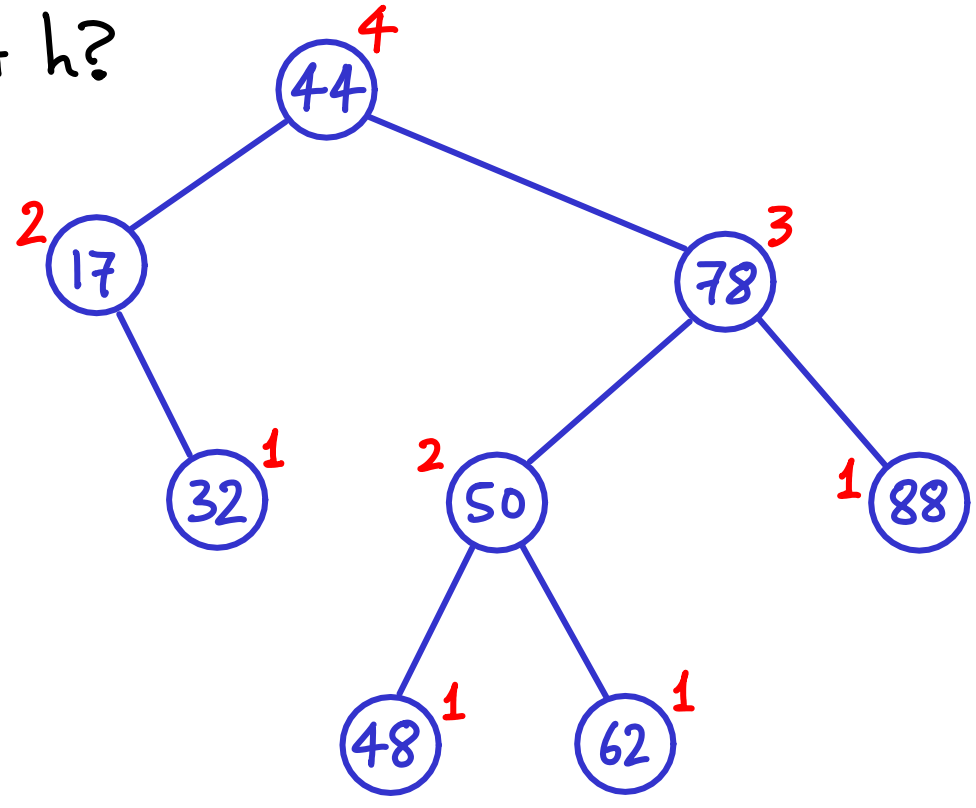
↳ N(h)

↓ minimize



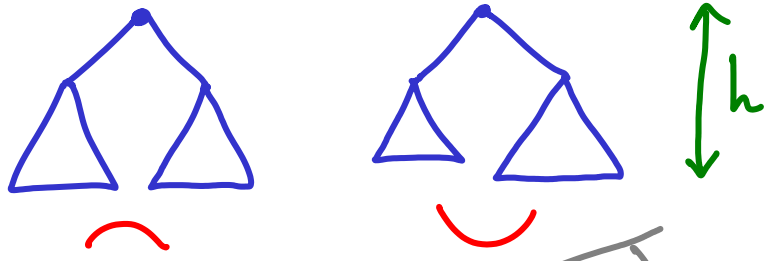
do this recursively



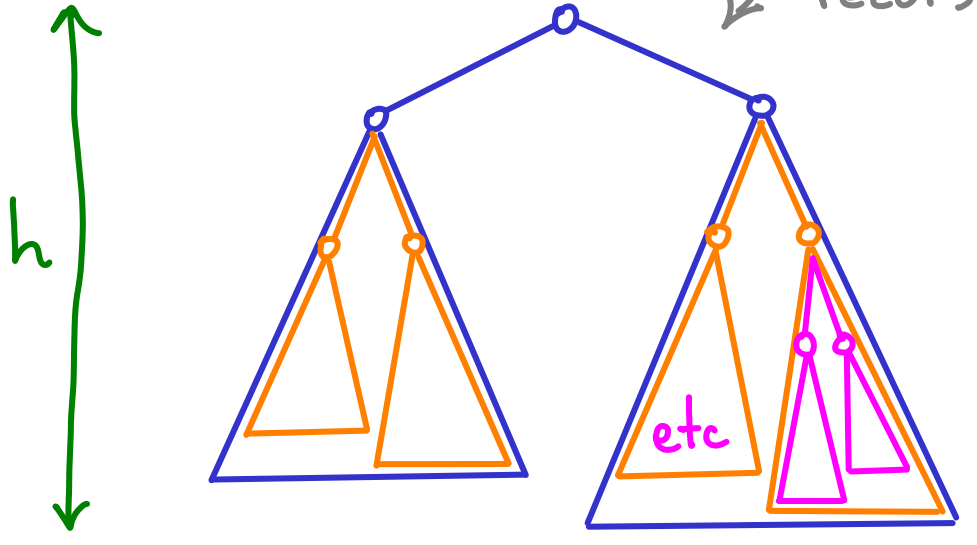


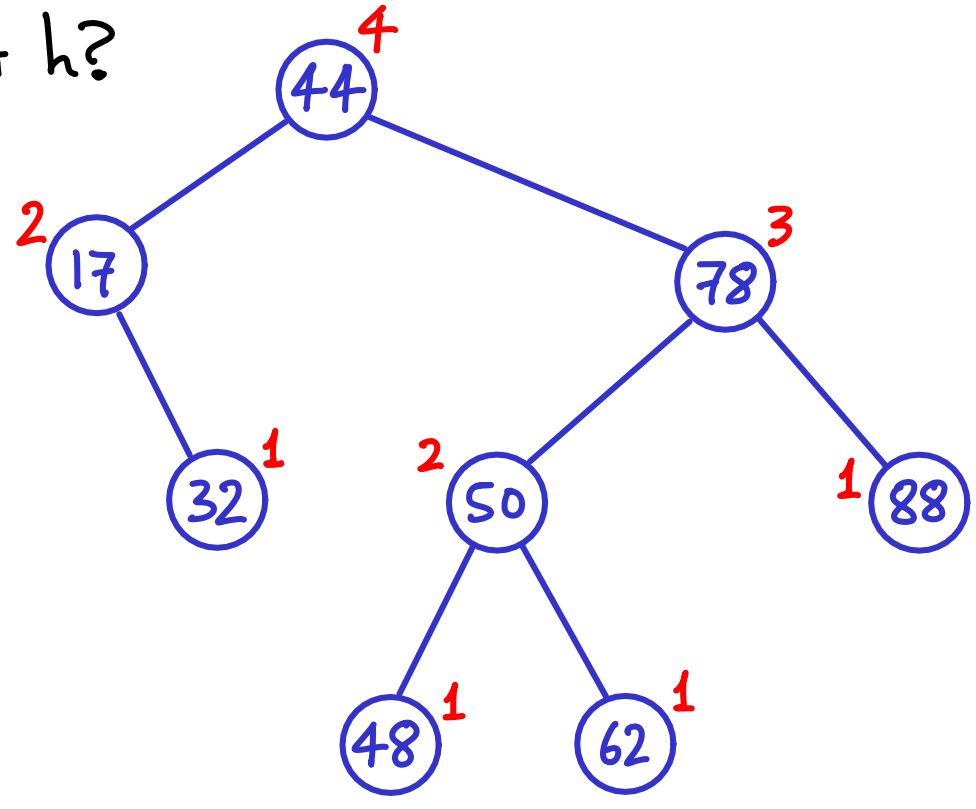
How many nodes are required to reach height h?

↳ N(h) ↓ minimize



do this recursively



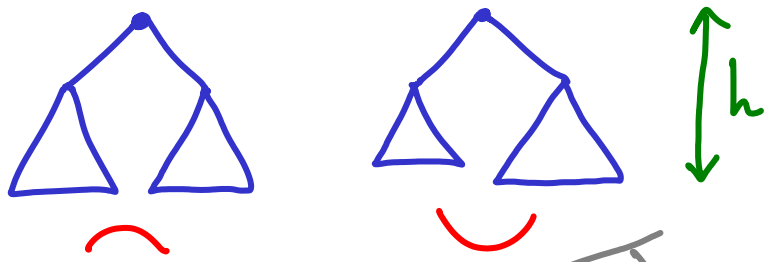


$N(h) = ?$

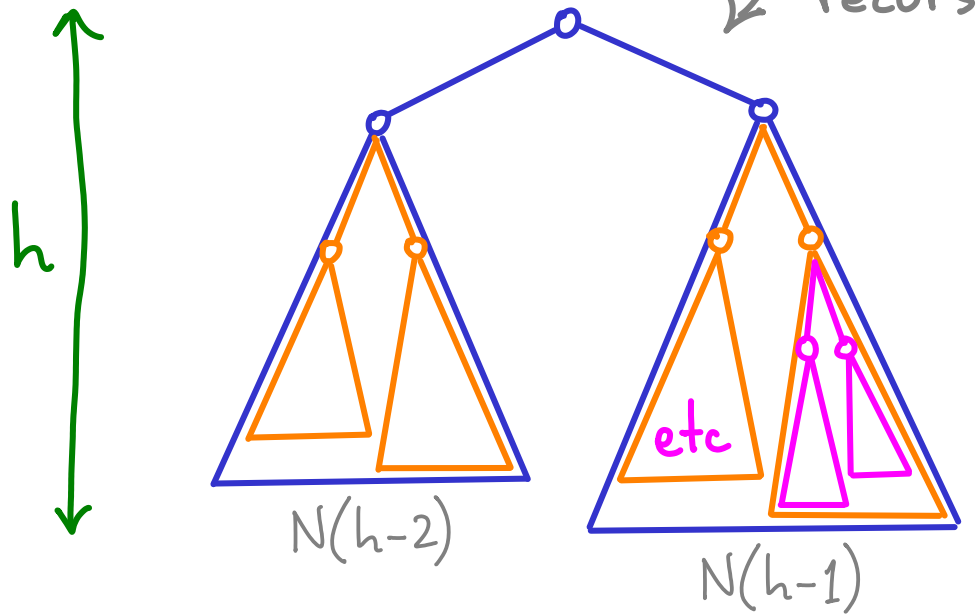
How many nodes are required to reach height h?

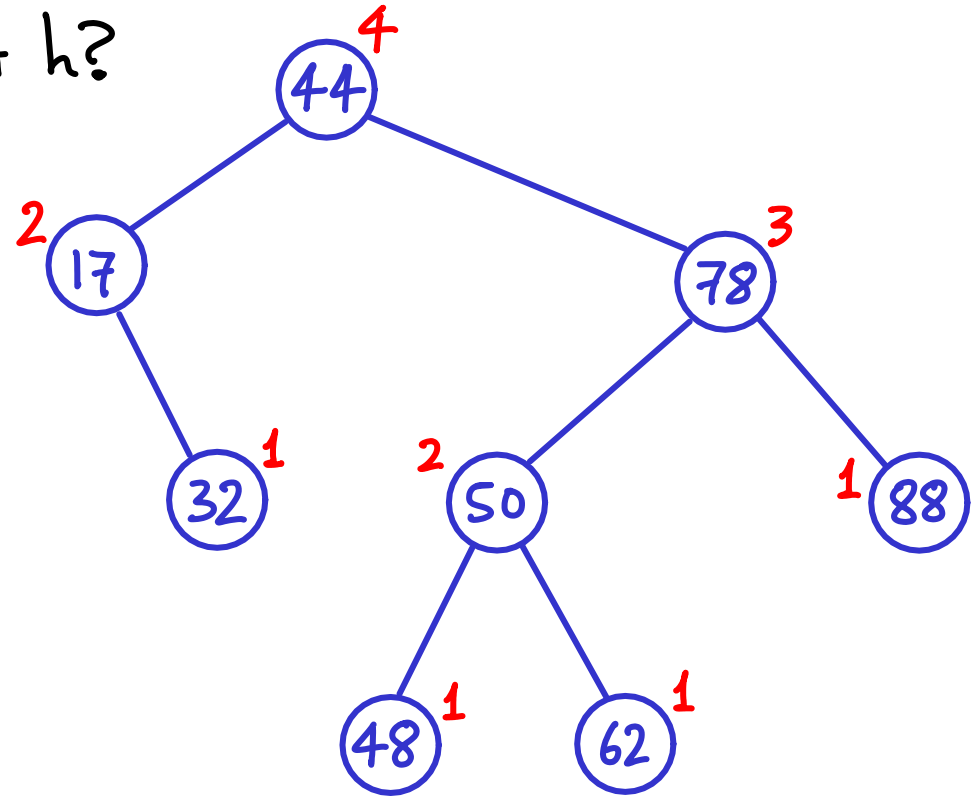
↳ N(h)

minimize



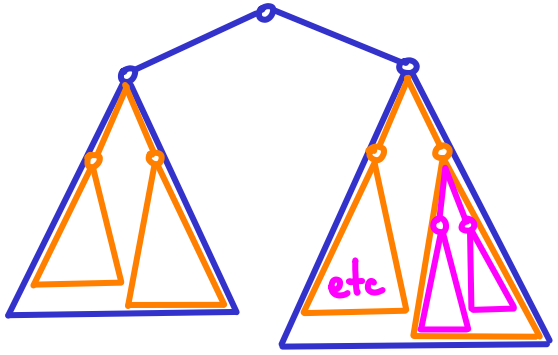
do this recursively





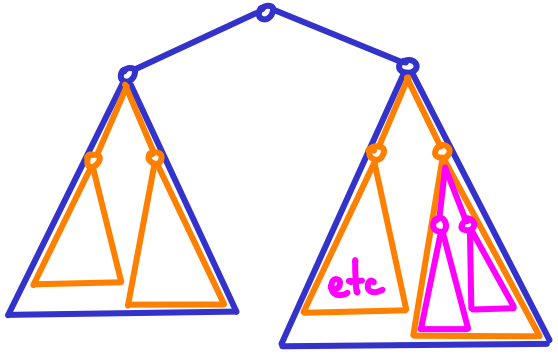
$$N(h) = 1 + N(h-2) + N(h-1)$$

$N(1) = 1$ $\quad$ $N(2) = 2$



$$N(h) = 1 + N(h-2) + N(h-1)$$

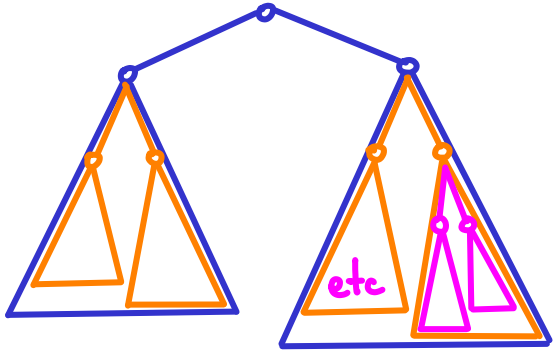
$N(1) = 1$ $N(2) = 2$



$$N(h) = 1 + N(h-2) + N(h-1)$$

$N(1) = 1$ $\quad$ $N(2) = 2$

$N(h) > \ ?$

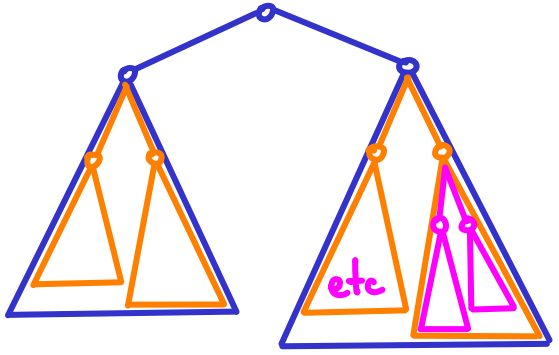


$$N(h) = 1 + N(h-2) + N(h-1)$$

$N(1) = 1$ $\qquad$ $N(2) = 2$

$$N(h) > 2N(h-2) = \Omega(2^{h/2})$$

$$\hookrightarrow h < 2\log_2 N$$



$$N(h) = 1 + N(h-2) + N(h-1)$$

$N(1) = 1$ $\quad$ $N(2) = 2$

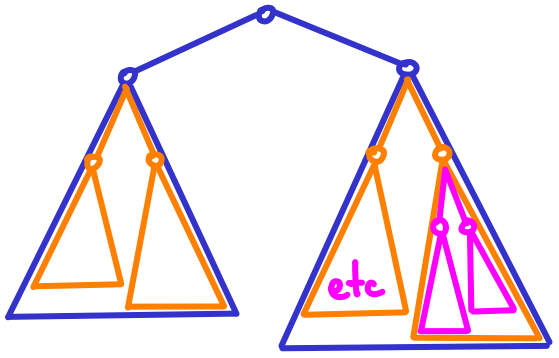
Fibonacci

$$F(h) = F(h-2) + F(h-1)$$

$$F(0) = F(1) = 1$$

$$N(h) > 2N(h-2) = \Omega(2^{h/2})$$

$\hookrightarrow h < 2\log_2 N$



$$N(h) = 1 + N(h-2) + N(h-1)$$

$N(1) = 1$ $\quad$ $N(2) = 2$

Fibonacci

$$F(h) = F(h-2) + F(h-1)$$

$$F(0) = F(1) = 1$$

$N() > F()$

$$N(h) > \frac{\phi^h}{\sqrt{5}}$$

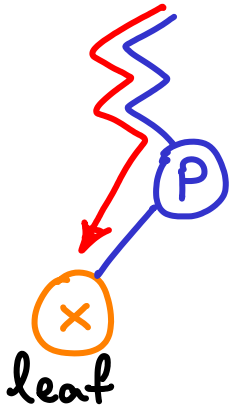
$$h < \log_\phi N + O(1)$$

$$\sim 1.44 \cdot \log_2 N$$

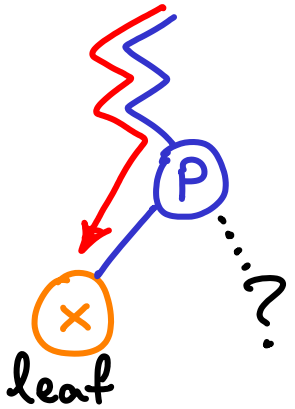
$$N(h) > 2N(h-2) = \Omega(2^{h/2})$$

$$\hookrightarrow h < 2\log_2 N$$

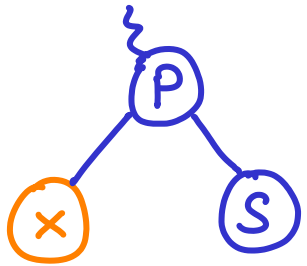
INSERTION IN AVL



INSERTION IN AVL

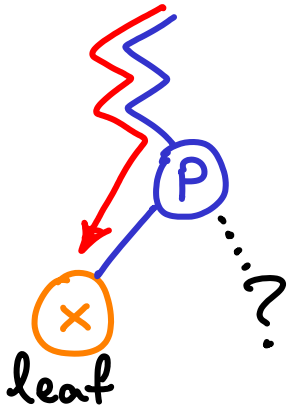


? is there a sibling?

