INSERTION - AVL TREES

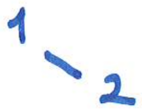
It is enough to perform <u>rotation</u> only at the first node violating the condition.

Why? — every rotation decreases height by one thus after fixing the node violating the AVL tree property, the upper level nodes have been fixed as well

Insert items from 1 to 7 to an empty tree

<u>insert 1</u> 1 (no violation, no ratation)

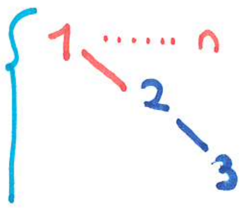
<u>insert 2</u>



(no violation, no rotation)

<u>insert 3</u>

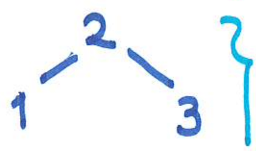
h= 3



red node is the one that violates the AVL tree property
red edge is the edge on which single rotation will be performed

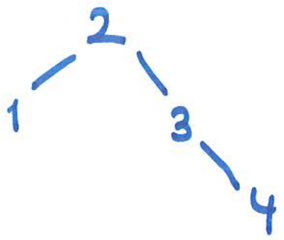
this is case (4) – inserting a new node into the right subtree of the right child of n

⇒ SINGLE LEFT



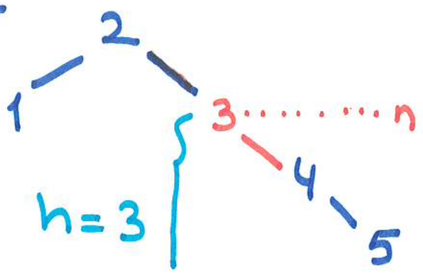
h=2

insert 4



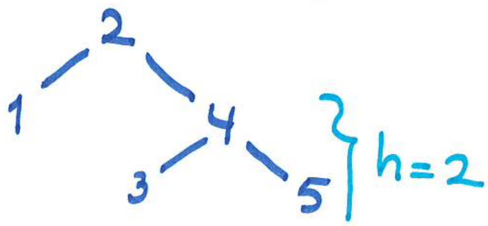
(no violation, no rotation)

insert 5



this is also case (4) — inserting a new node into the right subtree of the right child of n

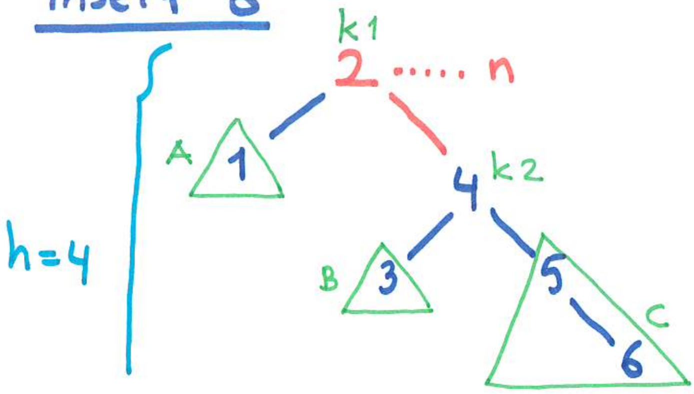
⇒ SINGLE LEFT



* HERE NOTE THAT THE AVL TREE PROPERTY IS ALSO VIOLATED FOR THE NODE 2 AT THE BEGINNING. BUT AFTER FIXING IT FOR NODE 3 IT DECREASES THE HEIGHT BY ONE AND THIS CHANGE RESULTS IN FIXING THE PROBLEM ALSO FOR NODE 2

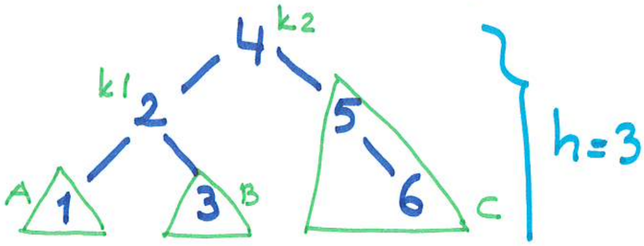
⇒ THUS, ONLY ONE ROTATION IS ENOUGH FOR INSERTION

insert 6



this is also case (4) – inserting a new node into the right subtree of the right child of n

⇒ SINGLE LEFT



here we know that if the height at node k1 is h → the height at node k2 should be h-1 (since k2 is the child of k1)

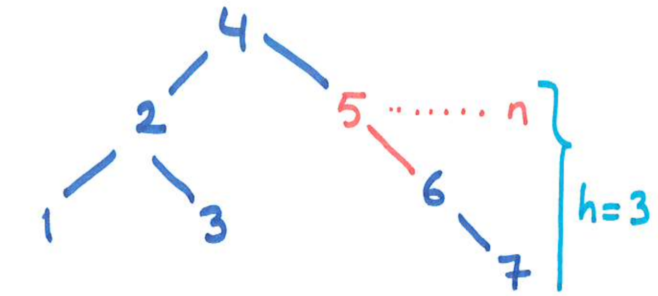
→ the height of the subtree C should be h-2 (since C is the subtree of k2)

