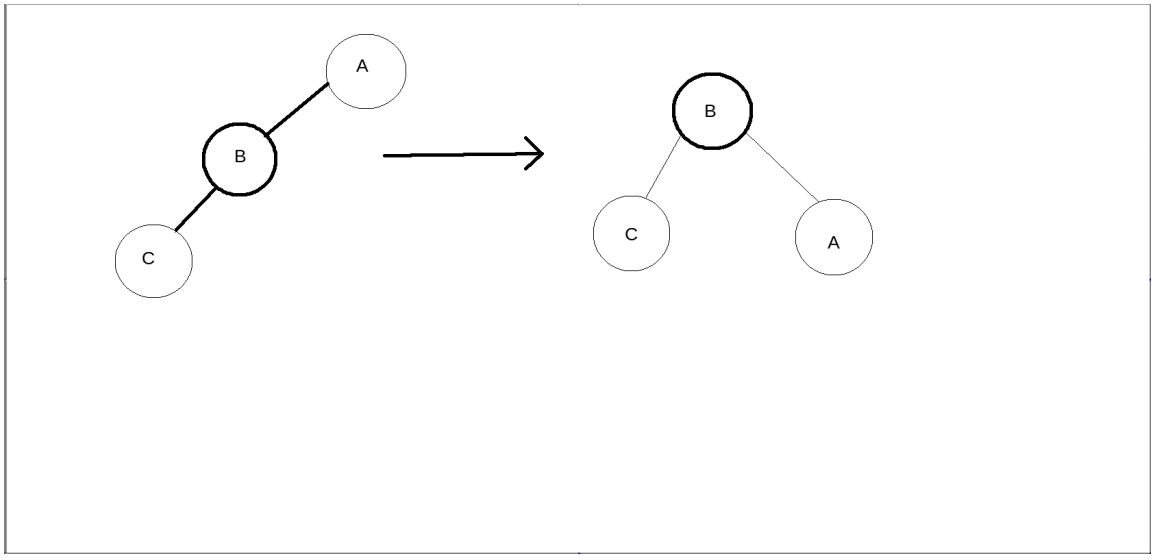
Zadaća 5

Zadatak 1

In [ ]:
```
def rightRotate(T, x):

    #ne mozes rotirati desno ako x nema lijevo dijete
    if(x.left == None):
        return

    y = x.left #za y uzimamo lijevo dijete od x
    x.left = y.right #za novo lijevo dijete od x uzimamo desno dijete od y

    if(y.right != None):
        y.right.parent = x #uspostavi i drugu stranu veze ako desno dijete o

    if(x.parent == None):
        T.root = y #slucaj kad je x bas root, y postaje root

    elif(x == x.parent.left):
        x.parent.left = y #ako je x lijevo dijete svog roditelja, njegovo no

    else:
        x.parent.right = y #ako je x desno dijete svog roditelja, njegovo no

    y.parent = x.parent #uspostavi i drugu stranu veze
    y.right = x #x postaje desno dijete od y
    x.parent = y #y postaje roditelj od x
```



Zadatak 2

Definicije:

- balanceFactor: node je balansiran ako je njegov balance factor -1, 0 ili 1. Balance factor definiramo kao: visina desnog djeteta vs visina lijevog djeteta. Visina lista je 0, a visina nullptr-a je -1
- lijevo težak node: balans mu narušava lijevo podstablo, dakle balans mu je -2
- desno težak node: balans mu narušava desno podstablo, dakle balans mu je 2

In [ ]:
```
class Node:
    def __init__(self, val) -> None:
        self.left = None
        self.right = None
        self.parent = None
        self.height = 0
        self.balanceFactor = 0
        self.value = val

def leftRotate(T, n):
    pass #analogno kao i right rotate

def update(node):

    #visina nullptra je -1
    lh = -1
    rh = -1

    if(node.left != None):
        lh = node.left.height
    if(node.right != None):
        rh = node.right.height

    node.height = 1 + max(lh, rh)
    node.balanceFactor = rh - lh

def avl_insert(T, node, val):
    if(node == None):
        return Node(val)


    if(val < node.value):
        node.left = avl_insert(T, node.left, val)
    else:
        node.right = avl_insert(T, node.right, val)

    #azuriraj balans faktor nodea
    update(node)

    return balance(node)

def balance(node):
    if(node.balanceFactor == -2):
        #stablo je lijevo-teško
```

```
        if node.left.balanceFactor <= 0:
            return leftLeftCase(node)
        else:
            return leftRightCase(node)

    elif(node.balanceFactor == 2):
        if(node.right.balanceFactor >= 0):
            return rightRightCase(node)
        else:
            return rightLeftCase(node)

    #inace je node balansiran
    return node

def leftLeftCase(node):
    return rightRotate(node)

def leftRightCase(node):
    node.left = leftRotate(node.left)
    return leftLeftCase(node)

def rightRightCase(node):
    return leftRotate(node)

def rightLeftCase(node):
    node.right = rightRotate(node.left)
    return rightRightCase(node)
```

Analiza vremenske složenosti

Kao što smo spomenuli na predavanju, broj razina stabla (a samim time i njegova visina) raste logaritamski u odnosu na broj čvorova. Kako kod svakog binarnog stabla insert operacija košta O(h) vremena, a znamo da je h = O(lgn), možemo zaključiti da insert jednog ključa u stablo košta O(lg n).