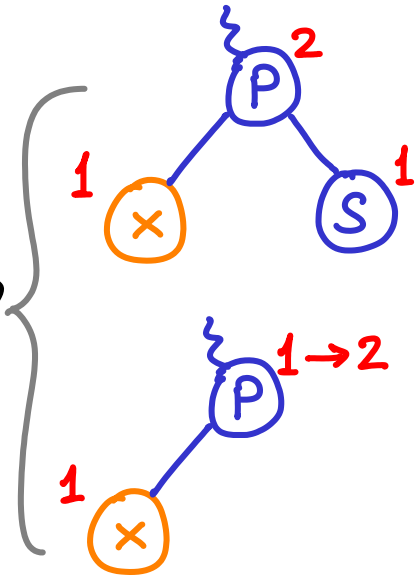


If yes then ?

INSERTION IN AVL



is there a sibling?

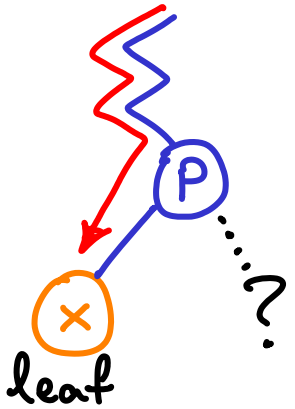


If yes, then done

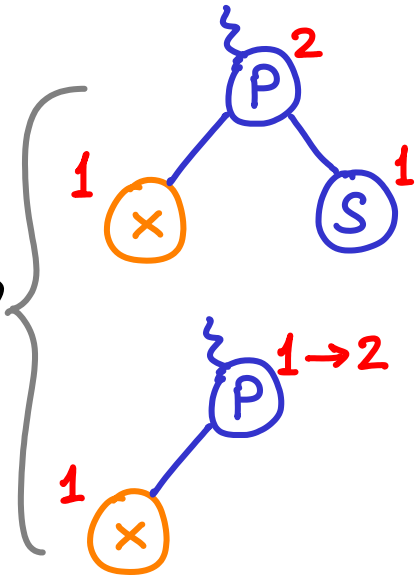
If not, parent is still happy

but other ancestors might not be

INSERTION IN AVL



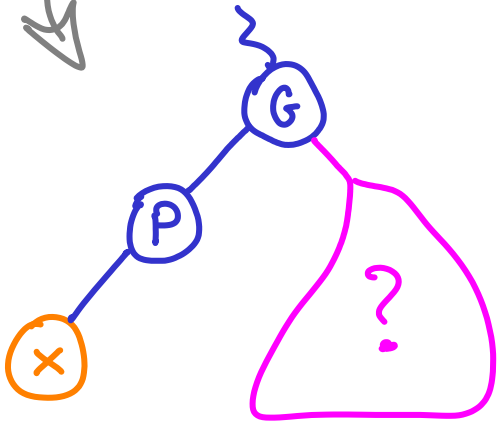
is there a sibling?



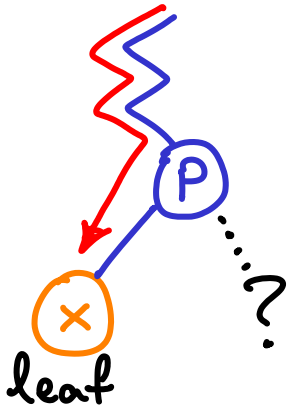
If yes, then done

If not, parent is still happy

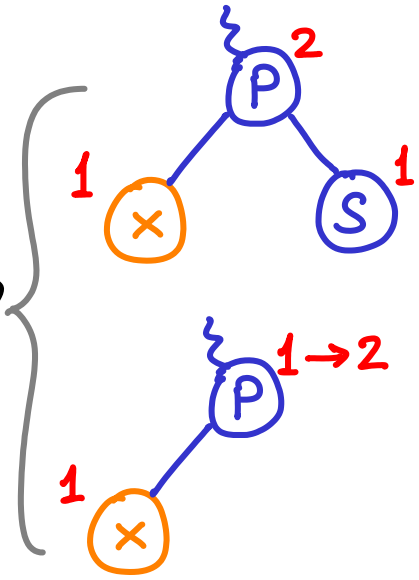
but other ancestors might not be



INSERTION IN AVL



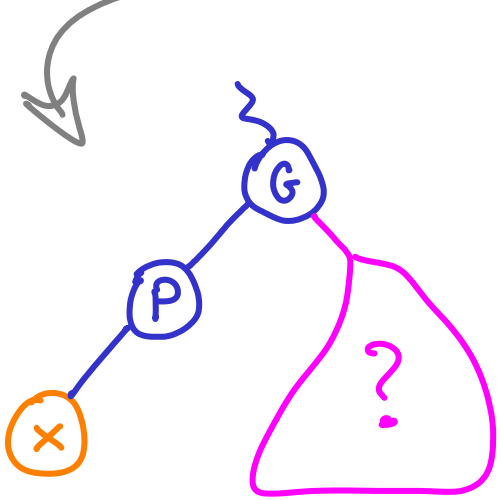
is there a sibling?

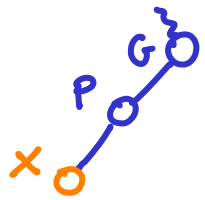


If yes, then done

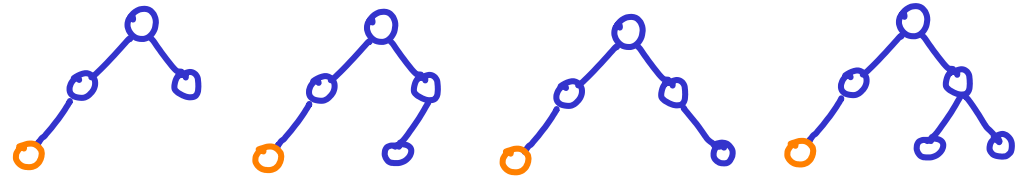
If not, parent is still happy

but other ancestors might not be

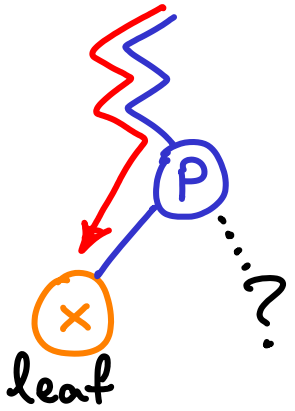




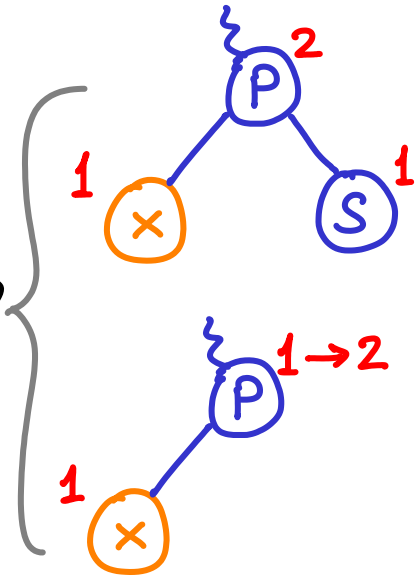
OR



INSERTION IN AVL



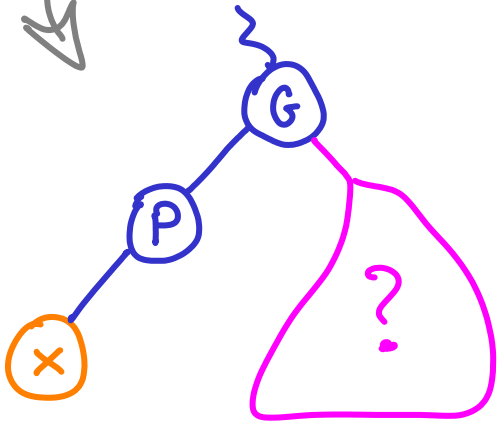
is there a sibling?

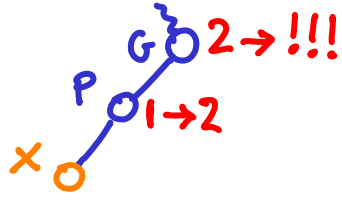


If yes, then done

If not, parent is still happy

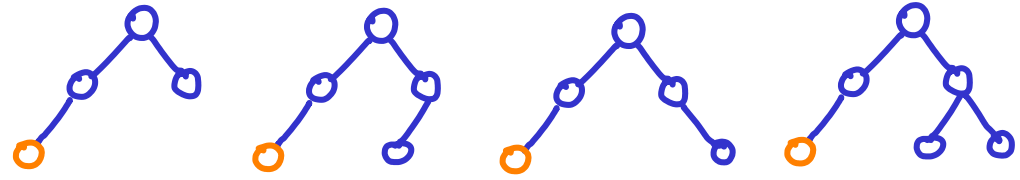
but other ancestors might not be



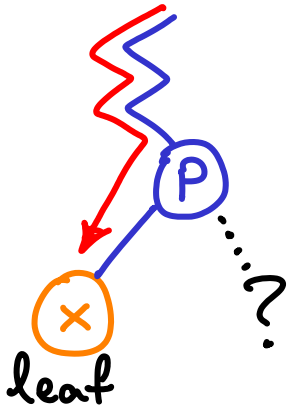


?

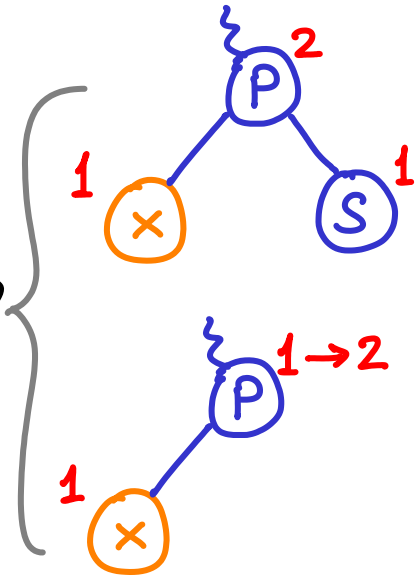
OR



INSERTION IN AVL



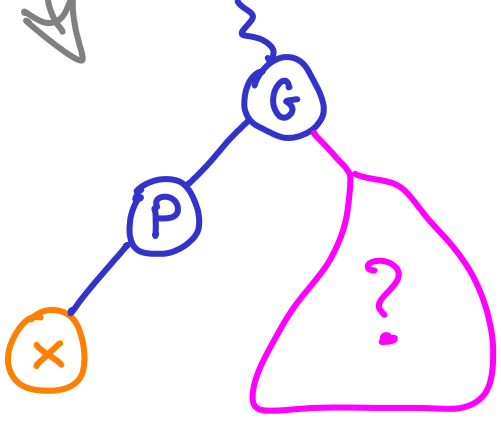
is there a sibling?

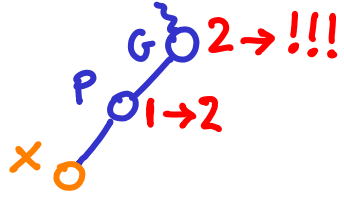


If yes, then done

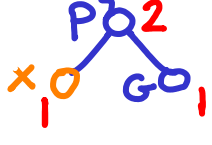
If not, parent is still happy

but other ancestors might not be



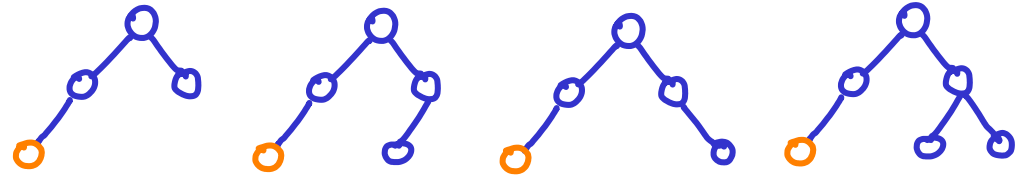


Right-rotate(G)



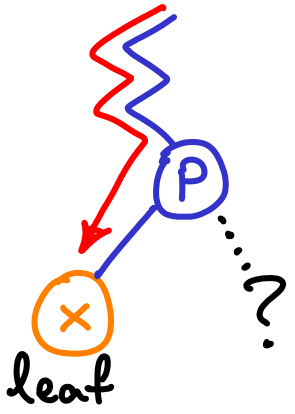
DONE

OR

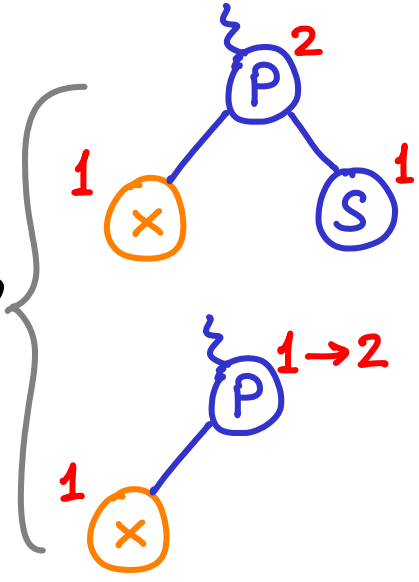


?

INSERTION IN AVL



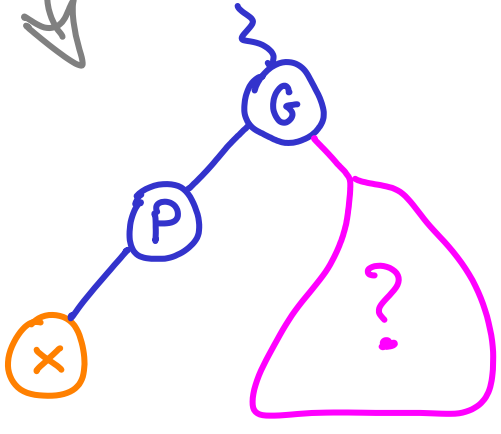
is there a sibling?

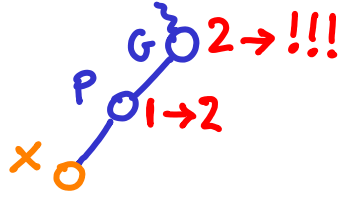


If yes, then done

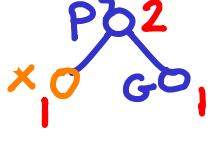
If not, parent is still happy

but other ancestors might not be



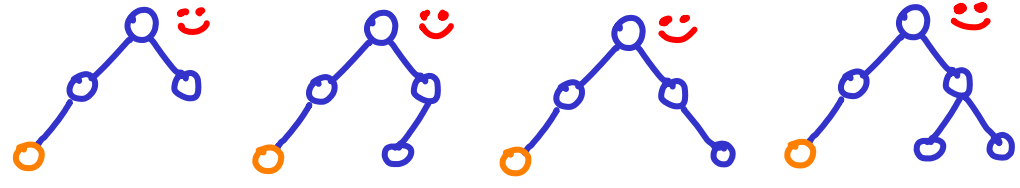


Right-rotate(G)



DONE

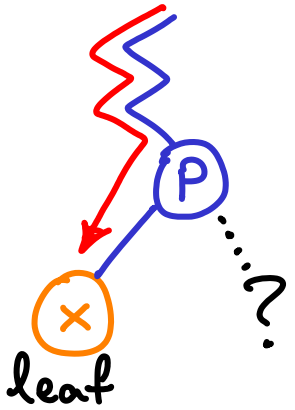
OR



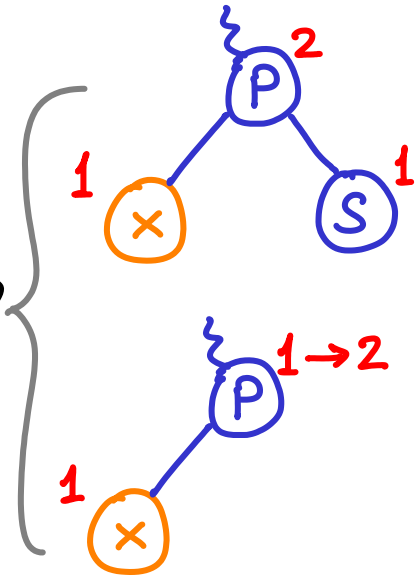
Now grandparent is happy

but other ancestors might not be

INSERTION IN AVL



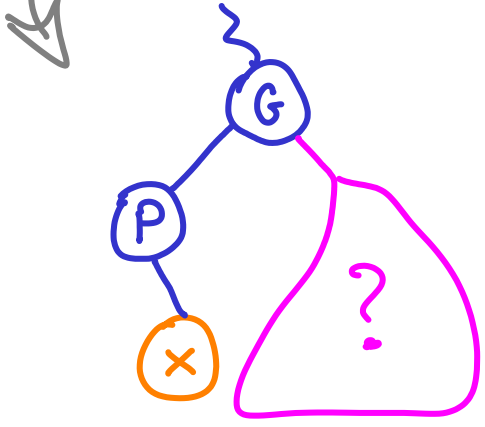
is there a sibling?

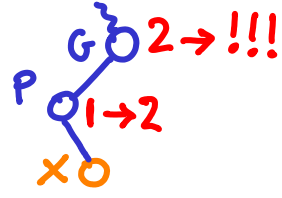


If yes, then done

If not, parent is still happy

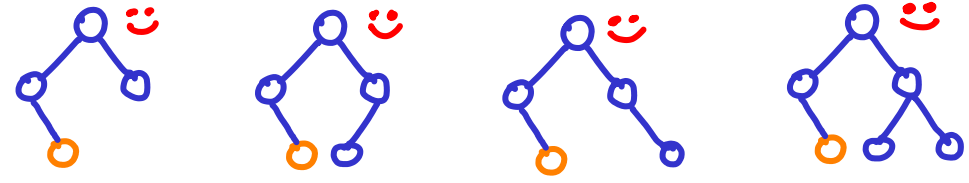
but other ancestors might not be





?