→ the height of the subtree B should be h-3 (since if it was longer, this would be a problem before and had to be fixed before)

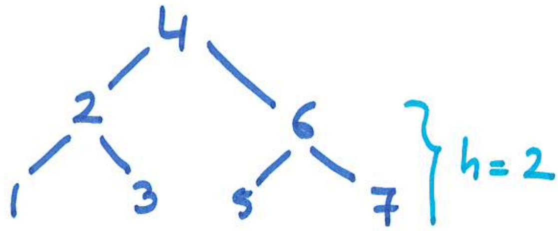
→ the height of the subtree A should also be h-3

(Since the difference between the left and right subtrees of k1 should be 2, otherwise it wouldn't violate the AVL tree property)

insert 7

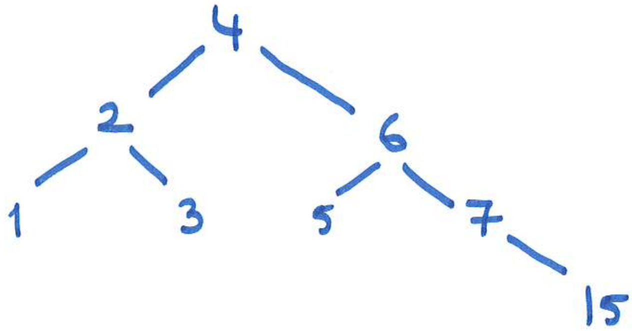


this is also case (4) —
Inserting a new node into the right subtree of the right child of n
⇒ SINGLE LEFT



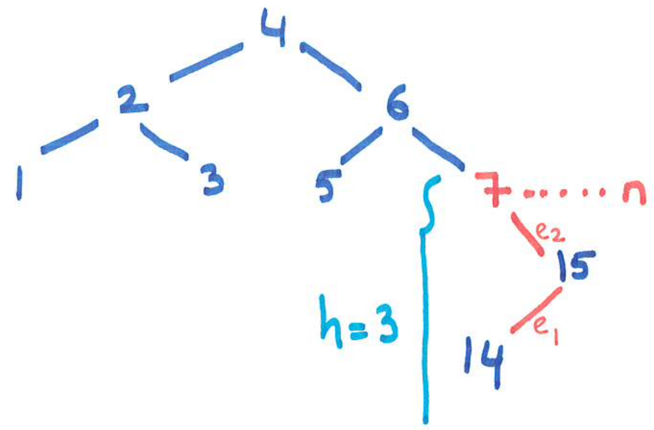
Now insert items from 15 to 7 to this tree

insert 15



(no violation, no rotation)

insert 14



this is case (3) — inserting a new node into the left subtree of the right child of node n

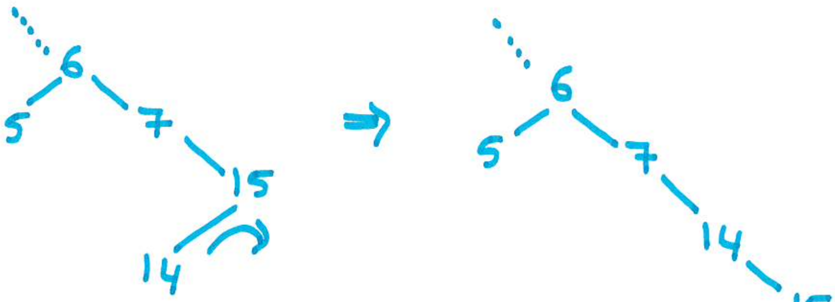
n is the node violating the AVL tree property

red edges are the ones on which double rotation will be performed

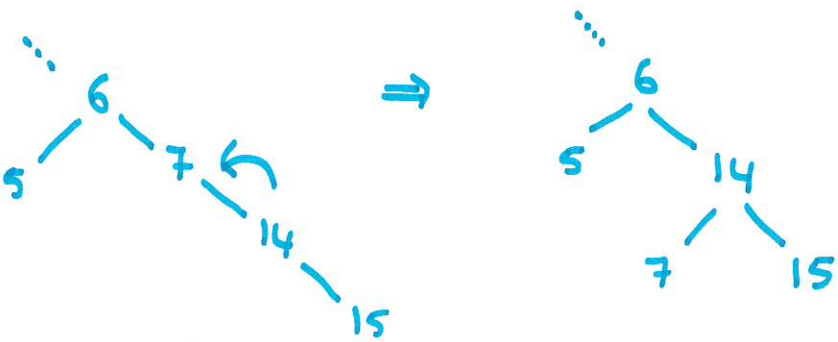
⇒ DOUBLE RIGHT-LEFT

Indeed, here we perform two rotations. First the right rotation on e_1 and then the left rotation on e_2

