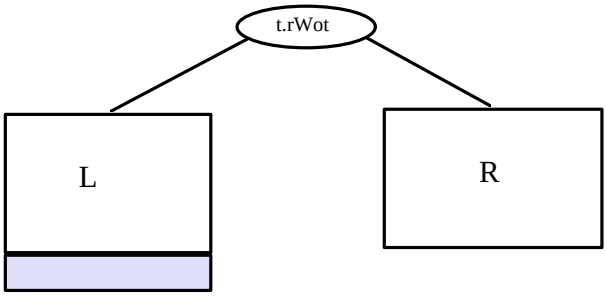
Insertion Qnto an AVL tree

An AVL tree Qs a Height Balanced Search Tree. We will use recursion to Qmplement the Qnsertion algorithm. In QnsertiVg an Qtem Qn an AVL we may distort the balance, Q.e. the heights of the left an6 right trees may differ by more than one. If tree Qs balanced after Qnsertion then return this tree.

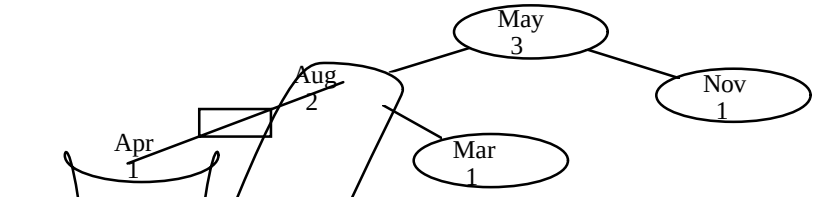
Assume Qnsertion has been Qn the left (sub)tree of t and assume the balance has beiVg dimaorted.



HL = HR + 2, HL -- height of Left, HR -- height of R

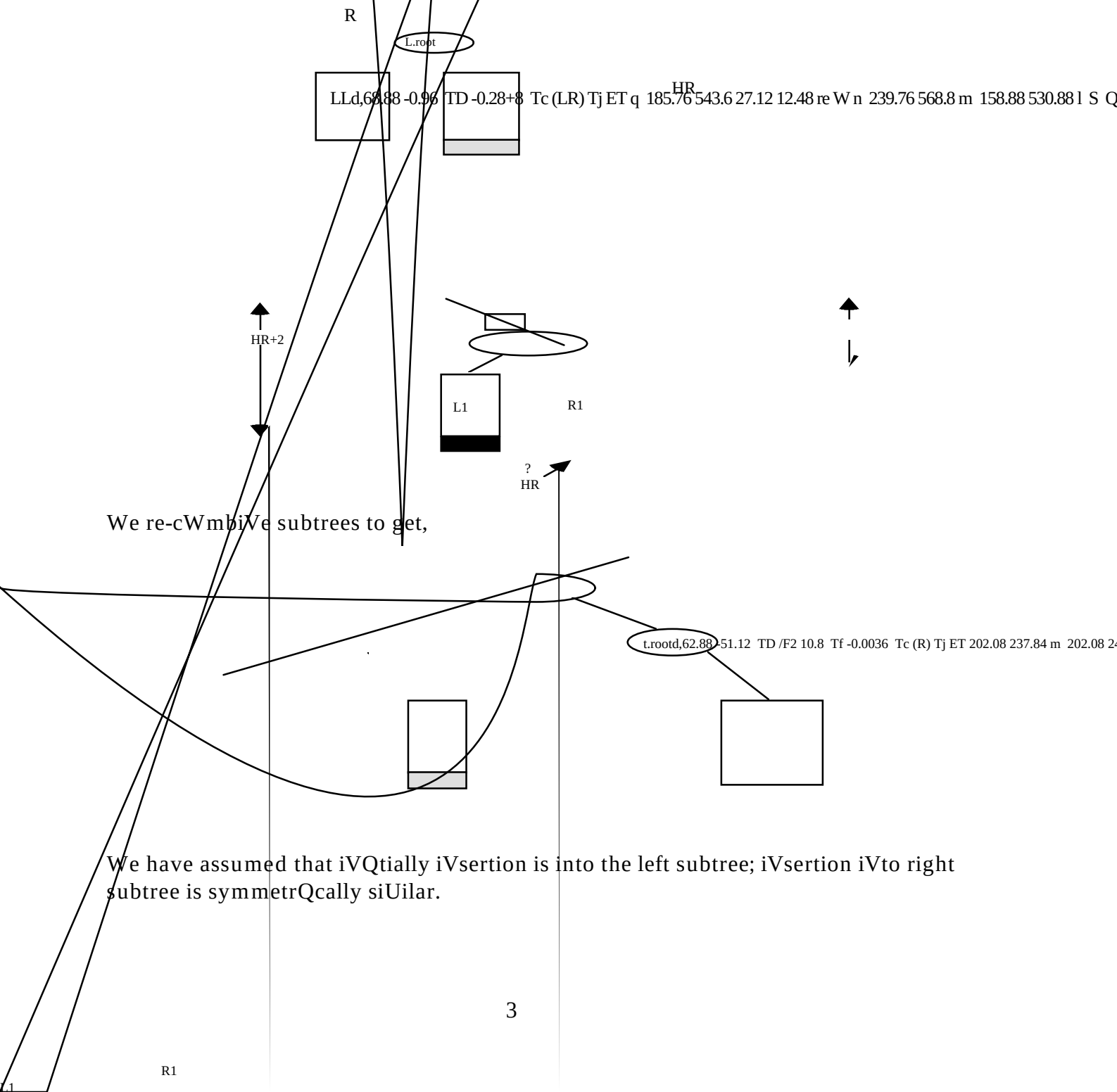
Case 1. Hgt(LL) > Hgt(LR) and Hgt(LL) > Hgt(R)