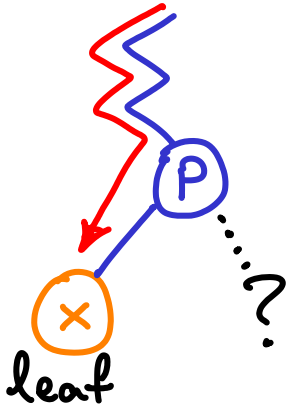
OR



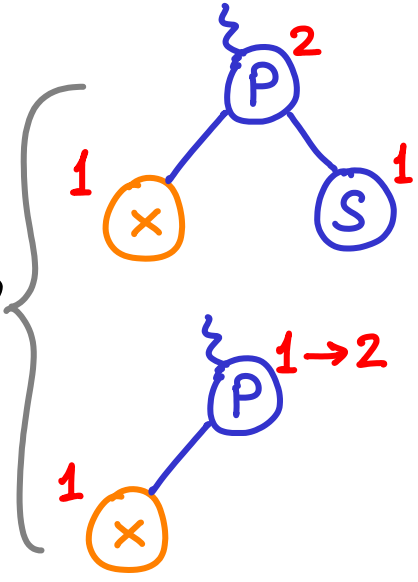
Now grandparent is happy

but other ancestors might not be

INSERTION IN AVL



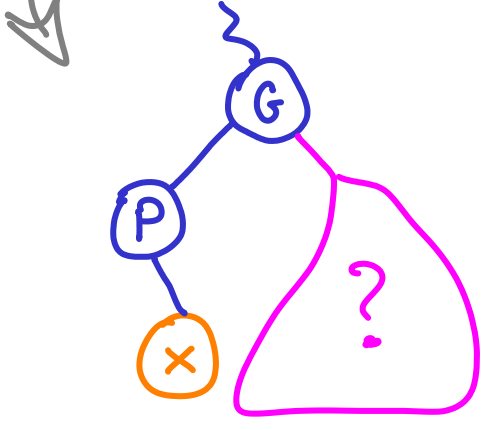
is there a sibling?

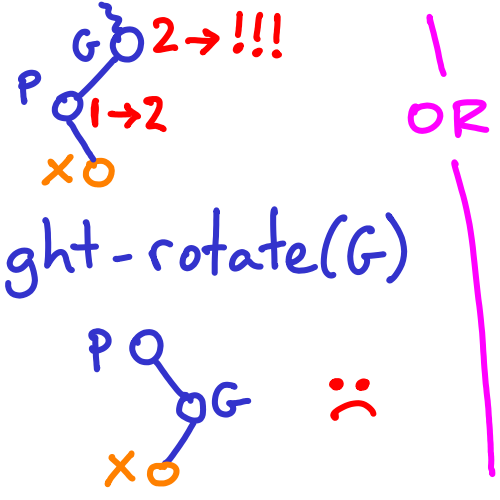


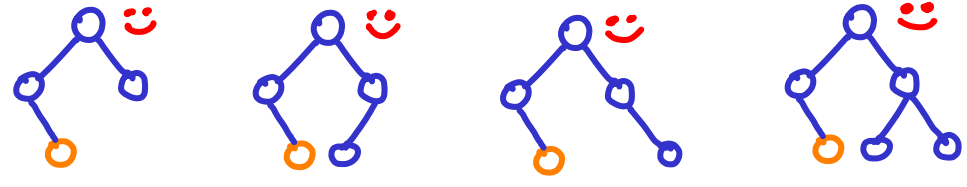
If yes, then done

If not, parent is still happy

but other ancestors might not be



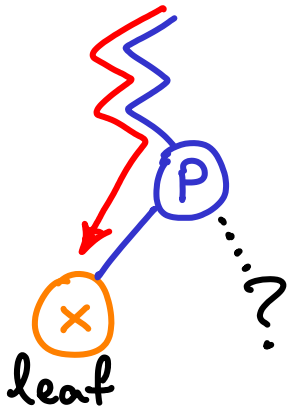




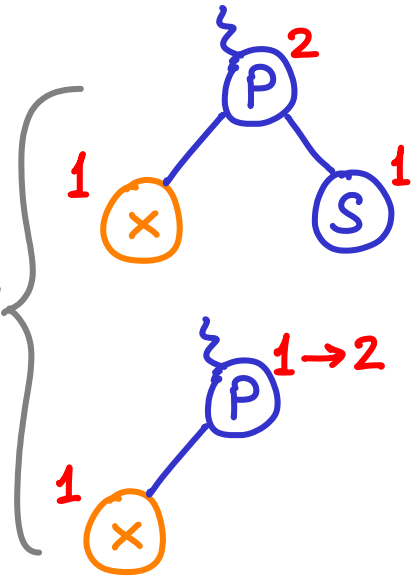
Now grandparent is happy

but other ancestors might not be

INSERTION IN AVL



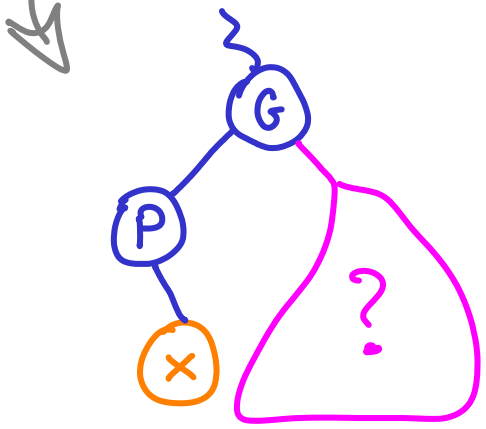
is there a sibling?

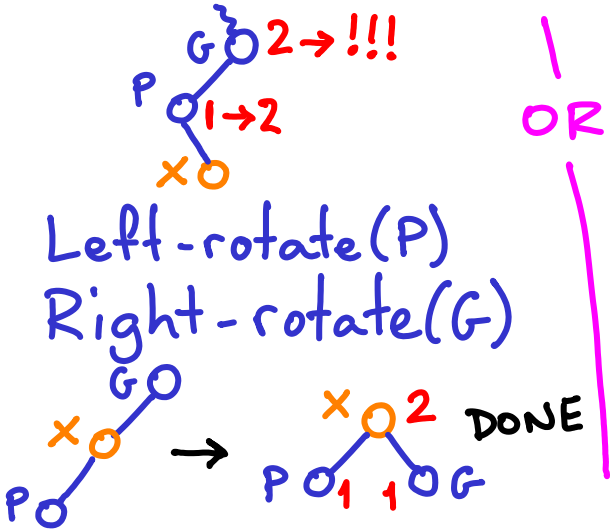


If yes, then done

If not, parent is still happy

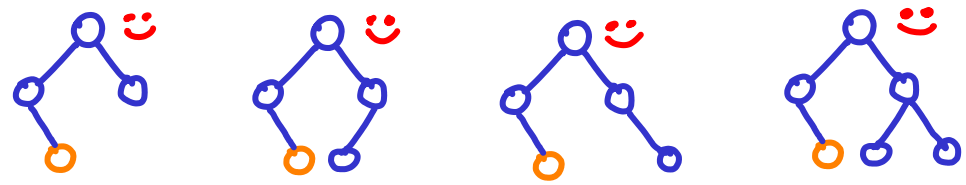
but other ancestors might not be





OR

DONE

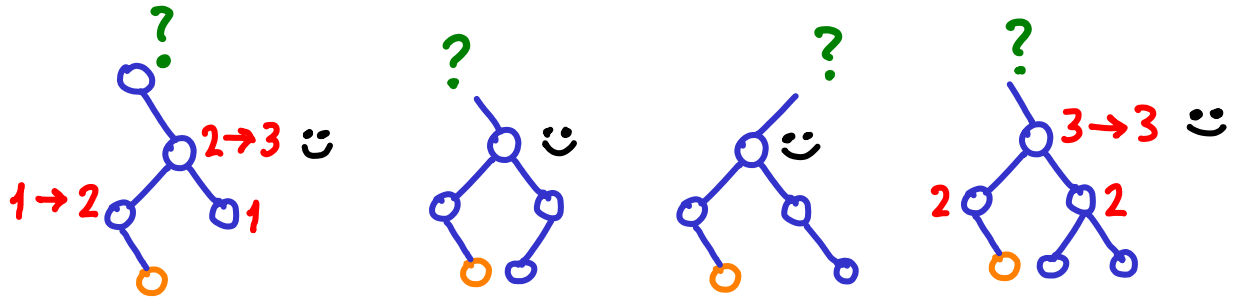


Now grandparent is happy

but other ancestors might not be

INSERTION IN AVL

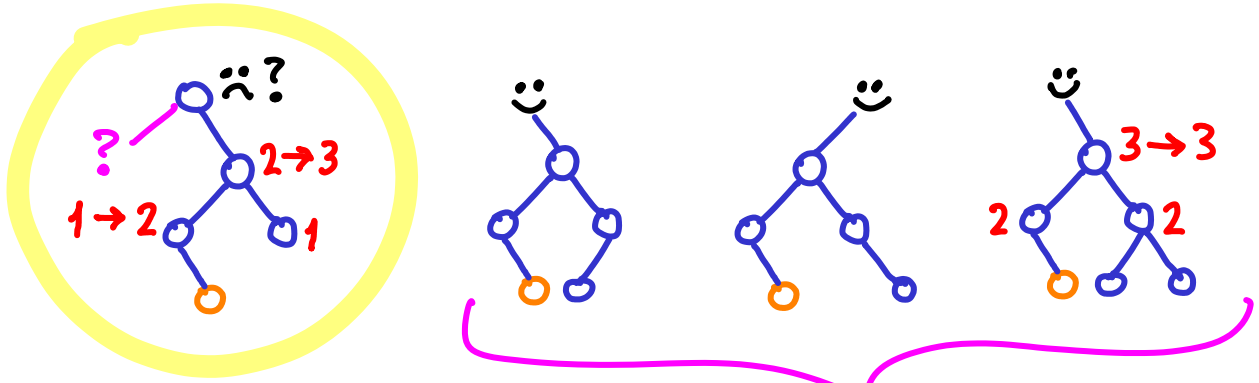
If not done, grandparent is happy but other ancestors might not be



Who's happy?

INSERTION IN AVL

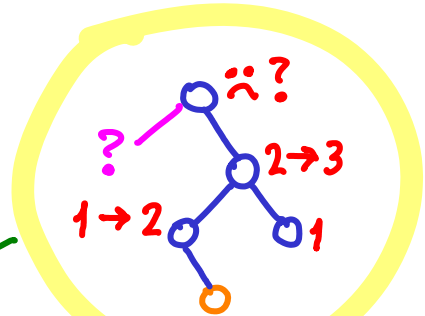
If not done, grandparent is happy but other ancestors might not be

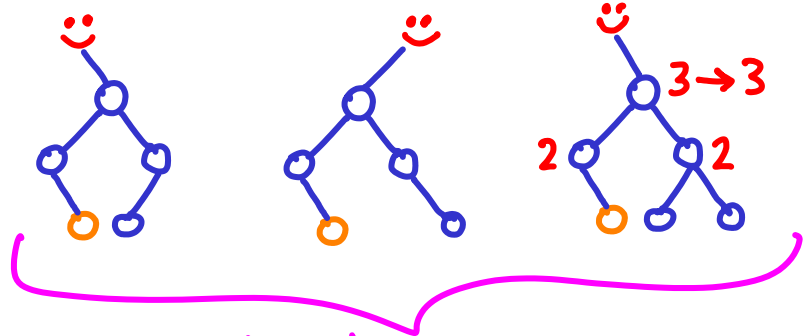


actually OK No score changes: DONE

INSERTION IN AVL

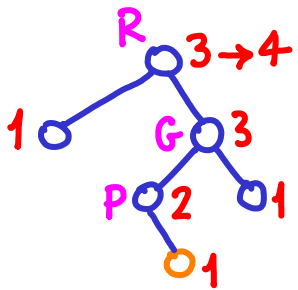
If not done, grandparent is happy but other ancestors might not be



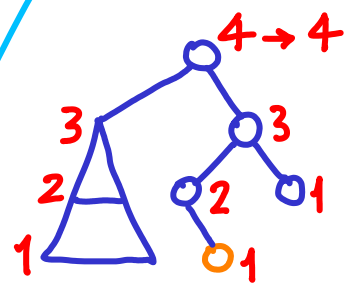


actually OK No score changes: DONE

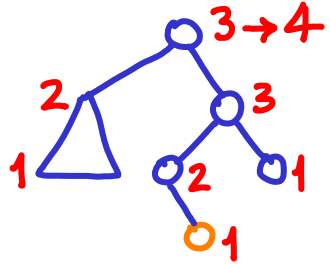
case 1



case 2

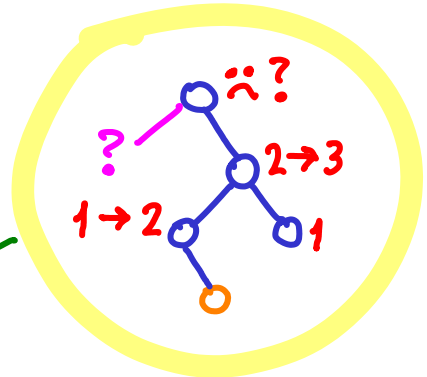


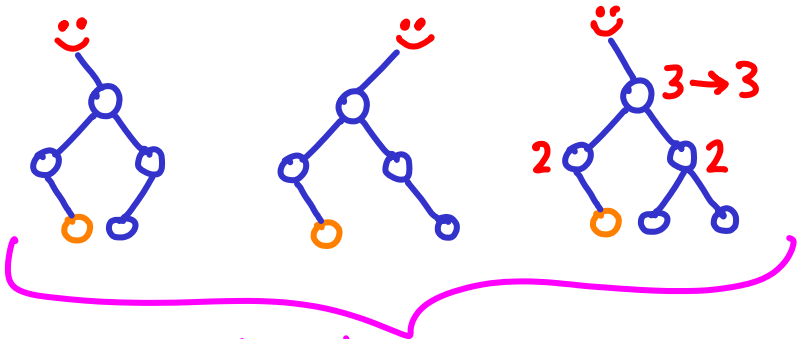
case3



INSERTION IN AVL

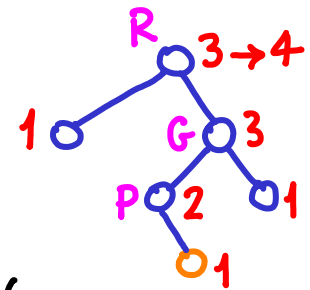
If not done, grandparent is happy but other ancestors might not be



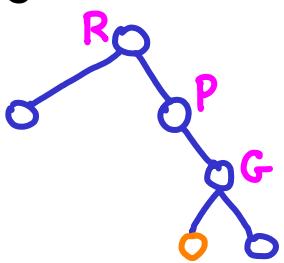


actually OK No score changes: DONE

Case 1

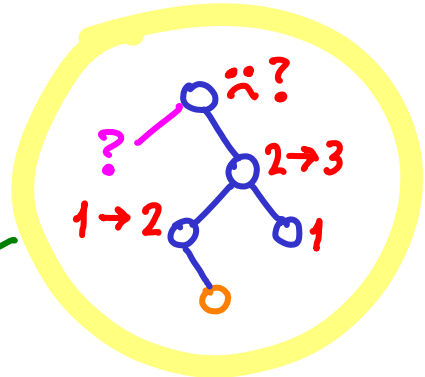


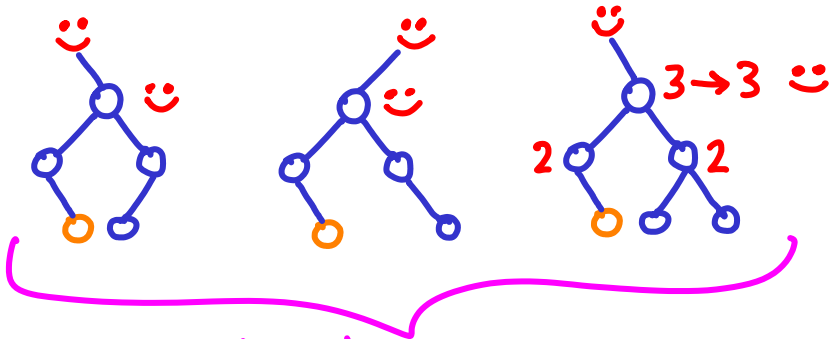
Right-rotate(G)



INSERTION IN AVL

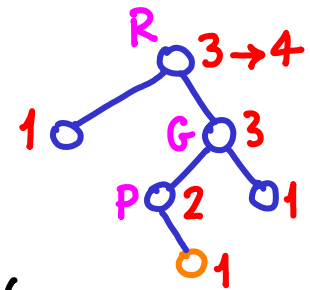
If not done, grandparent is happy but other ancestors might not be



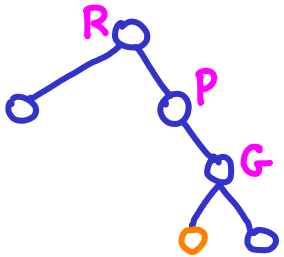


actually OK No score changes: DONE

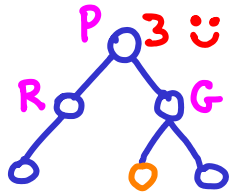
Case 1



(Right-rotate(G)



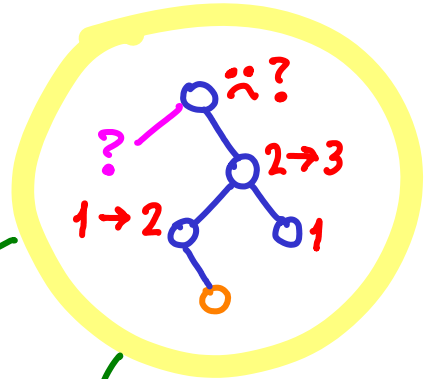
Left-rotate(R)