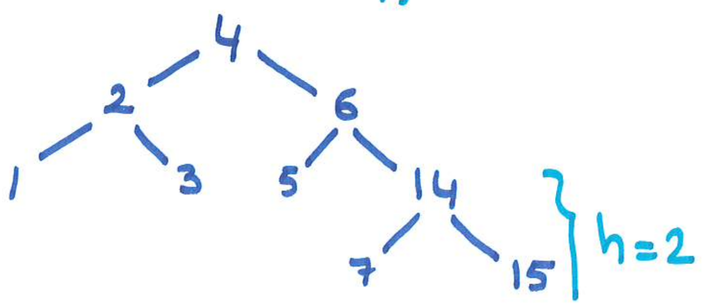
→ first right rotation on e1



→ then left rotation on e_2



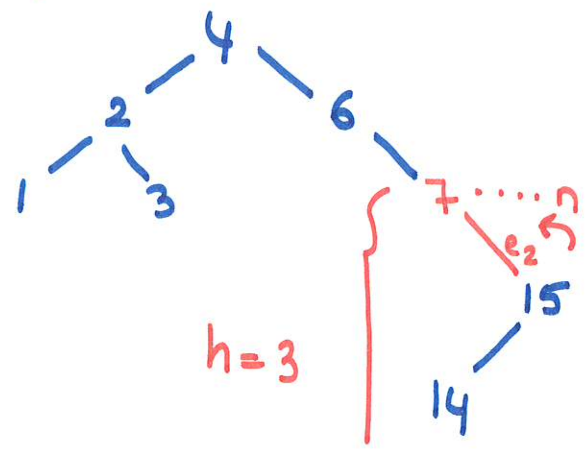
⇒ resulting tree



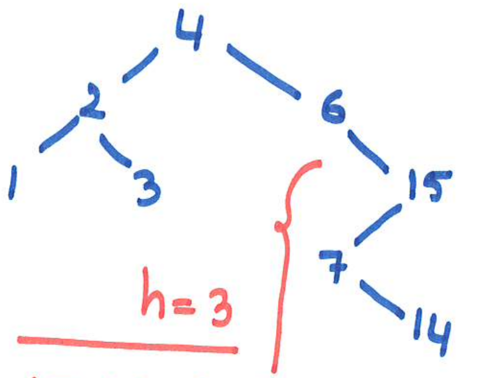
NOTE THAT IN THIS EXAMPLE SINGLE ROTATION ON e_2 WOULD NOT HELP

JUST TRY

before



after the single left rotation

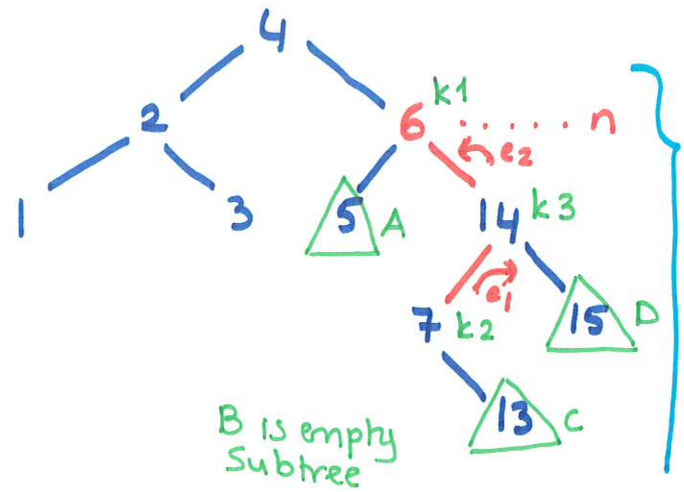


IT IS STILL THE SAME (NO DECREASE IN THE HEIGHT)

DOES NOT SOLVE THE PROBLEM

*** THUS, DOUBLE ROTATION IS NECESSARY

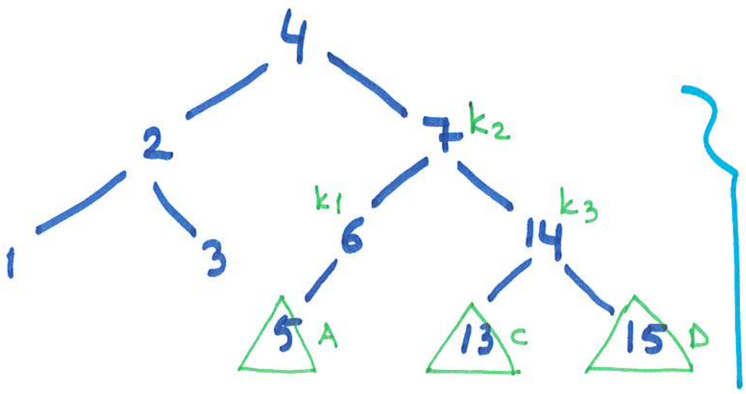
insert 13



h=4

this is also case (3) inserting a new node into the left subtree of the right child of node n