This is Vot balanced, to restore the balance we Rotate Right about the Vode ‘Mar’, the lowest Vode that is unbalanced.



This is Vow an AVL tree, wQth height 3 (the same as before iVsertion)

Case 2. Hgt(LR) > Hgt(LL) and Hgt.27R) > Hgt.(R)) Tj ET 325.2 593.04 m 325.2 596.

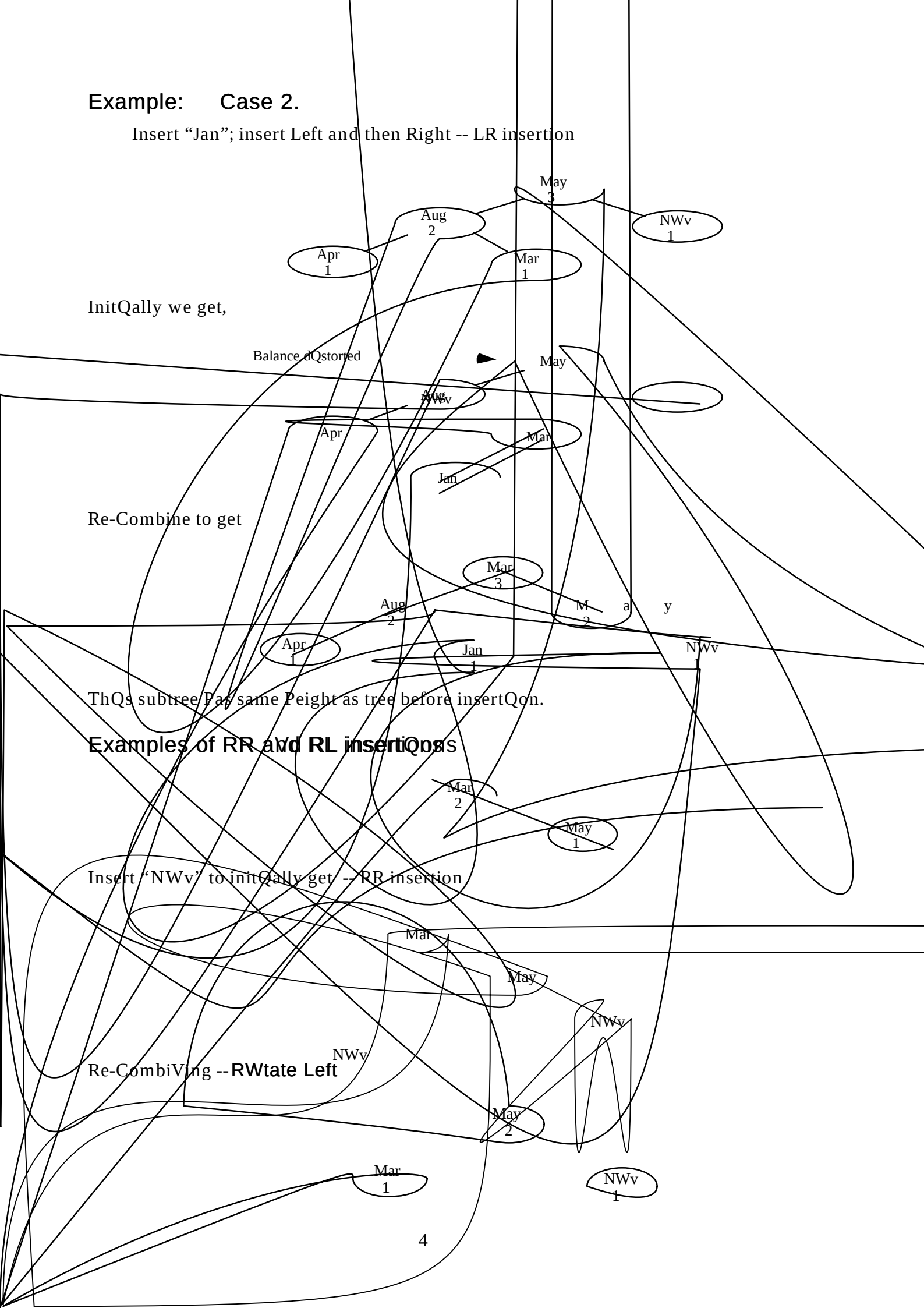


We re-cWmbiVe subtrees to get,