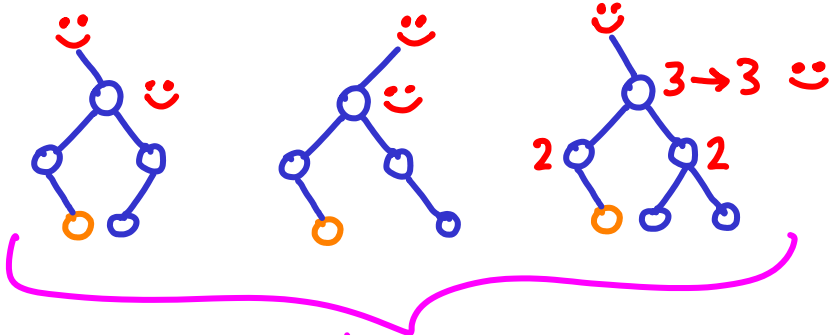


DONE

INSERTION IN AVL

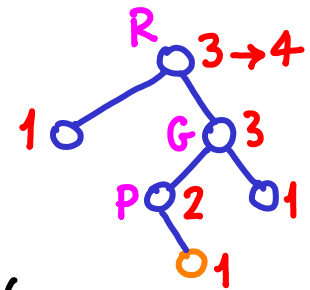
If not done, grandparent is happy but other ancestors might not be



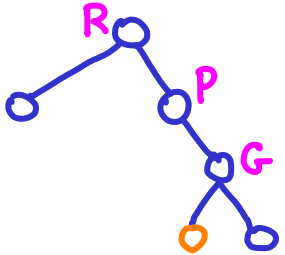


actually OK No score changes: DONE

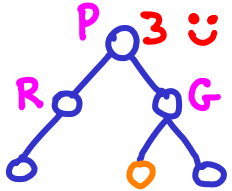
Case 1



(Right-rotate(G)

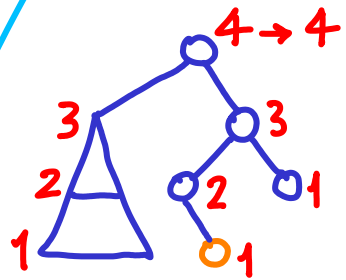


Left-rotate(R)



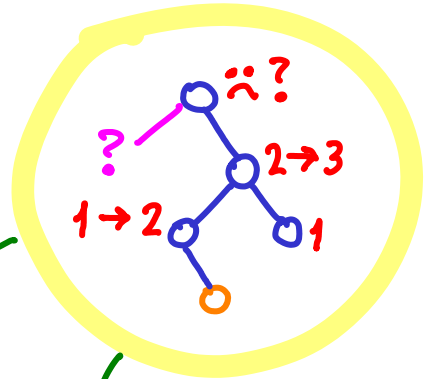
DONE

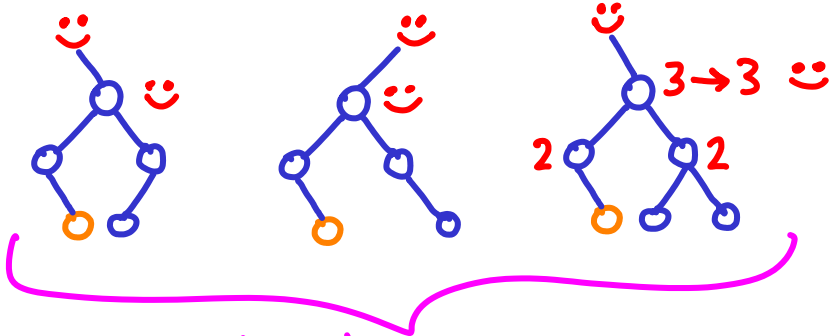
case 2



INSERTION IN AVL

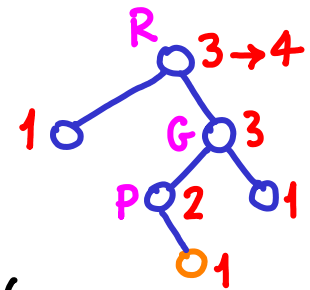
If not done, grandparent is happy but other ancestors might not be



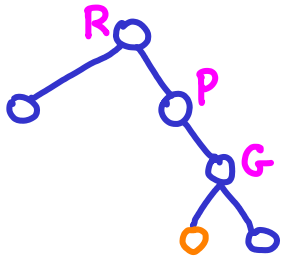


actually OK No score changes: DONE

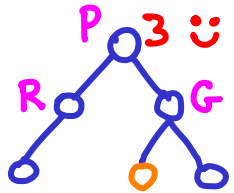
Case 1



(Right-rotate(G)

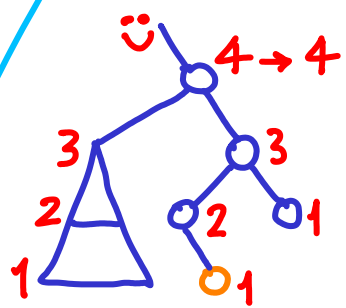


Left-rotate(R)



DONE

case 2

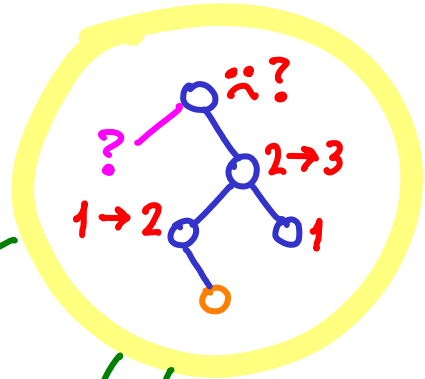


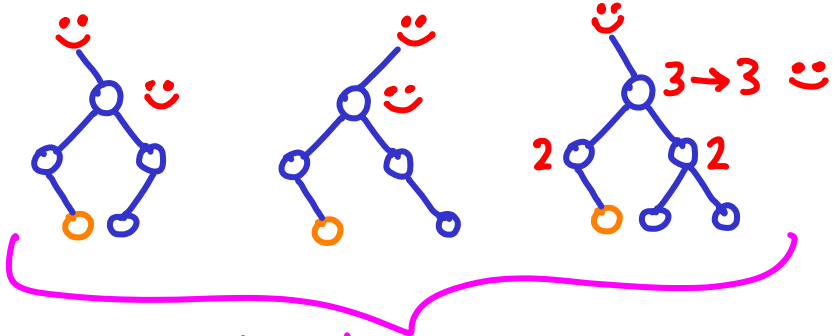
Same as above.

DONE

INSERTION IN AVL

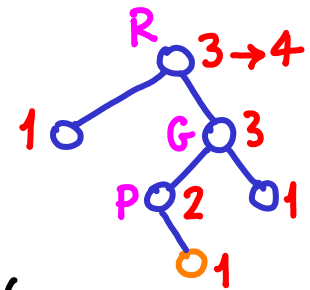
If not done, grandparent is happy but other ancestors might not be



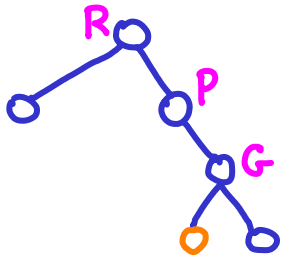


actually OK No score changes: DONE

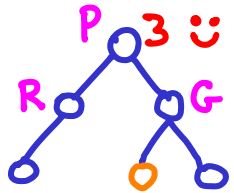
case 1



(Right-rotate(G)

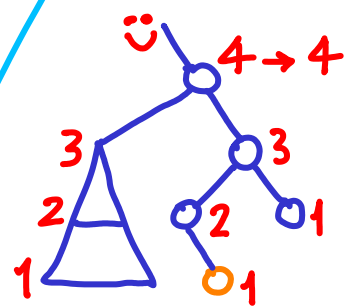


Left-rotate(R)



DONE

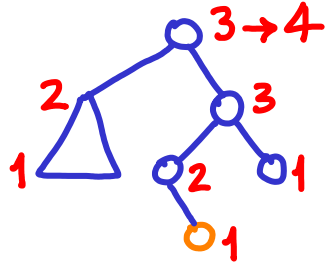
case 2



Same as above.

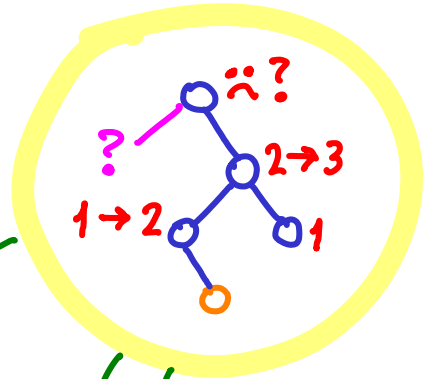
DONE

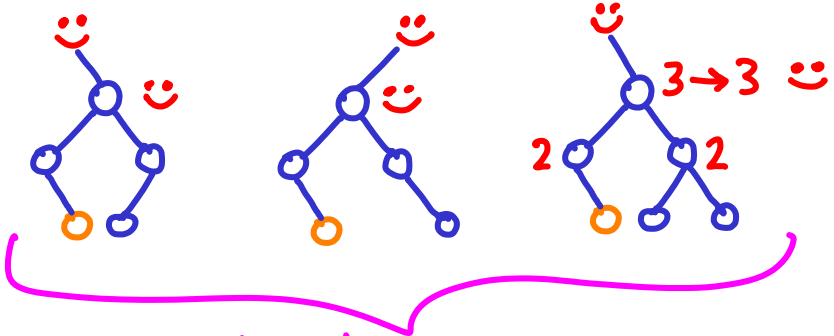
case3



INSERTION IN AVL

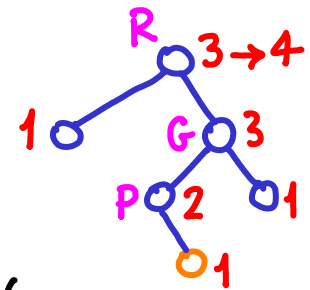
If not done, grandparent is happy but other ancestors might not be



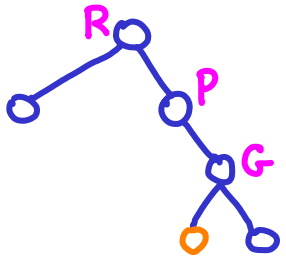


actually OK No score changes: DONE

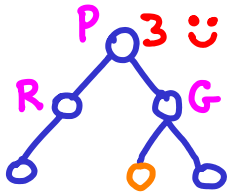
case 1



(Right-rotate(G)

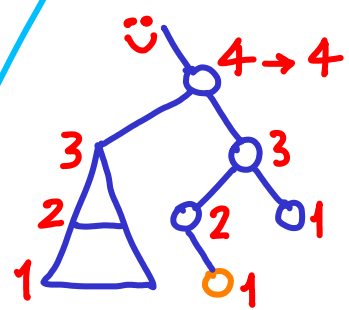


Left-rotate(R)



DONE

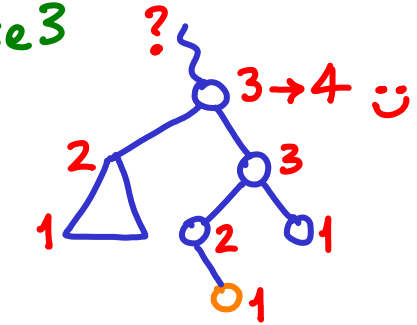
case 2



Same as above.

DONE

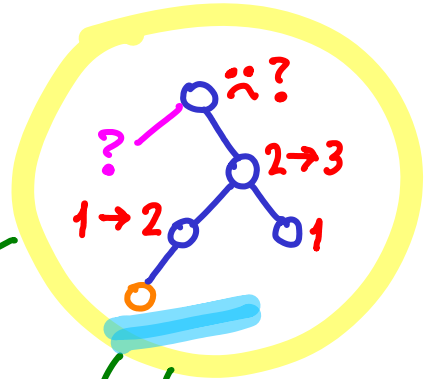
case3

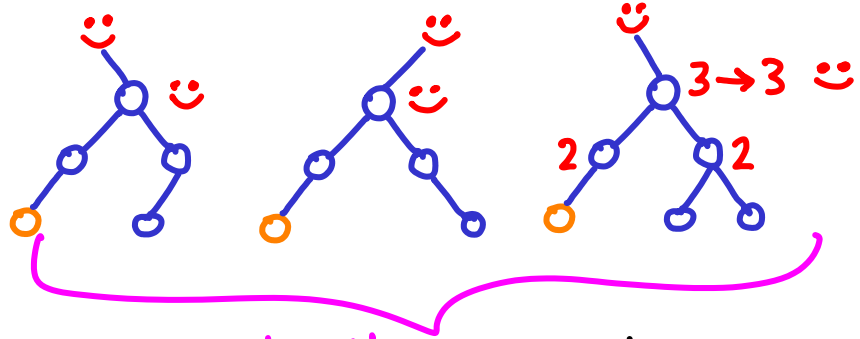


Keep searching up

INSERTION IN AVL

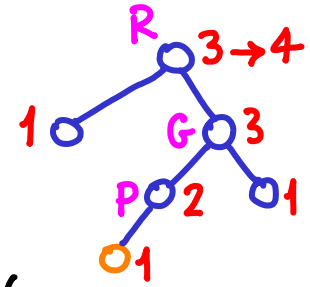
If not done, grandparent is happy but other ancestors might not be



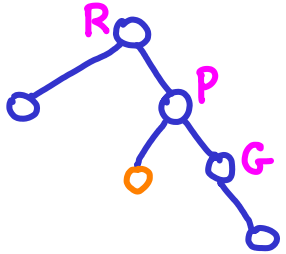


actually OK No score changes: DONE

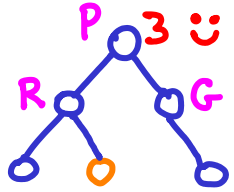
Case 1



(Right-rotate(G)

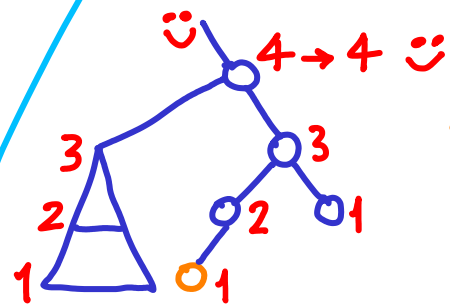


Left-rotate(R)



DONE

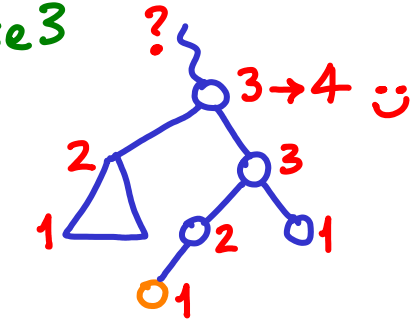
case 2



Same as above.

DONE

case3



Keep searching up

INSERTION IN AVL

In general, walk up ancestor path.

If any ancestor is ±1 balanced and has original score (height) then DONE.

INSERTION IN AVL

In general, walk up ancestor path.

If any ancestor is ±1 balanced and has original score (height) then DONE.

↳ if ±1 balanced but score went up, keep searching

search ends at root, $O(\log n)$ time

INSERTION IN AVL

In general, walk up ancestor path.

If any ancestor is ±1 balanced and has original score (height) then DONE.

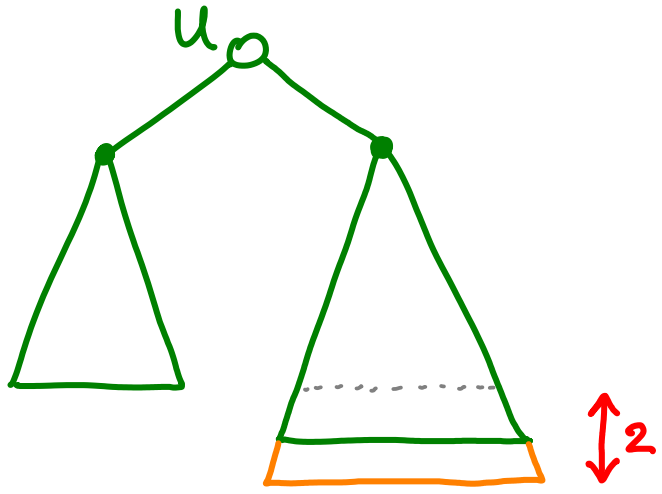
↳ if imbalance > 1, fix & stop.

↳ if ±1 balanced but score went up, keep searching

search ends at root, $O(\log n)$ time

Otherwise,

u = first really unbalanced ancestor



INSERTION IN AVL

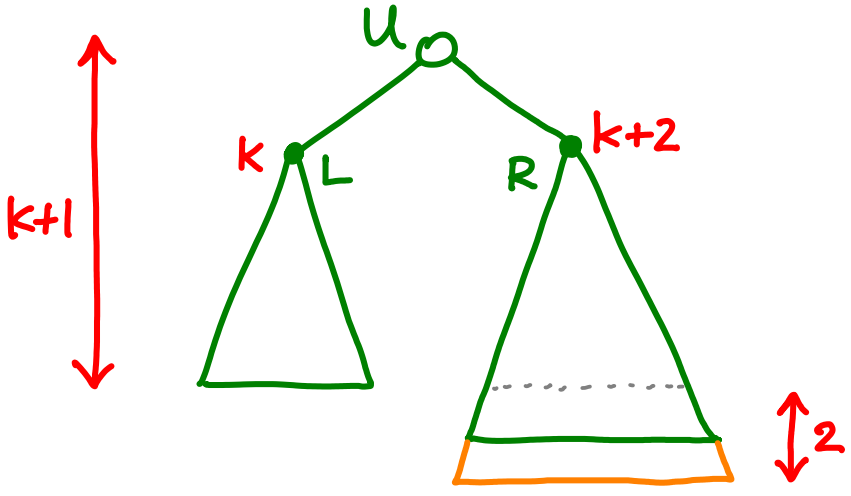
In general, walk up ancestor path.

If any ancestor is ±1 balanced and has original score (height) then DONE.

↳ if imbalance >1, fix & stop.

↳ if ±1 balanced but score went up, keep searching

search ends at root, $O(\log n)$ time



Otherwise,

u = first really unbalanced ancestor

u has one subtree with height k

& one with k+2: (w.l.o.g. right subtree, R)

INSERTION IN AVL

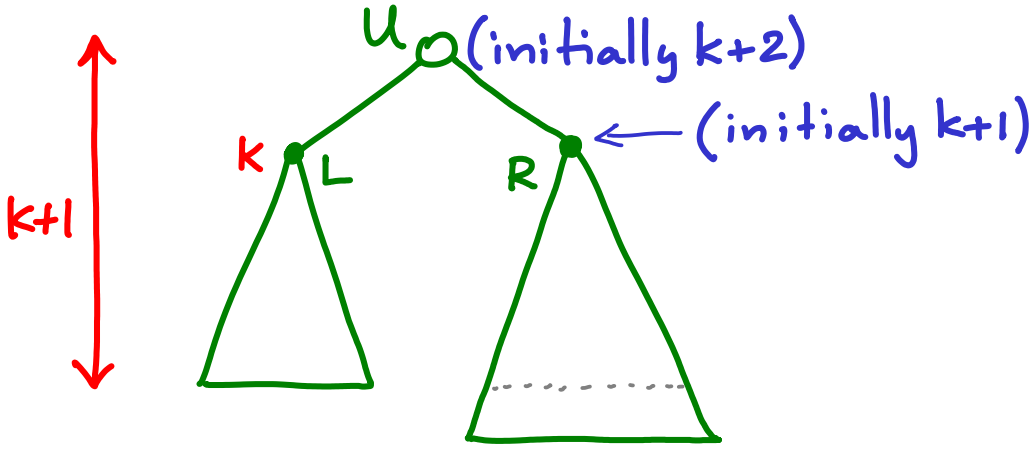
In general, walk up ancestor path.