⇒ DOUBLE RIGHT-LEFT

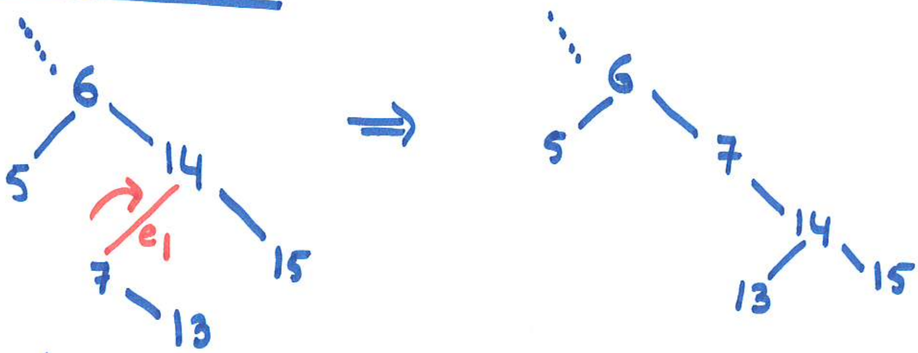


h=3

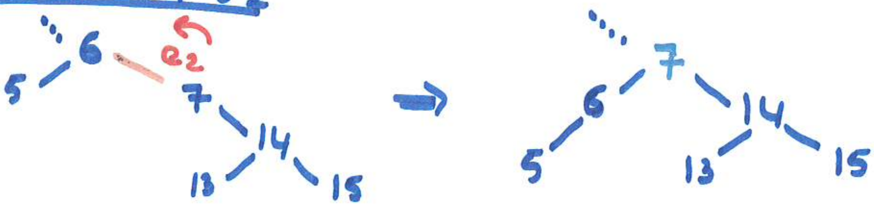
as expected height decreases by one

Again note that this double rotation indeed corresponds to two single rotation. First, right rotate on e_1 then left rotate on e_2

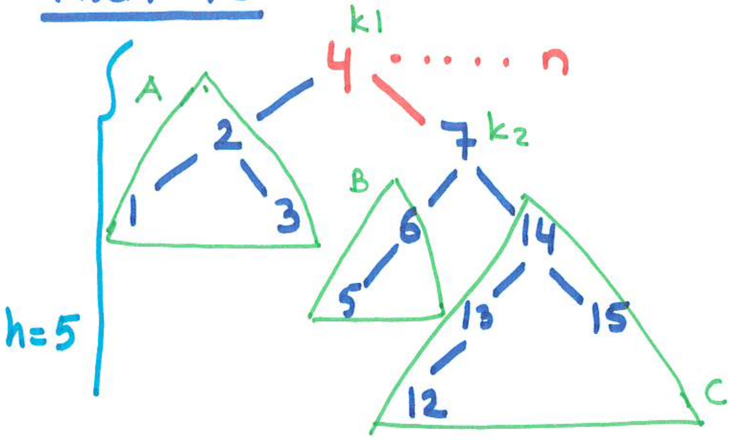
right rotate on e_1



left rotate on e_2

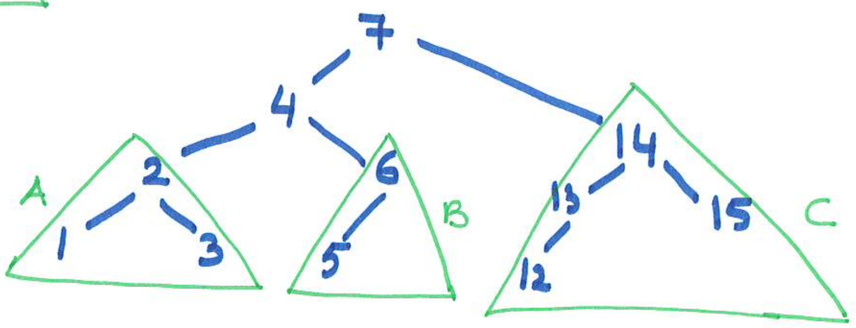


insert 12



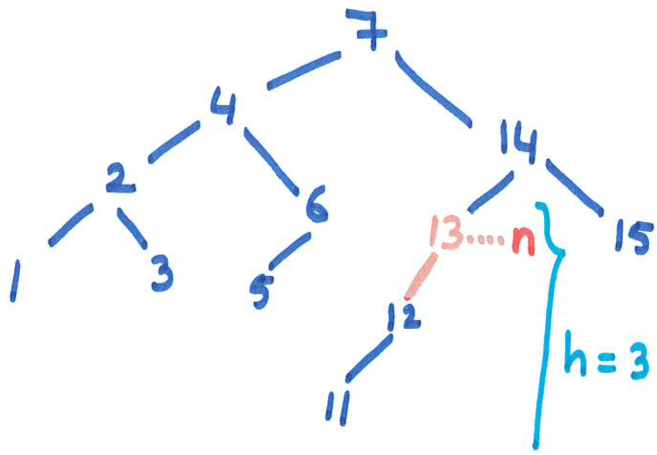
this is case (4) - inserting a new node into the right subtree of the right child of node n