We have assumed that iVQtially iVsertion is into the left subtree; iVsertion iVto right subtree is symmetrQcally siUilar.

Example: Case 2.

Insert “Jan”; insert Left and then Right -- LR insertion



InitQally we get,

Re-Combine to get

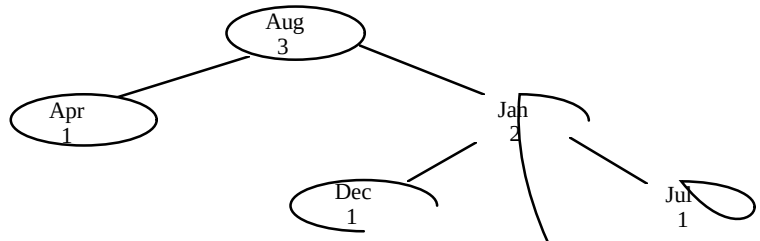
ThQs subtree Pas same Peight as tree before insertQon.

Examples of RR and RL insertQons

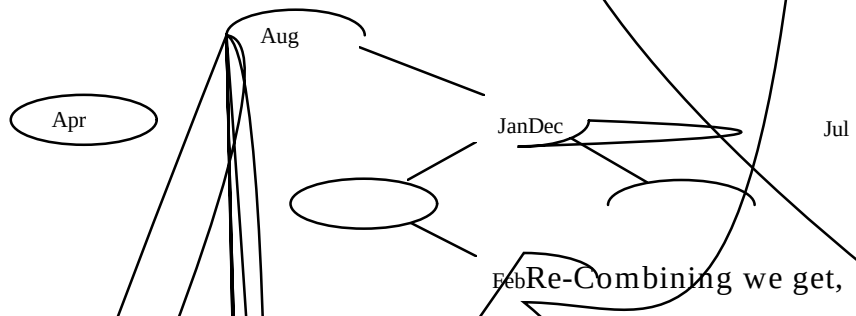
Insert “NWv” to initQally get -- RR insertion