If any ancestor is ±1 balanced and has original score (height) then DONE.

↳ if imbalance >1, fix & stop.

↳ if ±1 balanced but score went up, keep searching

search ends at root, O(log n) time



Otherwise,

U = first really unbalanced ancestor

U has one subtree with height k

& one with k+2: (w.l.o.g. right subtree, R)

INSERTION IN AVL

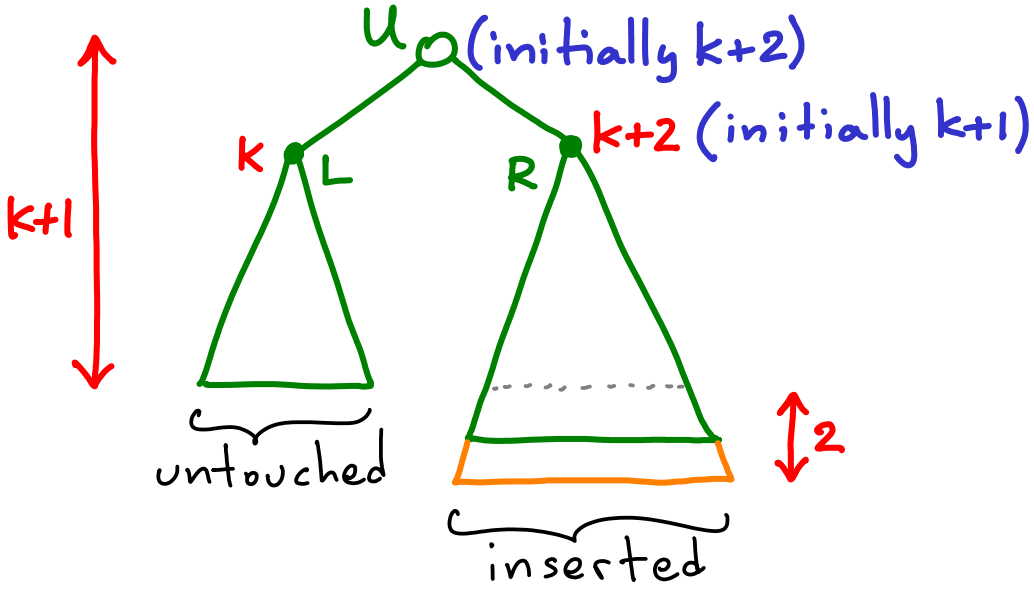
In general, walk up ancestor path.

If any ancestor is ±1 balanced and has original score (height) then DONE.

↳ if imbalance >1, fix & stop.

↳ if ±1 balanced but score went up, keep searching

search ends at root, O(log n) time

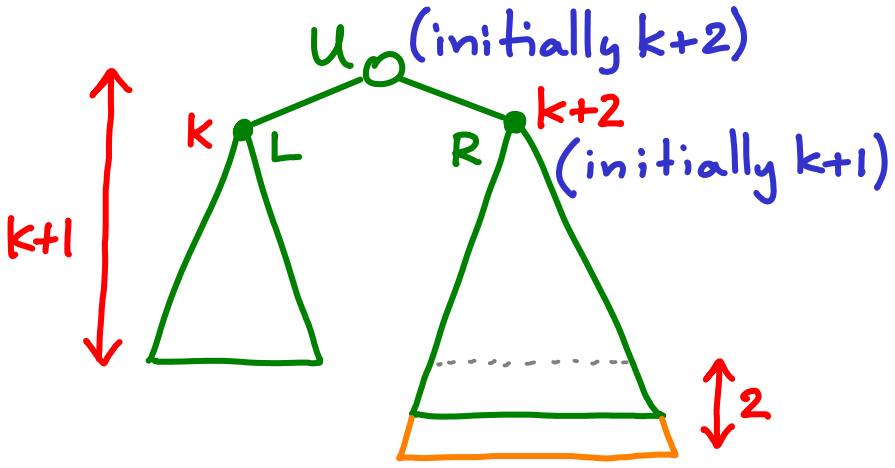


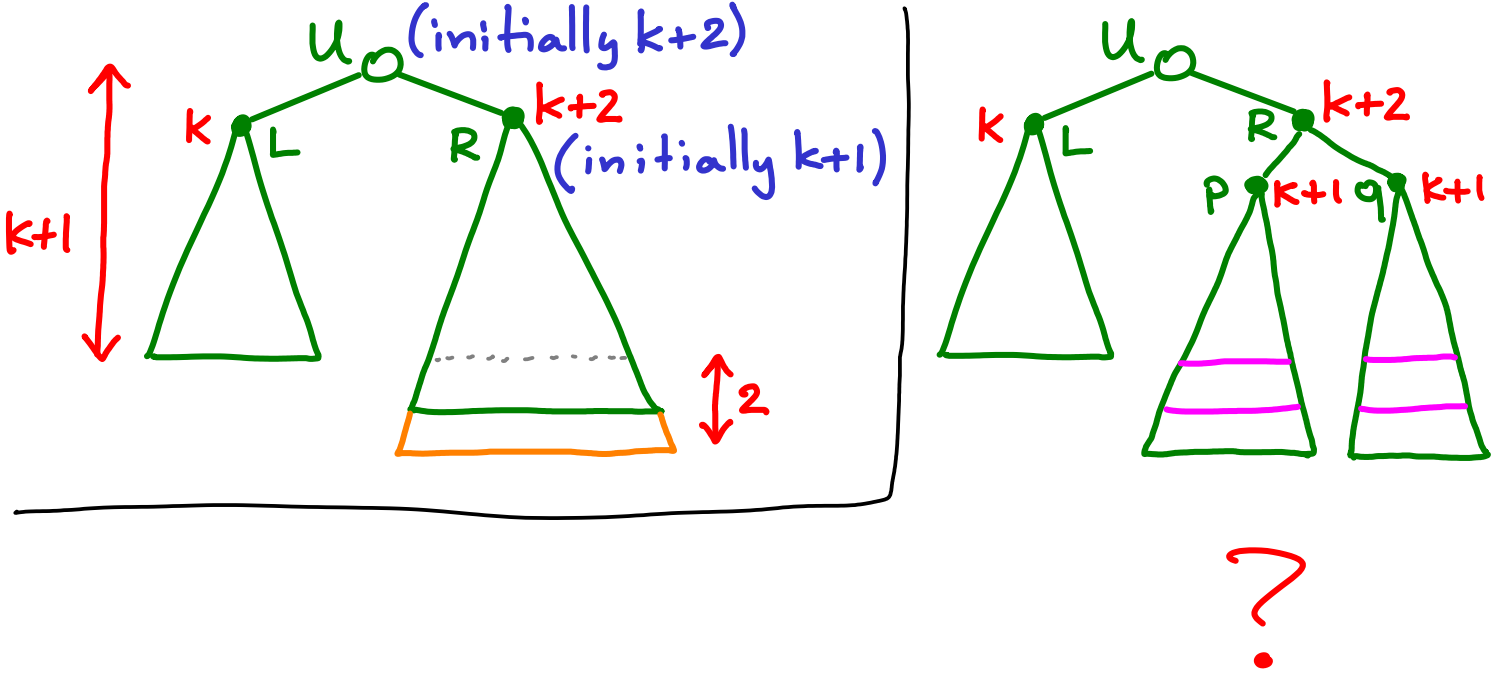
Otherwise,

u = first really unbalanced ancestor

u has one subtree with height k

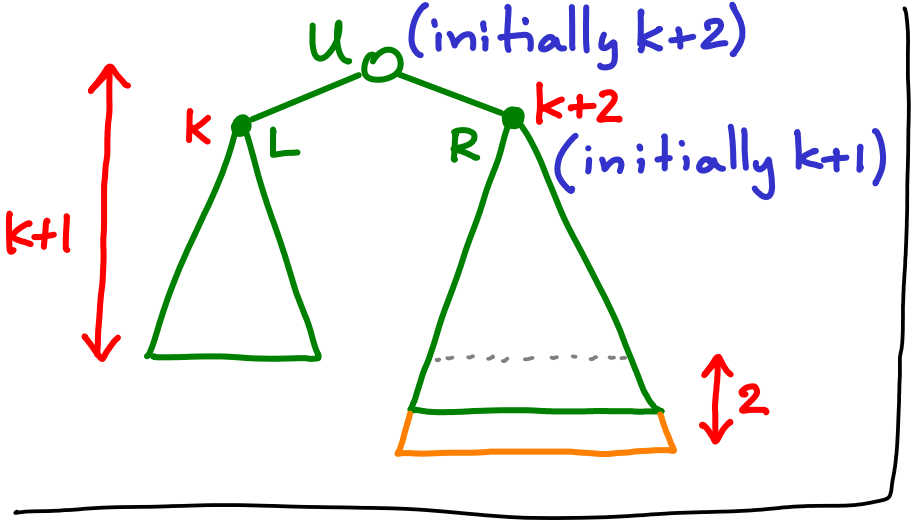
& one with k+2: (w.l.o.g. right subtree, R)

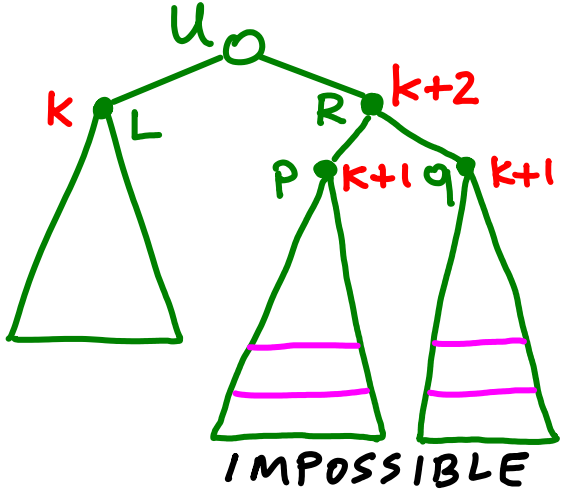

u
(initially k+2)
k
L
R
k+2
(initially k+1)
k+1
2


u
(initially k+2)
k
L
R
k+2
(initially k+1)
k+1
2

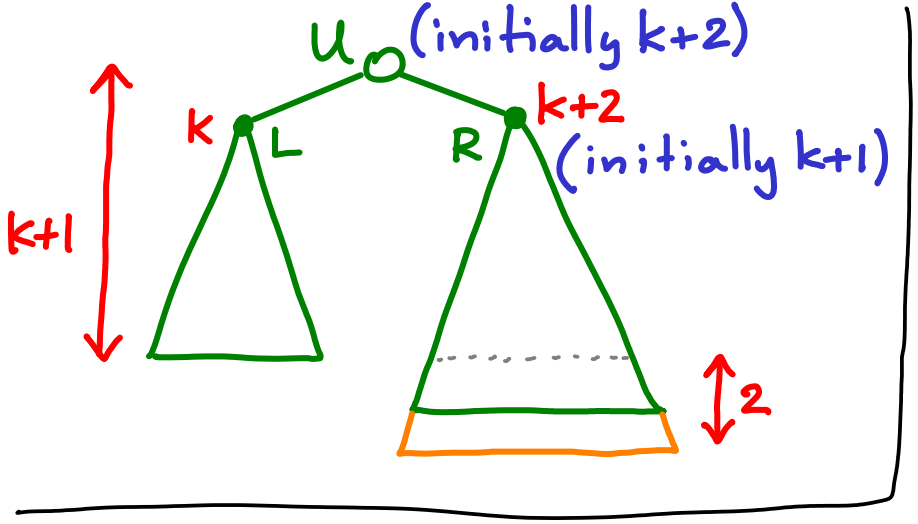
u
k
L
R
k+2
p
k+1
q
k+1

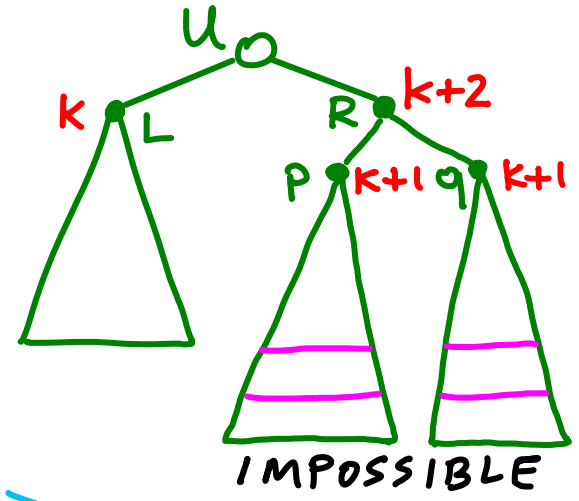
?





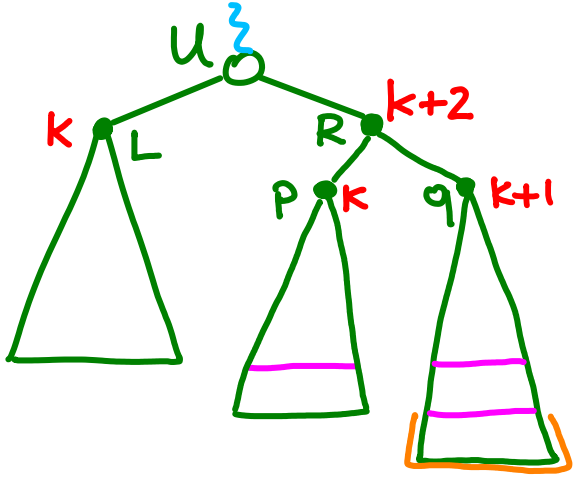
we haven't modified (rotated) anything yet, so R would have already had height k+2 via either p or q.



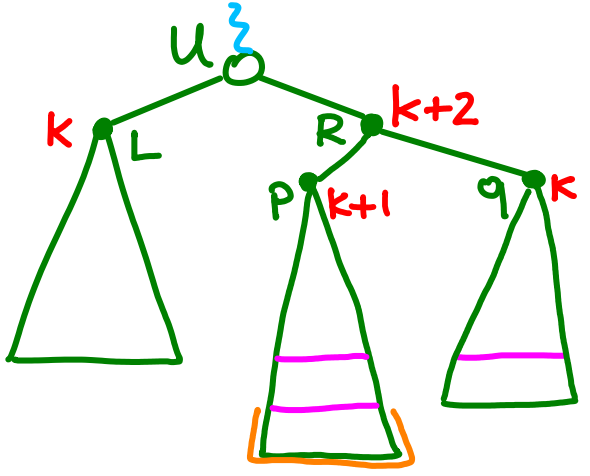


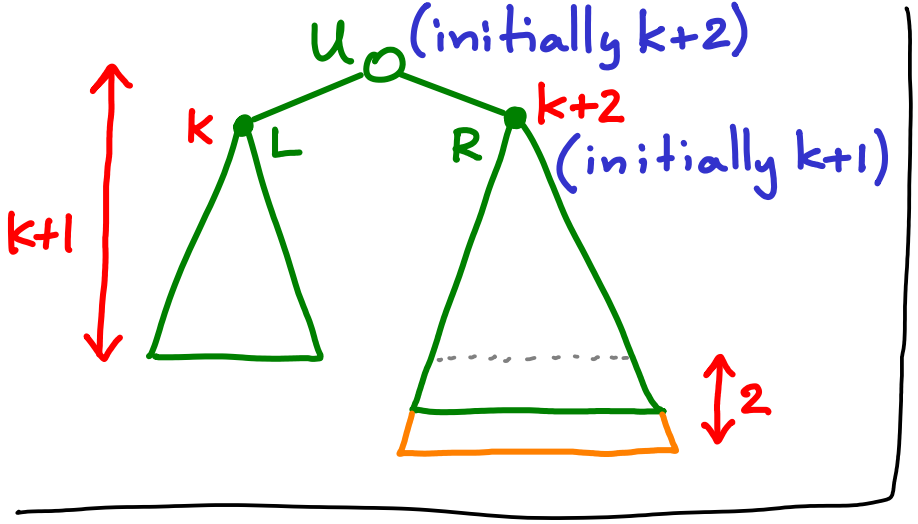
we haven't modified (rotated) anything yet, so R would have already had height k+2 via either p or q.