⇒ SINGLE LEFT



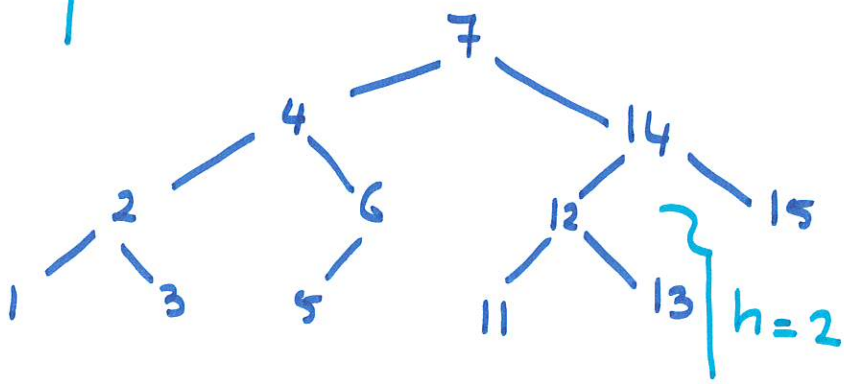
now height $h = 4$

insert 11

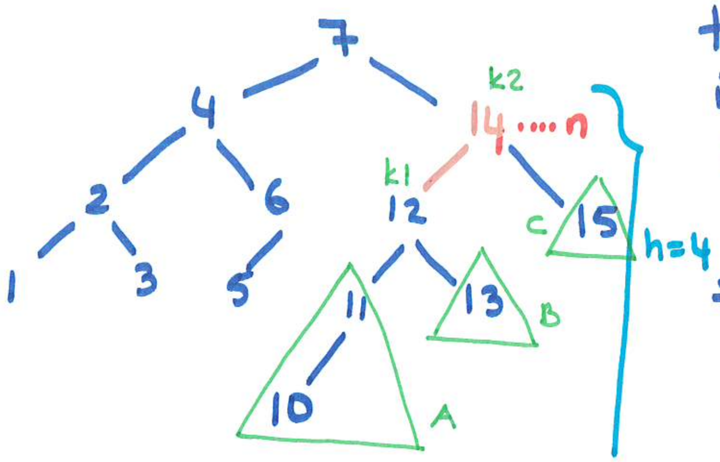


this is case (1) — inserting a new node into the left subtree of the left child of node n

⇒ SINGLE RIGHT

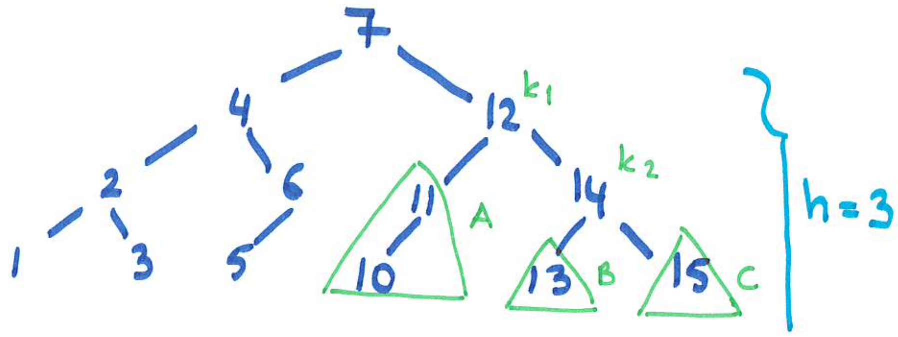


insert 10

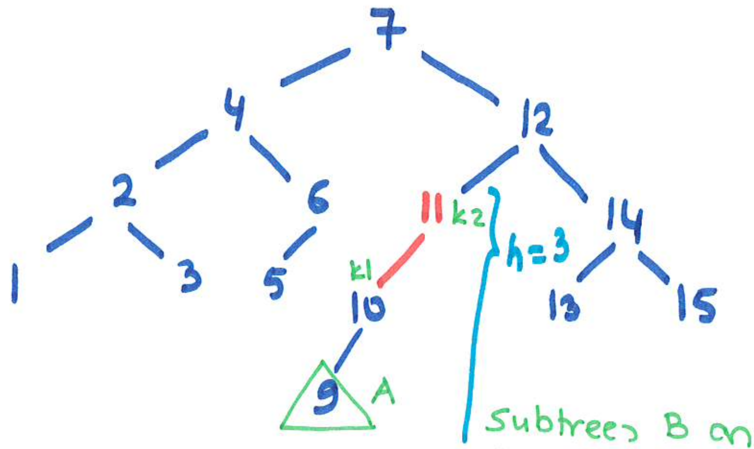


this is also case (1) – inserting a new node into the left subtree of the left child of n