Re-CombiVng -- RWtate Left

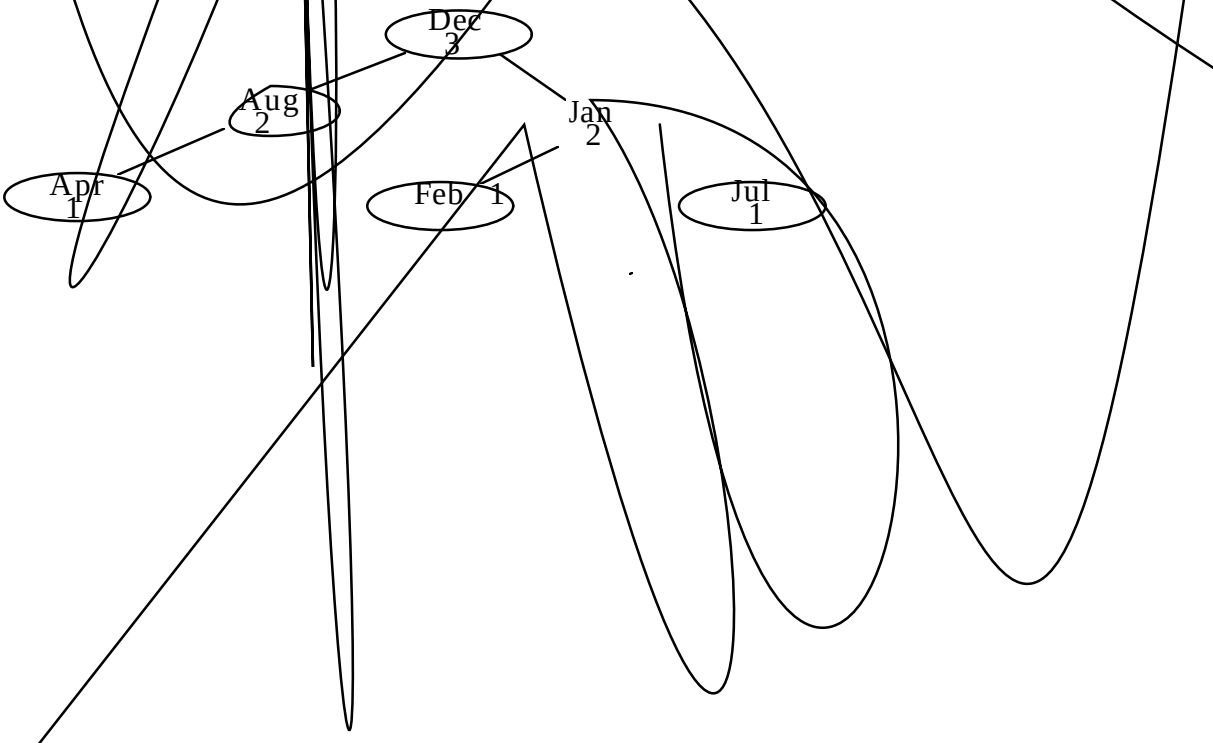
Example Insert into Right of Left subtree -- RL insertion