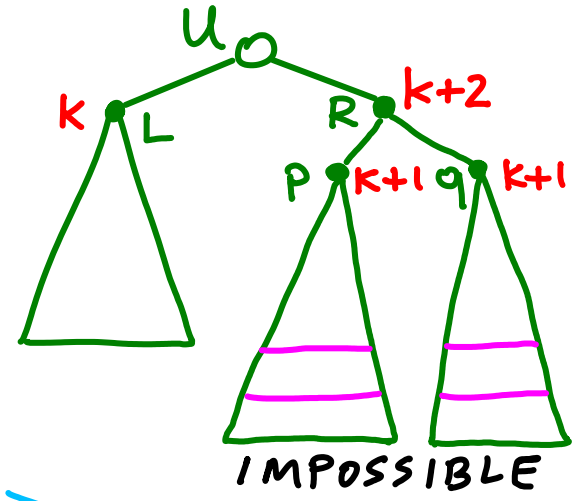
Case 1



Case 2

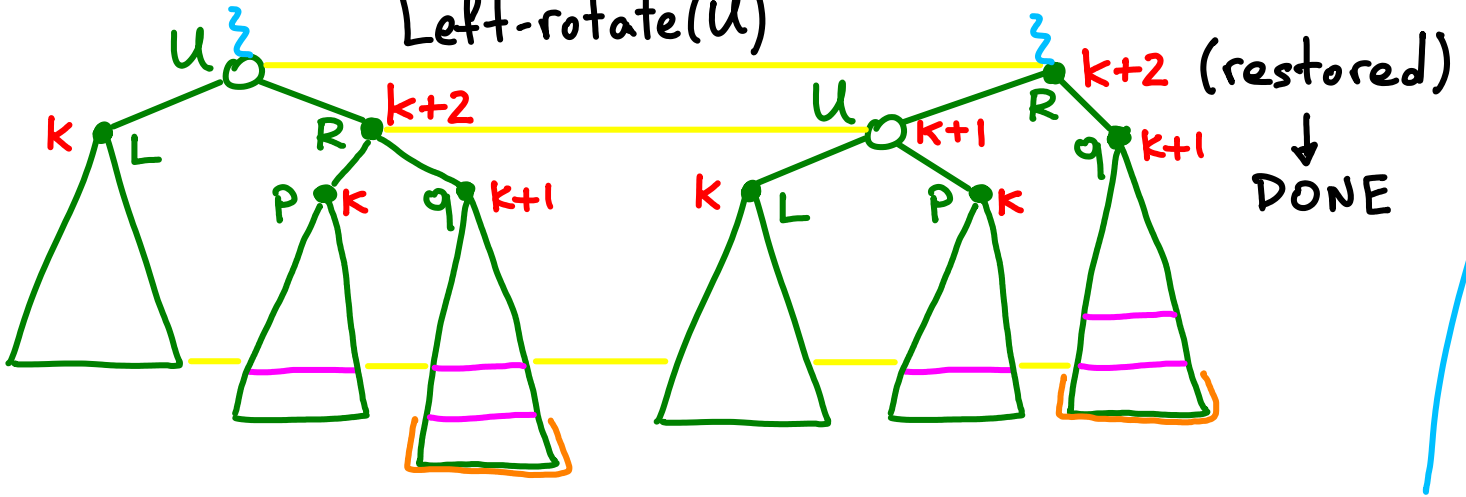




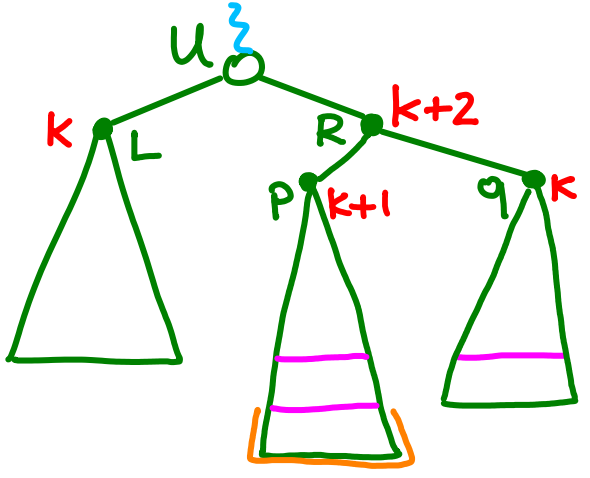


we haven't modified (rotated) anything yet, so R would have already had height k+2 via either p or q.