⇒ SINGLE RIGHT



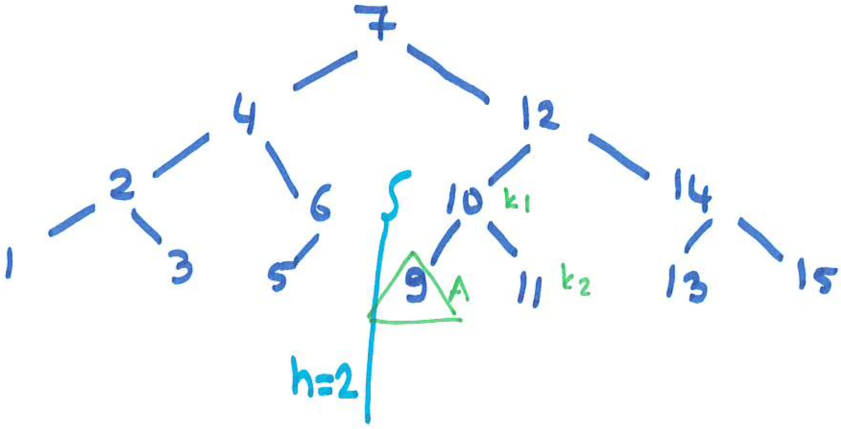
insert 9



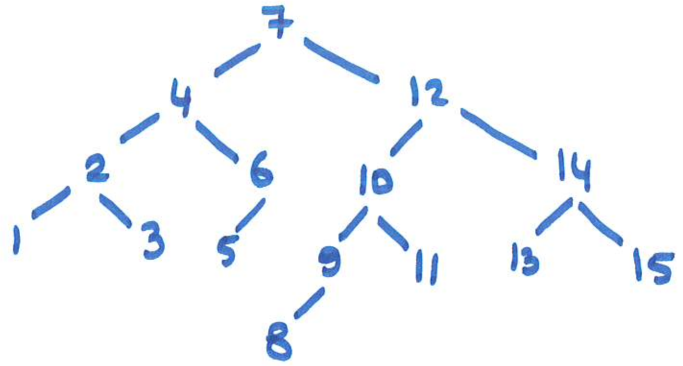
subtrees B and C are empty

this is also case (1)

⇒ SINGLE RIGHT

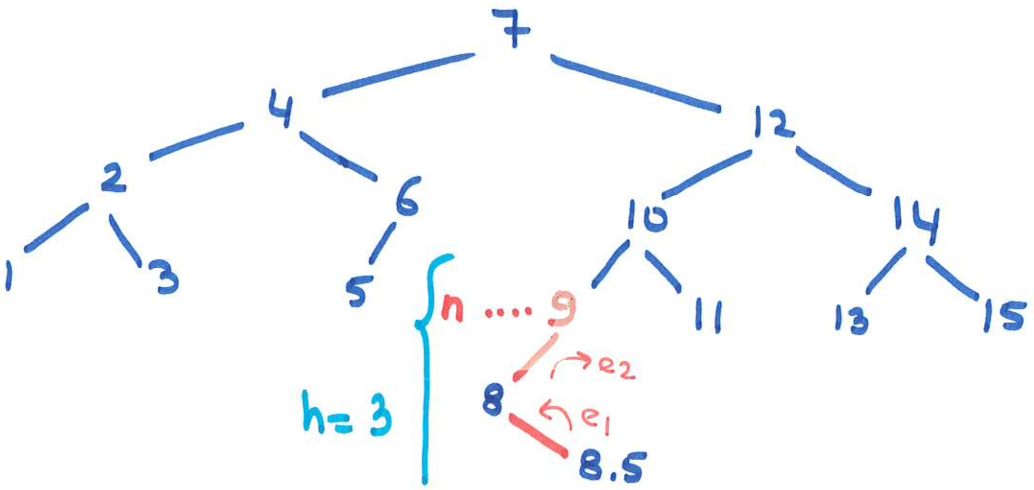


insert 8



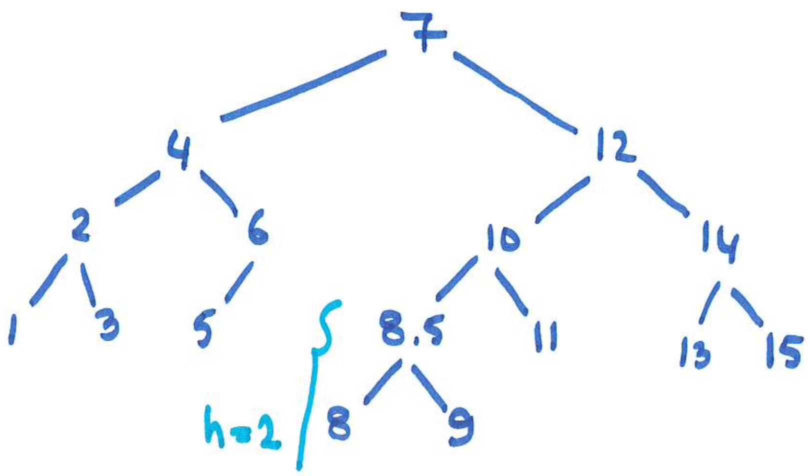
(no violation, no rotation)

Now insert 8.5



this is case (2) — inserting a new node into the right subtree of left child of node n