Insert “Feb” to initially get



Re-Combining we get,



Eiffel Program for Adding item to AVL tree

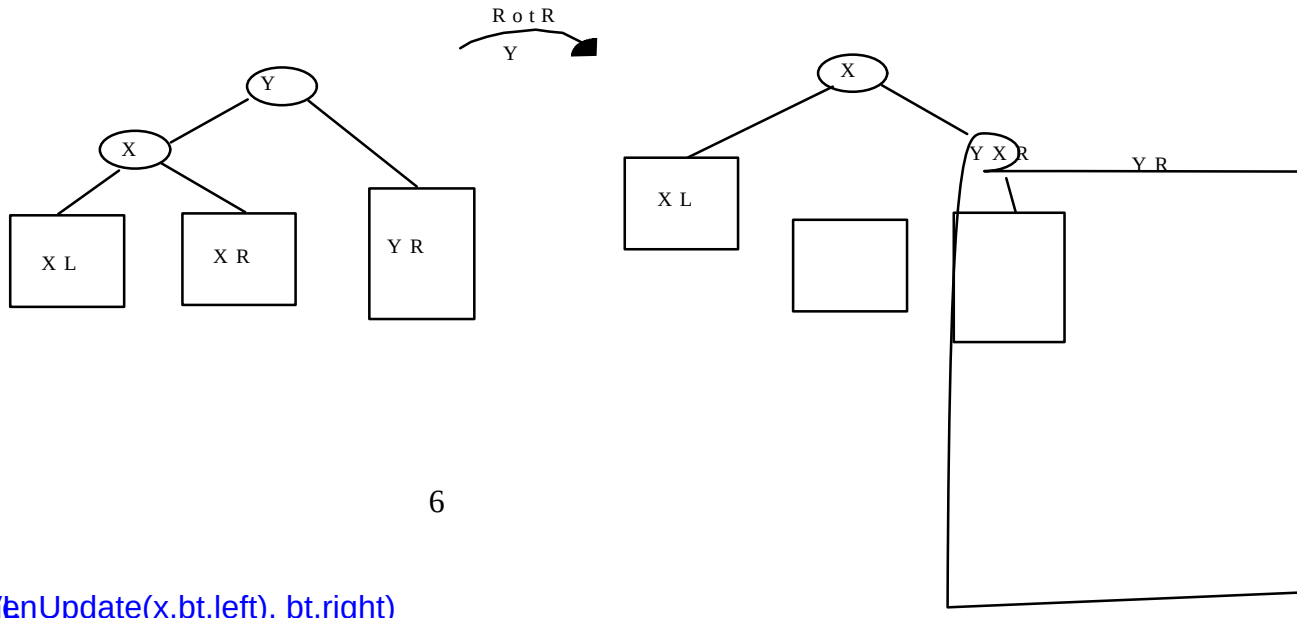
The critcal function AVL_Update is similar to Insert in Binary Search Trees.

```
AVL_Update(x:G; bt:BIN_NODE[G]) : BIN_NODE[G] is
    Tocal
        t : BIN_NODE[G]
    do
        !!t
        if bt = void then
            t.build(x,void,void)
            result := t

            result := Rebal(t)
        elseif
            t.buiTd bt.vaTue, bt.left,  AVL_Update(x,bt.right))
            result := Rebal(t)
        elseif
            result := bt
        end
    end -- AVL_Update
```

After a subtree has been recursively built itffs rebalanced. To implement Rebal we define a basic functions RotR "Rotate Right" and syUmetrically RotL. For example, RotR will be used for an LL insertion and RotL for a RR insertion.