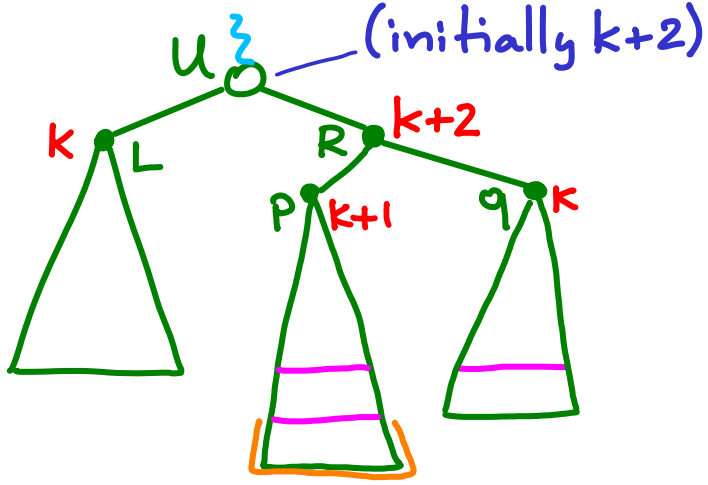
Case 1



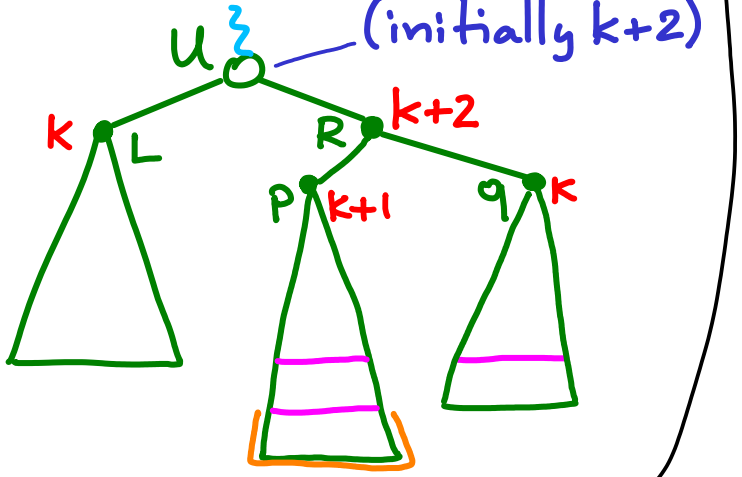
Case 2

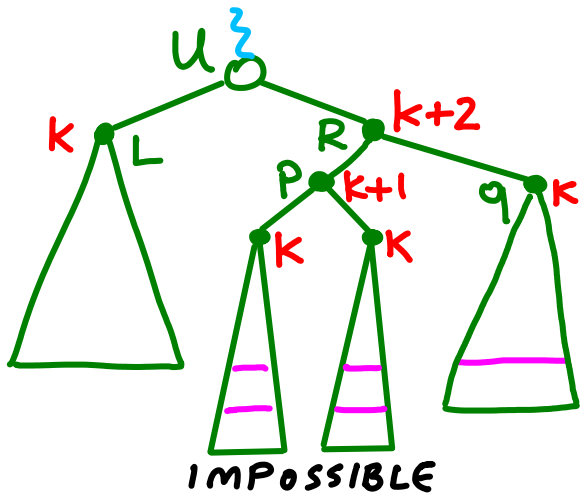


Case 2



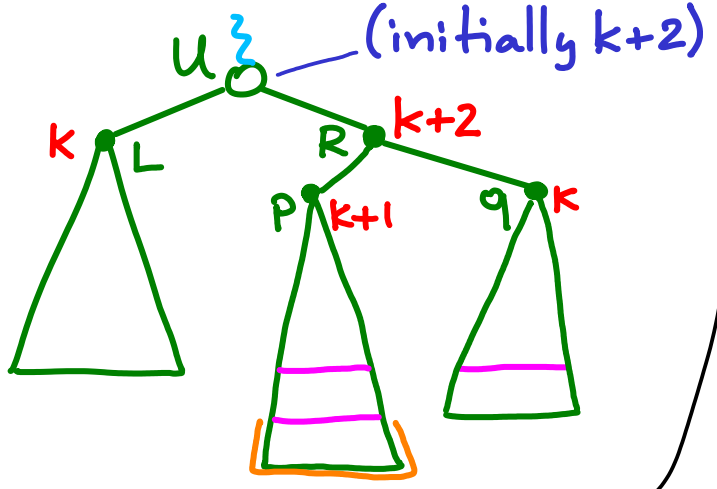
Case 2

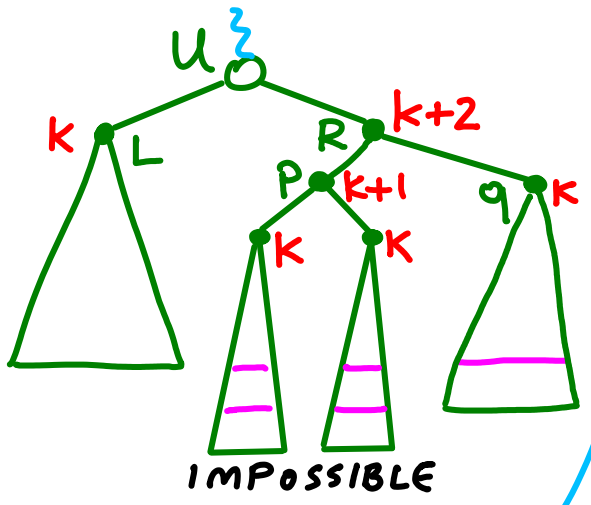




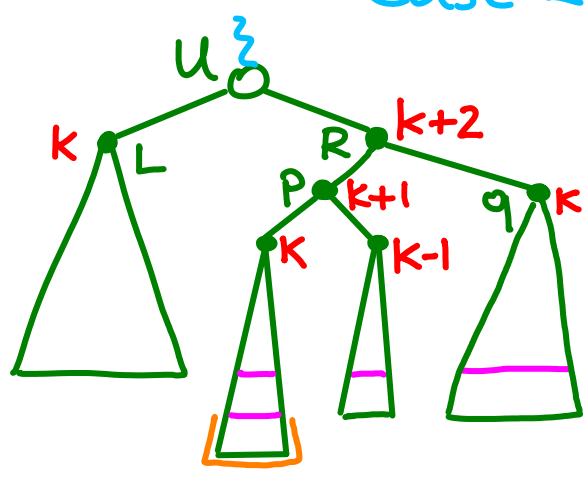
IMPOSSIBLE

Case 2

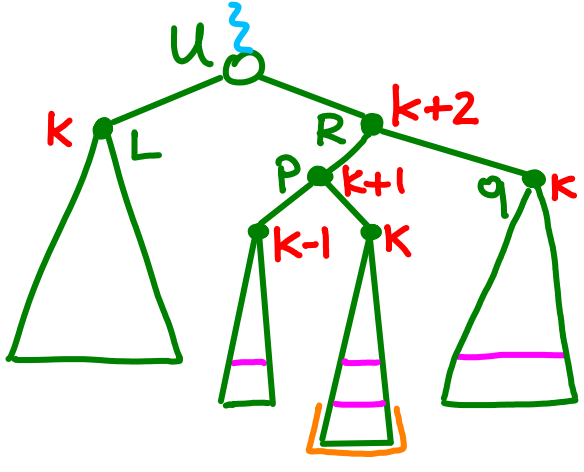




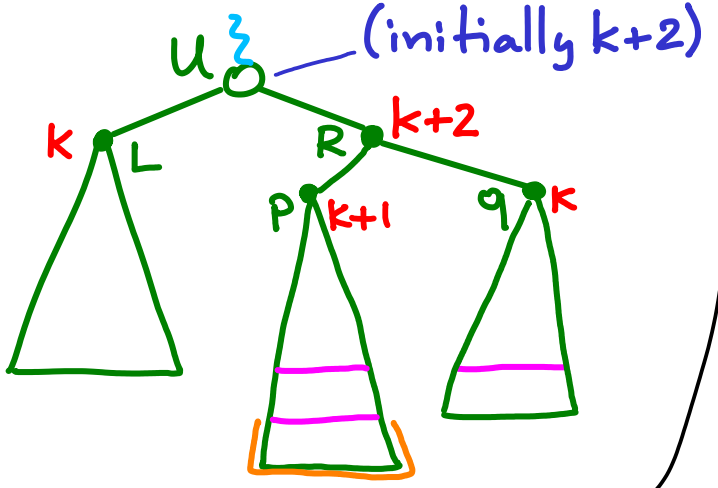
Case 2B

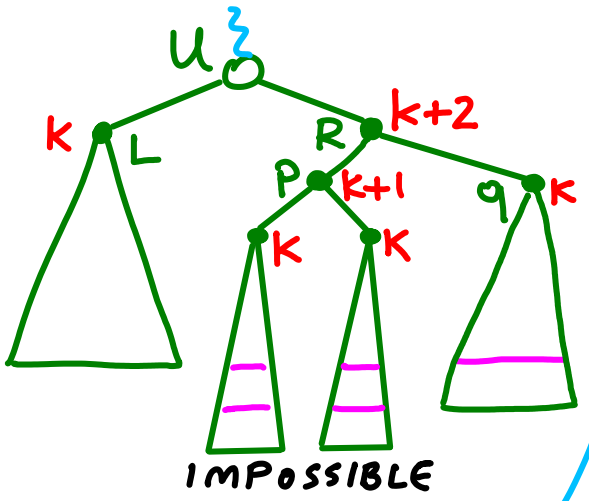


Case 2A



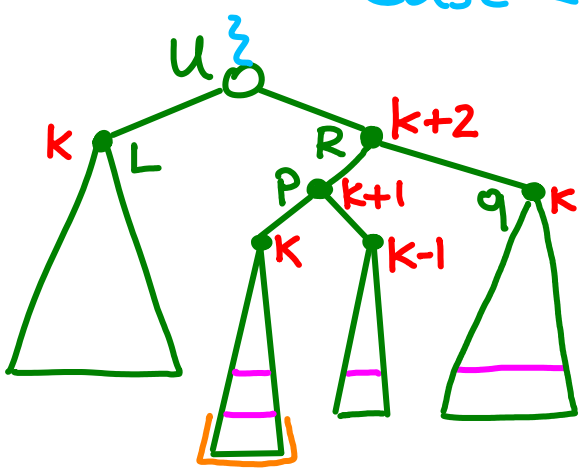
Case 2





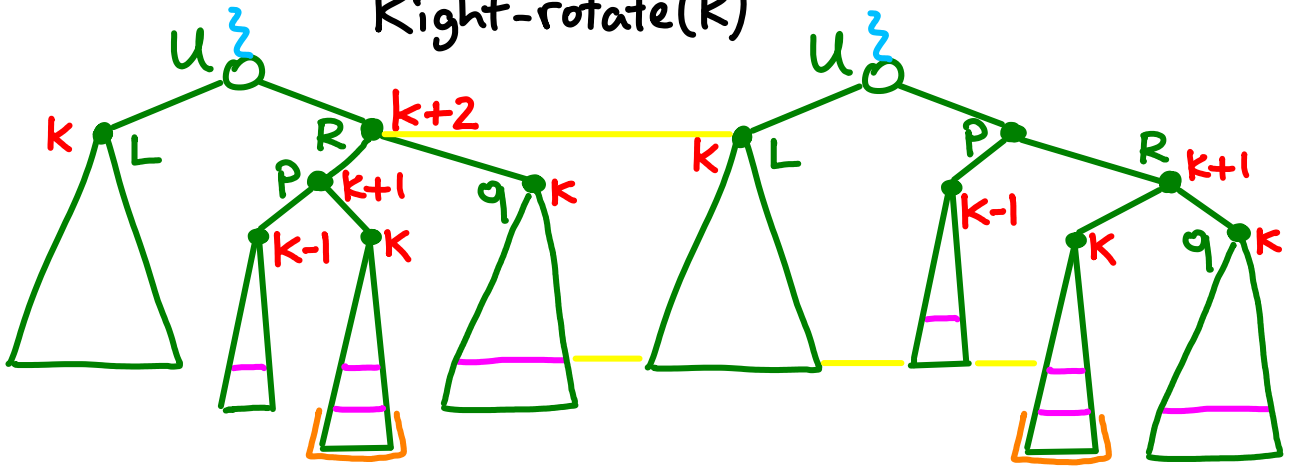
IMPOSSIBLE

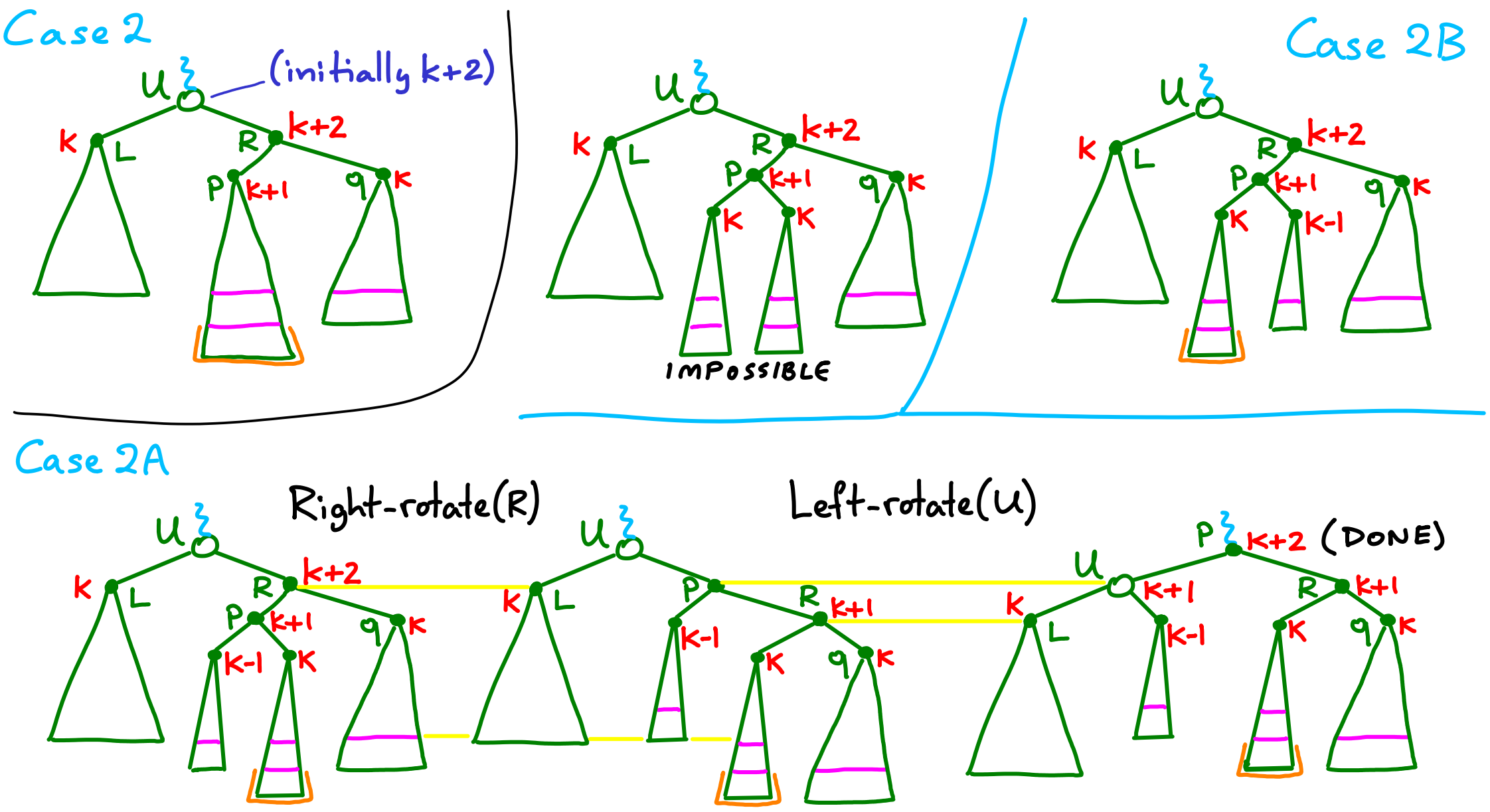
Case 2B

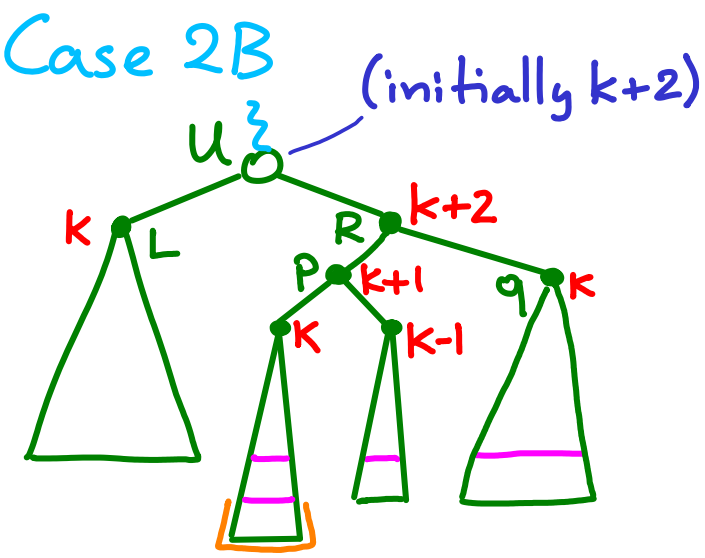


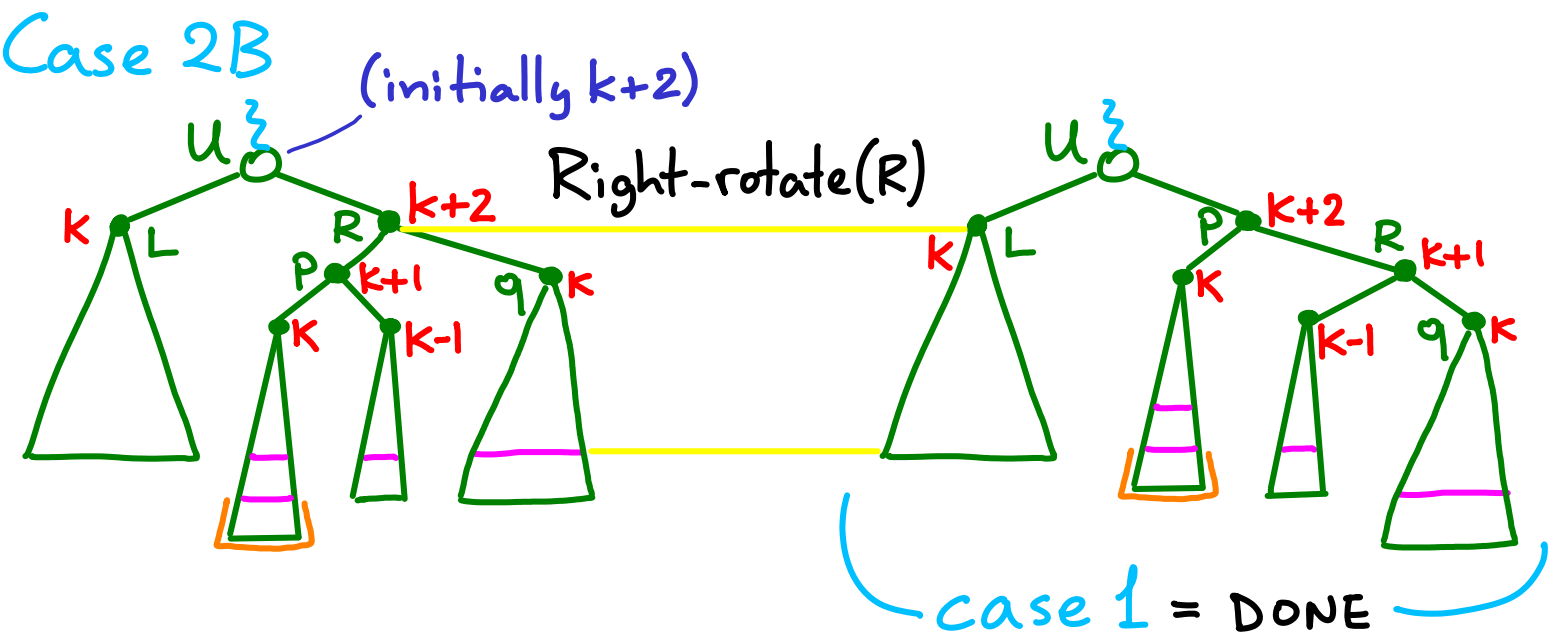
Case 2A

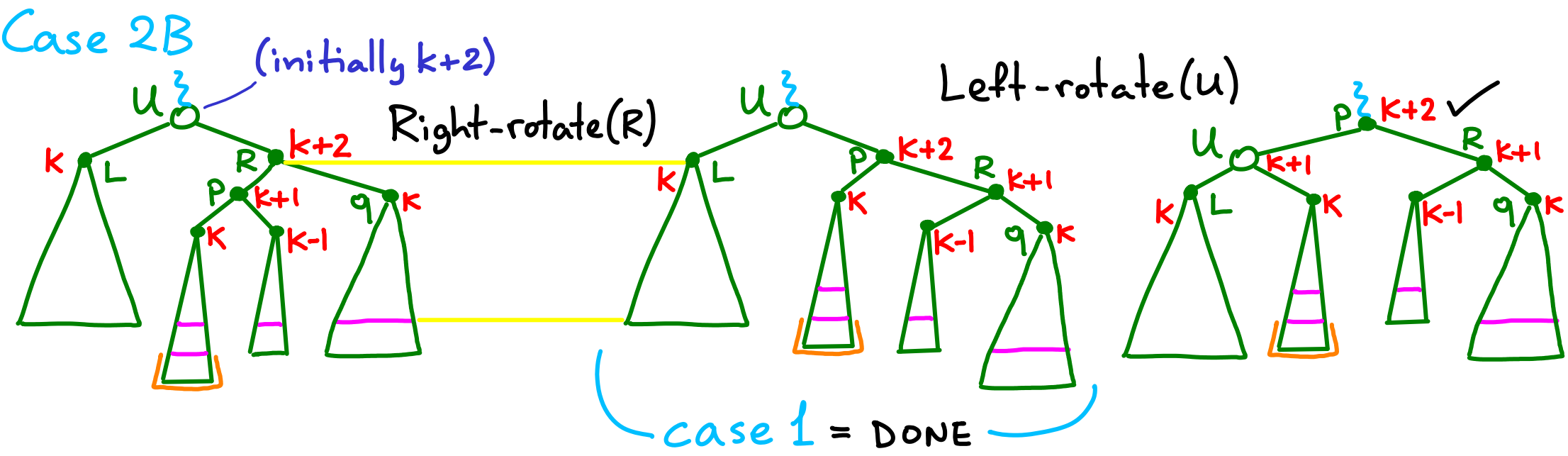
Right-rotate(R)




Case 2
(initially k+2)
u
k
L
R
k+2
P
k+1
q
K
IMPOSSIBLE
Case 2B
k-1
Case 2A
Right-rotate(R)
Left-rotate(u)
(DONE)


Case 2B
(initially k+2)
u
k
L
R
k+2
P
k+1
q
k
K
K-1


Case 2B
(initially k+2)
u
k
L
R
k+2
P
k+1
q
K
K
k-1
Right-rotate(R)
u
K
L
P
k+2
K
R
k+1
k-1
q
K
case 1 = DONE


Case 2B
(initially k+2)
u
k
L
R
k+2
P
k+1
q
K
K
K-1
Right-rotate(R)
u
k
L
P
k+2
K
R
k+1
K-1
q
K
Left-rotate(u)
p
k+2
u
k+1
R
k+1
k
L
K
K-1
q
K
case 1 = DONE

AVL INSERTION RECAP

- regular BST insertion, fix upward

AVL INSERTION RECAP

- regular BST insertion, fix upward
- if any ancestor encountered has original conditions, DONE

AVL INSERTION RECAP

- regular BST insertion, fix upward
- if any ancestor encountered has original conditions, DONE
- else if only problem is a height increase, check further up

AVL INSERTION RECAP

- regular BST insertion, fix upward
- if any ancestor encountered has original conditions, DONE
- else if only problem is a height increase, check further up
- else imbalance >1 // extra problem
  - ↳ fix things locally with 1 or 2 rotations

AVL INSERTION RECAP

- regular BST insertion, fix upward
- if any ancestor encountered has original conditions, DONE
- else if only problem is a height increase, check further up
- else imbalance >1 // extra problem
  - ↳ fix things locally with 1 or 2 rotations
  - safe to stop // extra problem was good news

$O(\log n)$ time with only $O(1)$ rotations

AVL INSERTION RECAP

- regular BST insertion, fix upward
- if any ancestor encountered has original conditions, DONE
- else if only problem is a height increase, check further up
- else imbalance >1 // extra problem
  - ↳ fix things locally with 1 or 2 rotations
  - safe to stop // extra problem was good news

$O(\log n)$ time with only $O(1)$ rotations

same as red-black trees but max depth is a bit less

AVL INSERTION RECAP

- regular BST insertion, fix upward
- if any ancestor encountered has original conditions, DONE
- else if only problem is a height increase, check further up
- else imbalance >1 // extra problem
  ↳ fix things locally with 1 or 2 rotations
  safe to stop // extra problem was good news

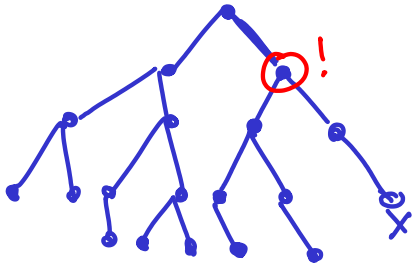
can figure all this out using O(1) info about height differences instead of value (save more space)

O(log n) time with only O(1) rotations

same as red-black trees but max depth is a bit less

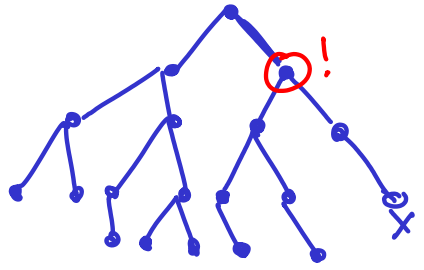
AVL DELETION

- Imbalance can be created at one node (at most) !



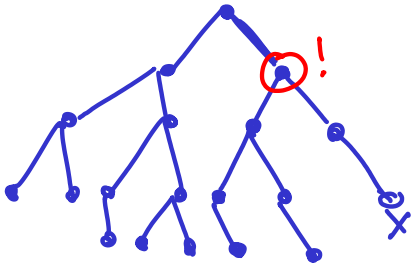
AVL DELETION

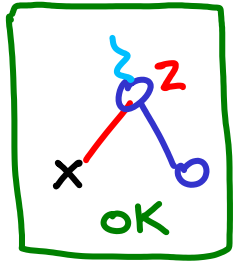
- Imbalance can be created at one node (at most)!
- Still not easier than insertion: { O(logn) time, O(logn) rotations } > RB



AVL DELETION

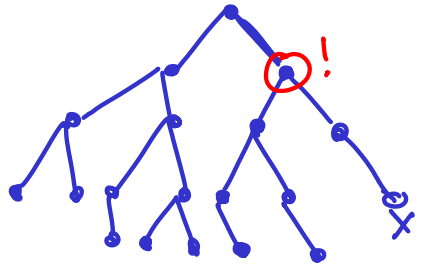
- Imbalance can be created at one node (at most) !
- Still not easier than insertion: { O(logn) time, O(logn) rotations } > RB

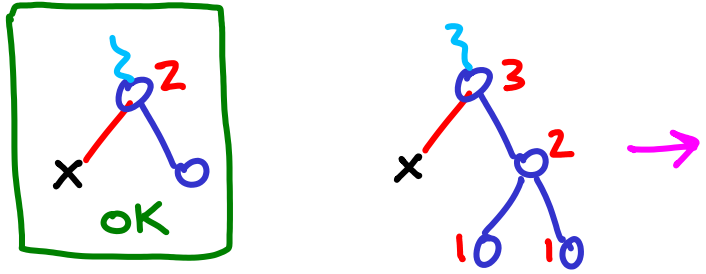




AVL DELETION

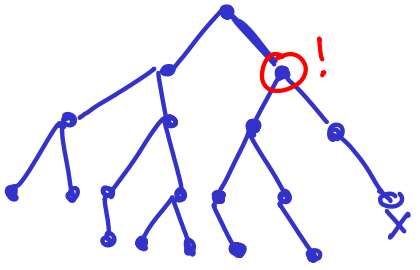
- Imbalance can be created at one node (at most) !
- Still not easier than insertion: { O(logn) time, O(logn) rotations } > RB

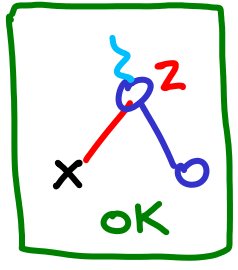


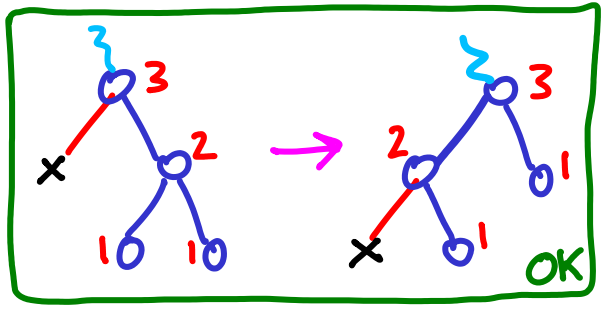


AVL DELETION

- Imbalance can be created at one node (at most) !
- Still not easier than insertion: { O(logn) time, O(logn) rotations } > RB

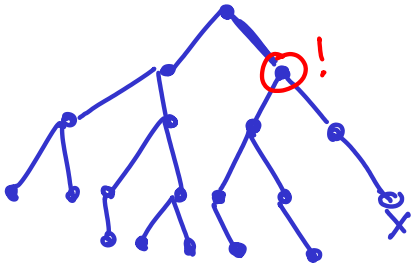


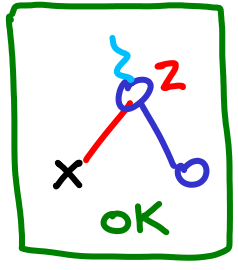


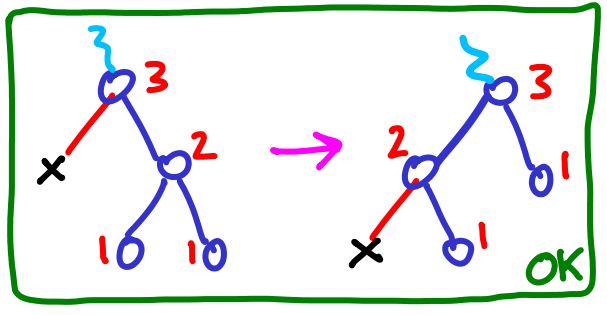


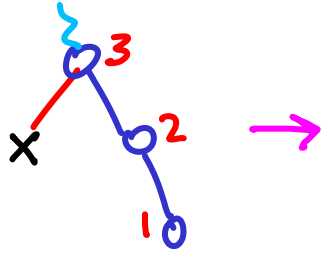
AVL DELETION

- Imbalance can be created at one node (at most) !
- Still not easier than insertion: { O(logn) time, O(logn) rotations } > RB



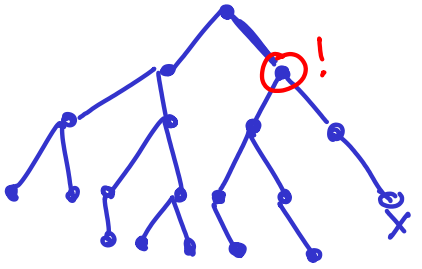


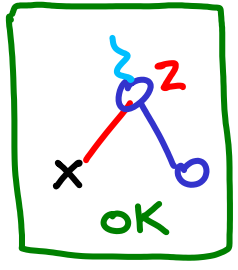


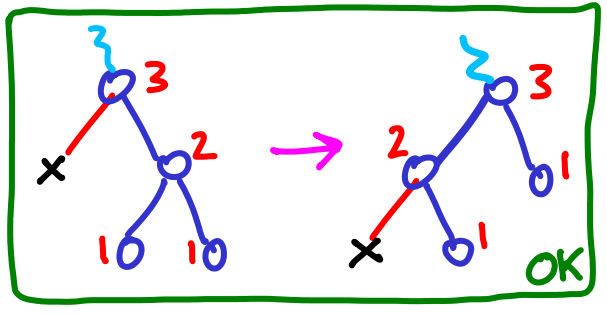


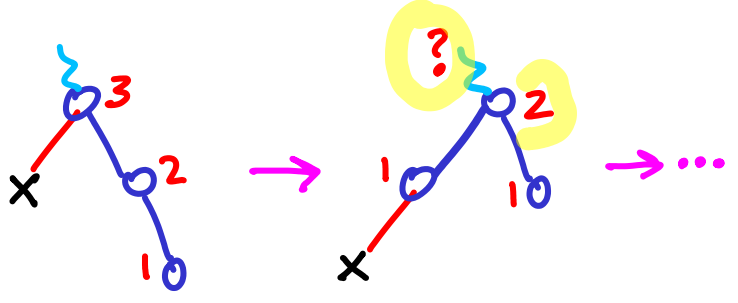
AVL DELETION

- Imbalance can be created at one node (at most) !
- Still not easier than insertion: { O(logn) time, O(logn) rotations } > RB