⇒ DOUBLE LEFT-RIGHT

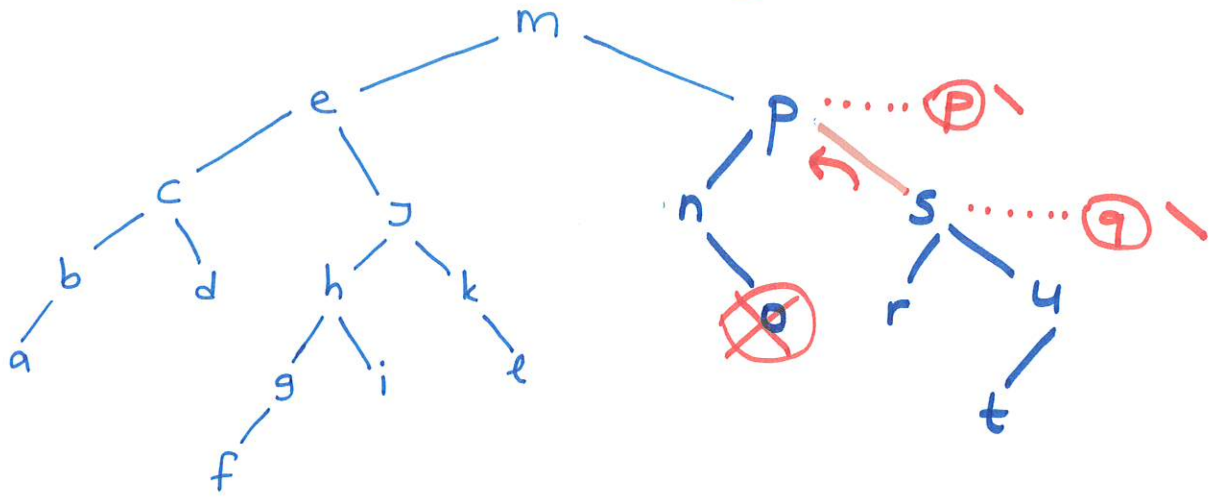


Likewise, there are two single rotations here. First left rotate on e_1 then right rotate on e_2

corresponding double left-right rotation

DELETION – AVL TREES

steps for the example given on the slides



step 1 : check node n → its taller subtree is shortened

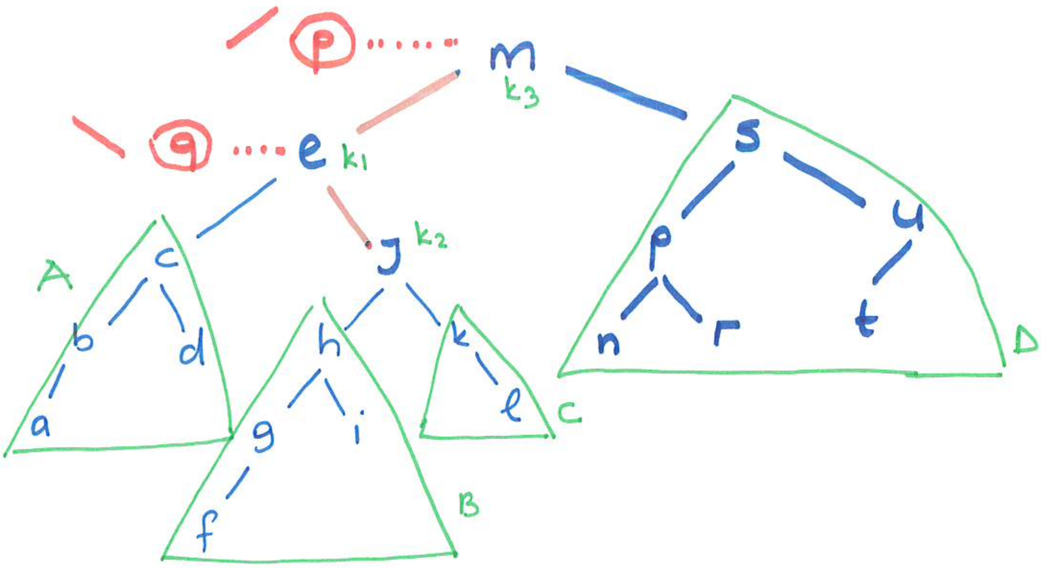
THIS IS CASE 2
no rotations
change the balance factor of n
(from \ to —)
shorter remains true
which means you have to continue with the parent of n

step 2 : check node p → its shorter subtree is shortened

CASE 3

check the balance factor of the root of its taller subtree (which is node s)
[this node is indicated with letter q on the slide]