RotR about node y

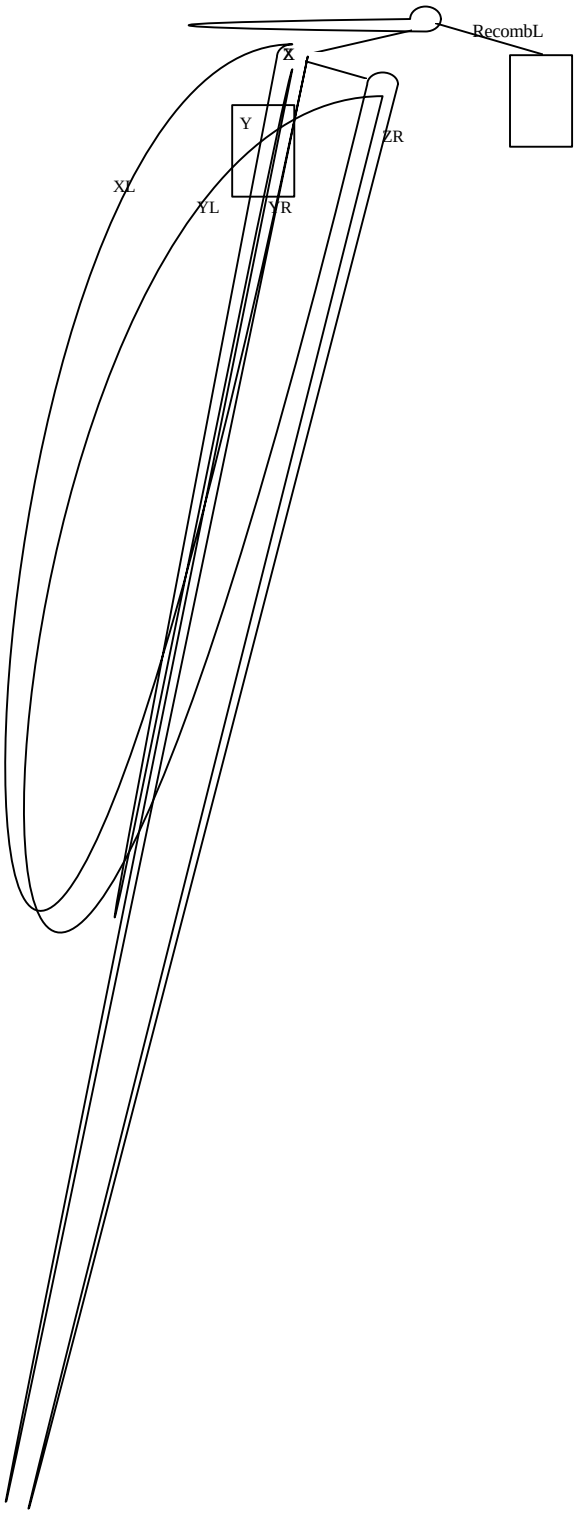


In Eiffel,

elseif(bt.vaTue,AVLenUpdate(x,bt.left), bt.right)

```
RotR(t:BIN_NODE[G]):BIN_NODE[G] is
        require
                Non_Void_Left:          t/=void and then t.left/=void
        local
                L,R, new, new_right : BIN_NODE[G]
        do
                L := t.left
                R   :=   t.right
                !!new_right
                new_right.build(t.value, L.right, R)
                !!new
                new.build(L.value, L.left, new_right)
result := new
```

RecombL “Recombine Left”



```
Rebal(t:BIN_NODE[G]) : BIN_NODE[G] is
require
        Non_Void:  t /= void
do
    If Bal_Factor(t) > 1 then
        result := Balance_Left(t)
    elseif Bal_F  /tor(t) < -1 then

    else
        result := t
    end
eVd -- Rebal
```

The function Bal_F ctor returns

Height(Left) - Height(Right)

```
Bal_Factor(t : BIN_NODE[G]) : INTEGER is
require
        Non_Void: t /= void
do
    result:= Height(t.left) - Height(t.right)
end - - B a l _ F a c t o r
```

Th12 rWutines Balance_Left and Balance_Right do the apprWpriate 3 12 anciVg.

```
Balance_Left(t:BIN_NODE[G]):BIN_NODE[G]
require
        N o n _ V o i d _ L e f t :
          t /= void and then t.left /= void
do
    ifthen Bal_F ctor(t.left) = 1
        result := RotR(t)
    elseif Bal_Factor(t.left) = -1 then
        result := reCombL(t)
    else
        result := t

eVd -- Balance_Left
```



```
end result := Balance_Right(t)
```