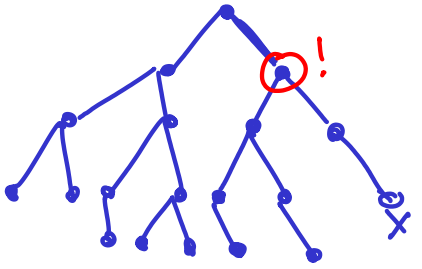


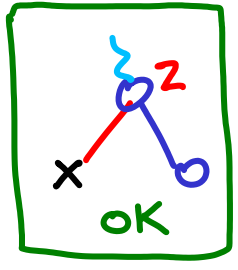


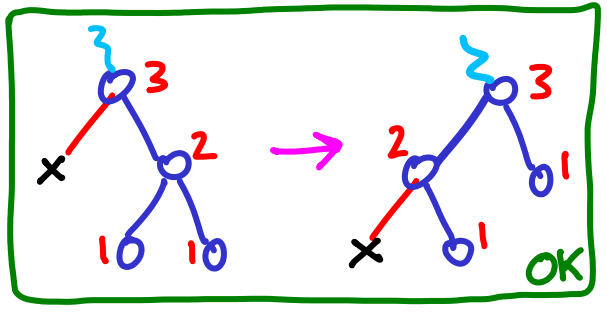


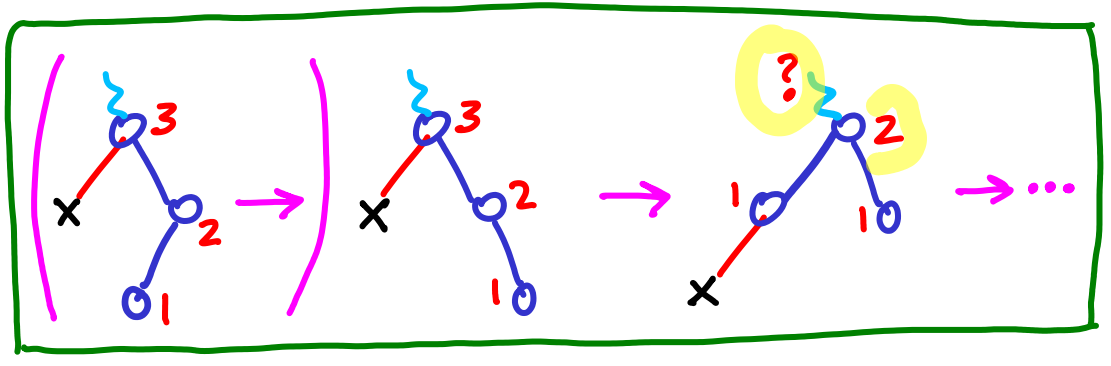
AVL DELETION

- Imbalance can be created at one node (at most) !
- Still not easier than insertion: { O(logn) time, O(logn) rotations } > RB



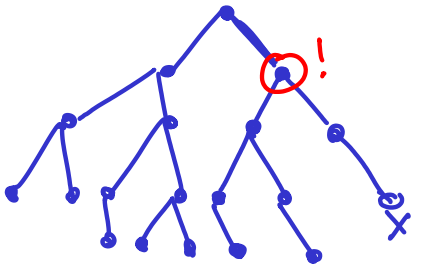


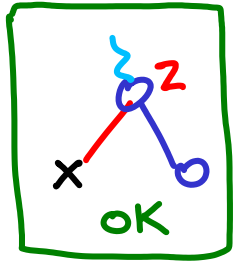


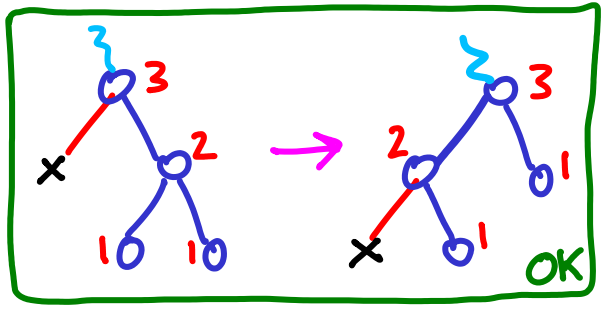


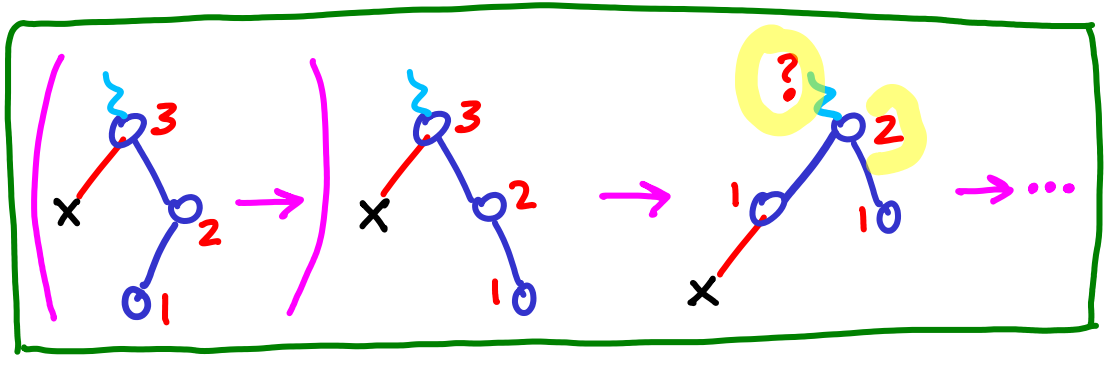
AVL DELETION

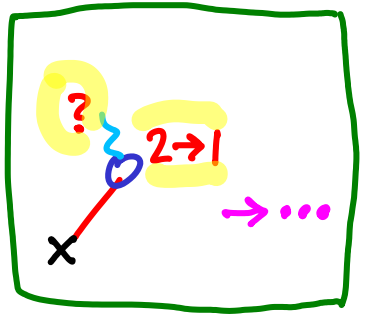
- Imbalance can be created at one node (at most) !
- Still not easier than insertion: { O(logn) time, O(logn) rotations } > RB





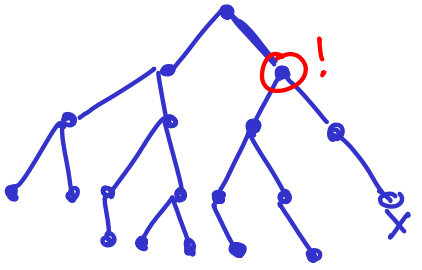


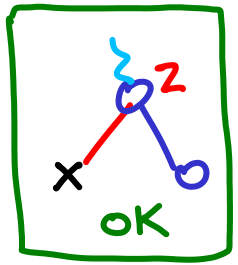


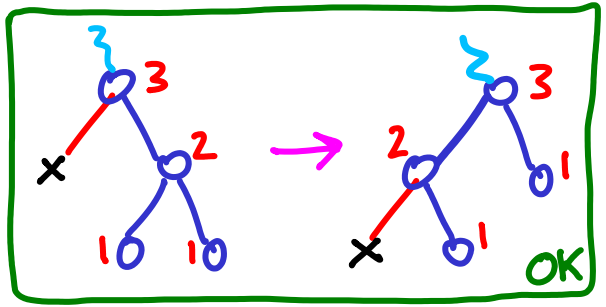


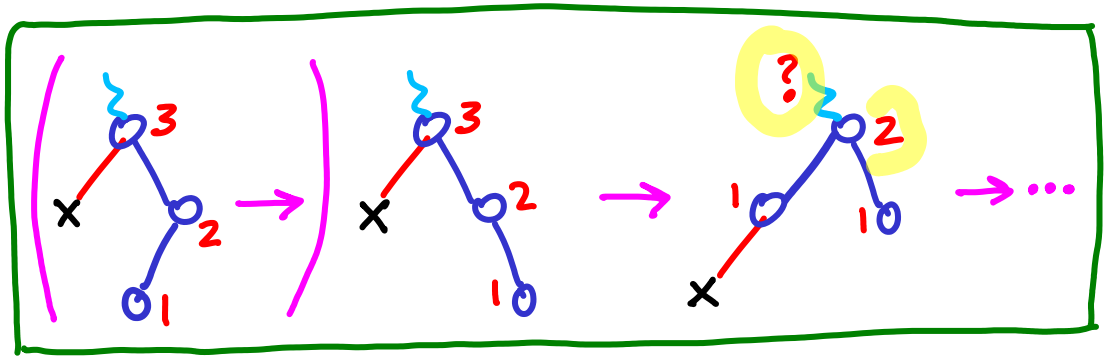
AVL DELETION

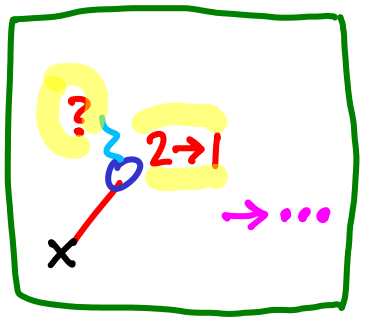
- Imbalance can be created at one node (at most) !
- Still not easier than insertion: { O(log n) time, O(log n) rotations } > RB



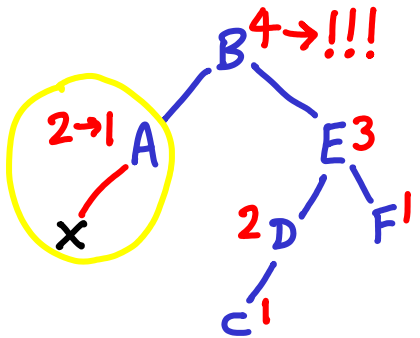






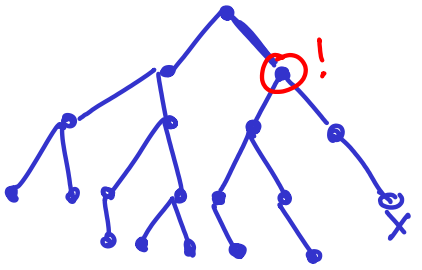


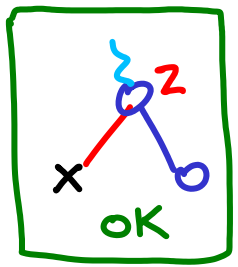
e.g.:

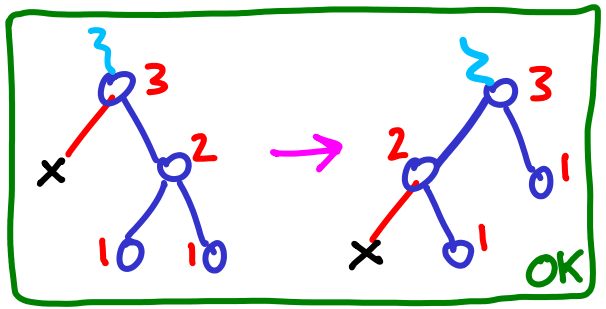


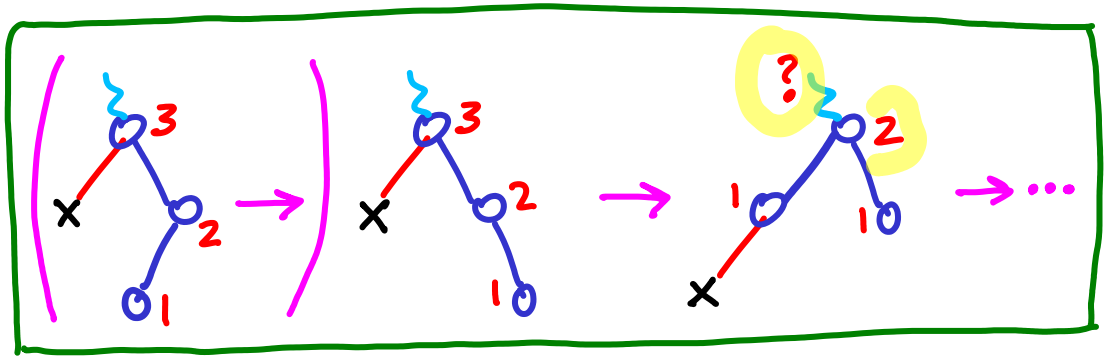
AVL DELETION

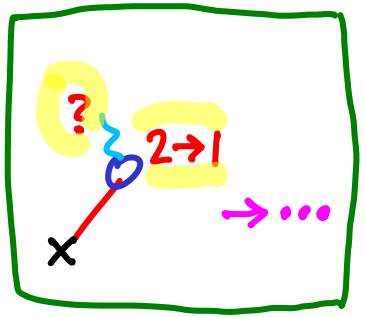
- Imbalance can be created at one node (at most) !
- Still not easier than insertion: { O(log n) time, O(log n) rotations } > RB



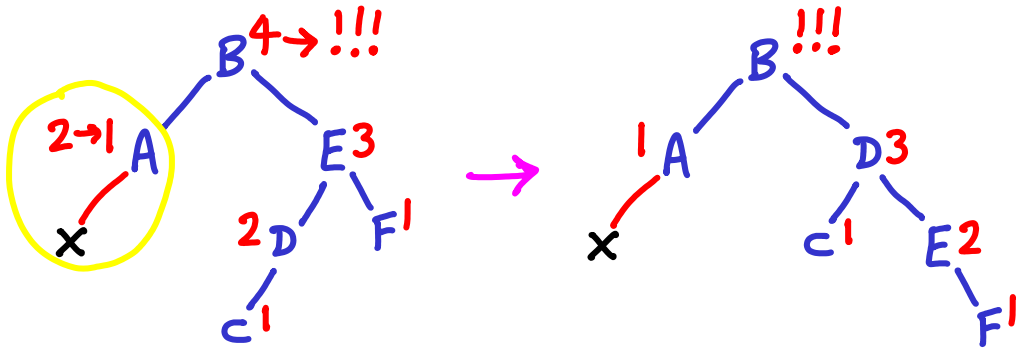






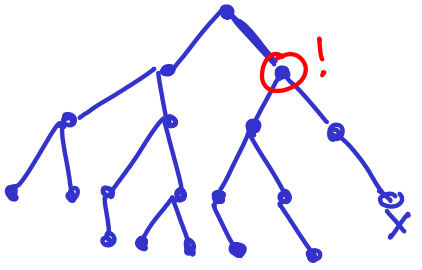


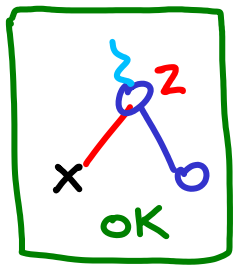
e.g.:

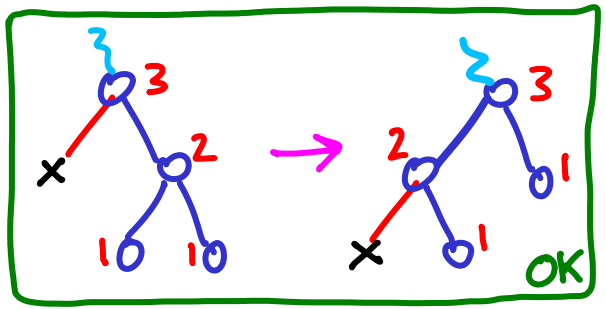


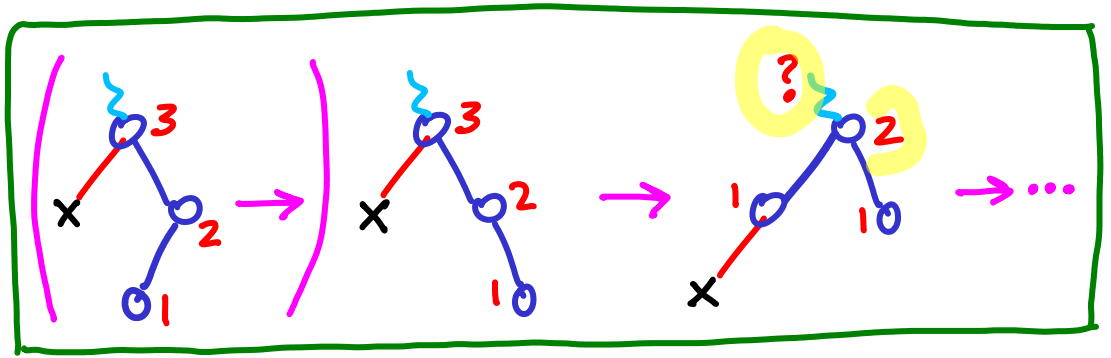
AVL DELETION

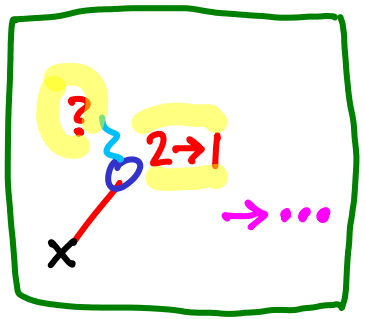
- Imbalance can be created at one node (at most) !
- Still not easier than insertion: { O(logn) time, O(logn) rotations } > RB



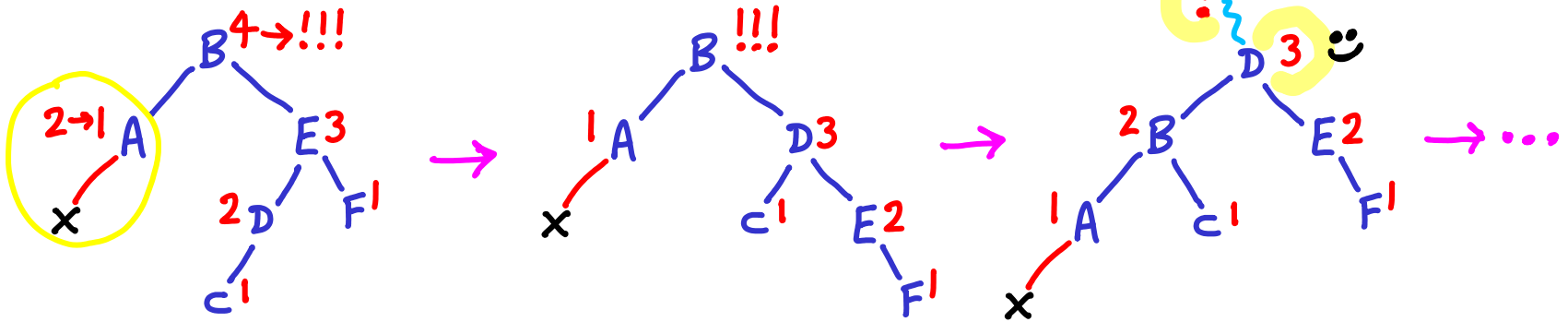








e.g.:



worst case		AVL	RB
	height	1.4 log n	2 log n
insert	# rotations	O(1)	O(1)
delete	# rotations	O(log n)	O(1)