⇒ CASE 3B , single rotation , shorter remains true

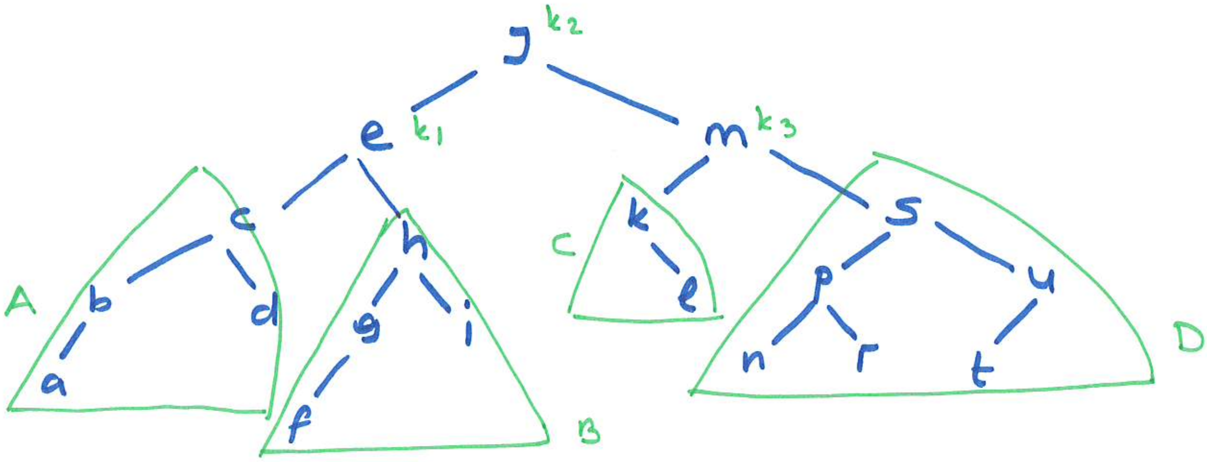


Step 3 check node m → its shorter subtree is shortened

CASE 3

check the balance factor of the root of its taller subtree (which is node e)

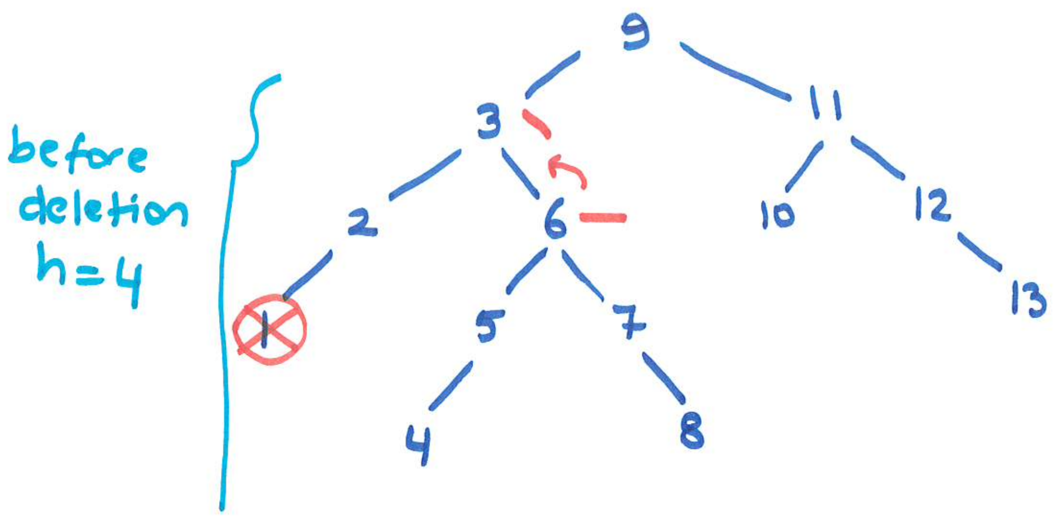
⇒ CASE 3C, double rotation, shorter remains true



DOUBLE LEFT-RIGHT ROTATION

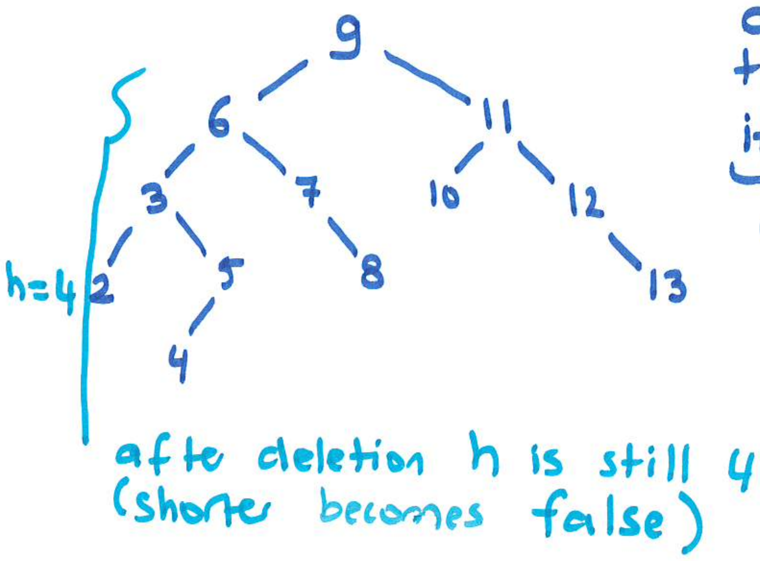
ANOTHER EXAMPLE

delete 1



step 1 : check node 2 → its taller subtree is shortened
CASE 2 , no rotations , but shorter remains true

step 2 : check node 3 → its shorter subtree is shortened
CASE 3
⇓
check the balance factor of the root of its taller subtree
it is equal (–)
CASE 3A , single rotation , shorter becomes false



Step 3 : STOP , do not check node 9 and the other upper nodes since shorter becomes false

EXERCISES

① delete 9, 10, 5, and 13 from the tree given in the previous example

② how to implement double right-left rotation in C++ ?

```
class TreeNode{
        int item;
        TreeNode *left;
        TreeNode *right;
};

void rotateRightLeft(TreeNode *&k1){
        :
```