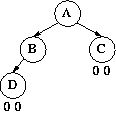
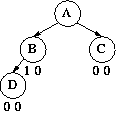
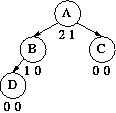
Is it height-balanced?

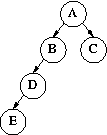
Ex1



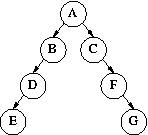




Ex2

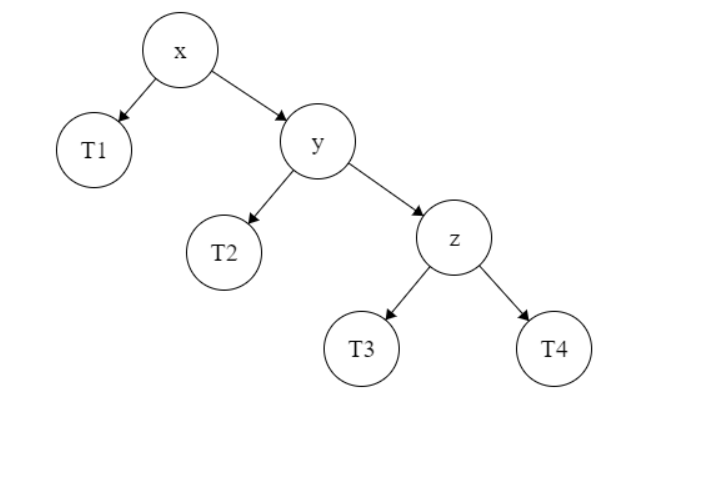


Ex3

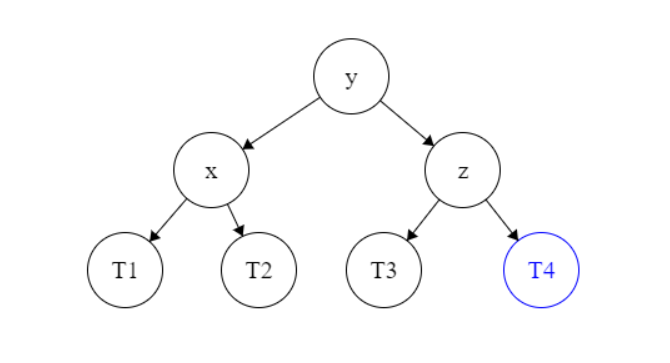


Right Right Case

Unbalanced

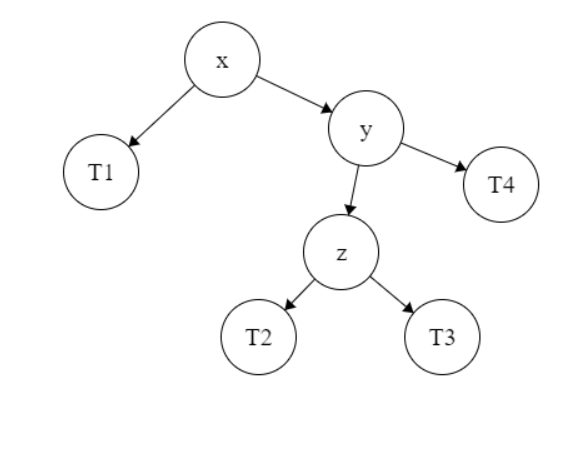


Balanced (left rotate x)

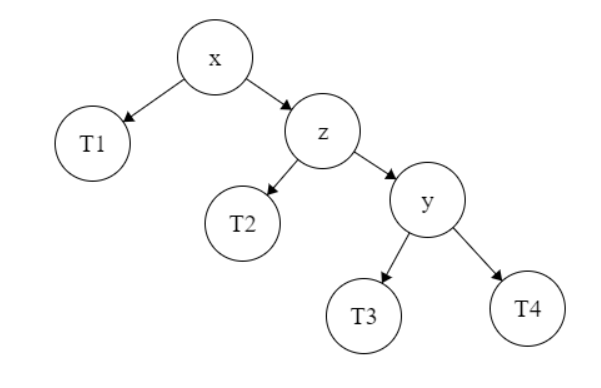
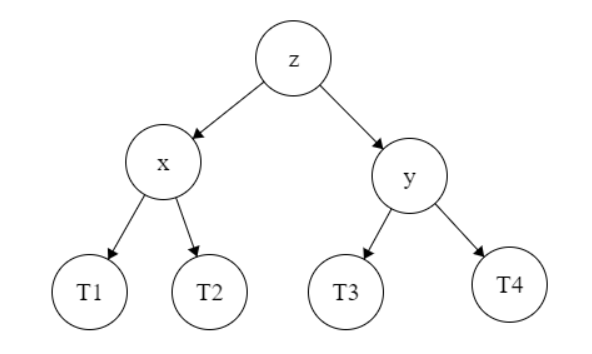


Right Left Case

Unbalanced

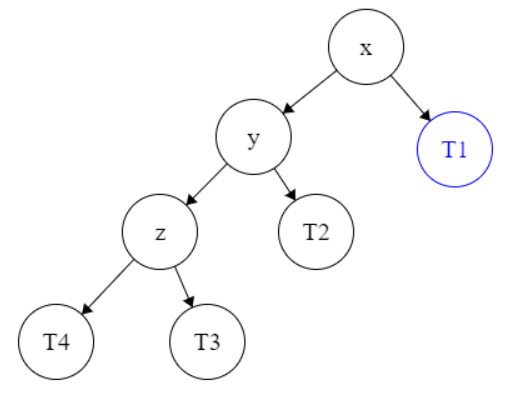


Balanced (right rotate z & left rotate x)

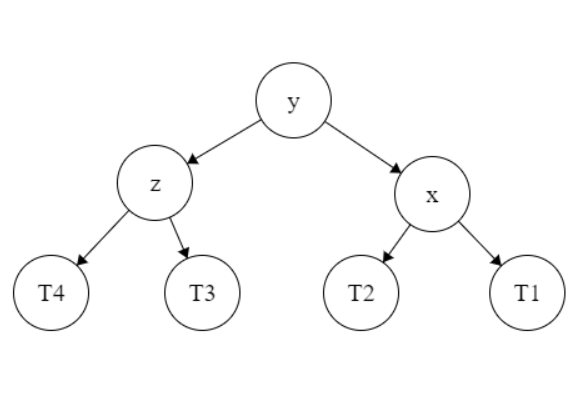
right rotate z  left rotate x

Left Left Case

Unbalanced

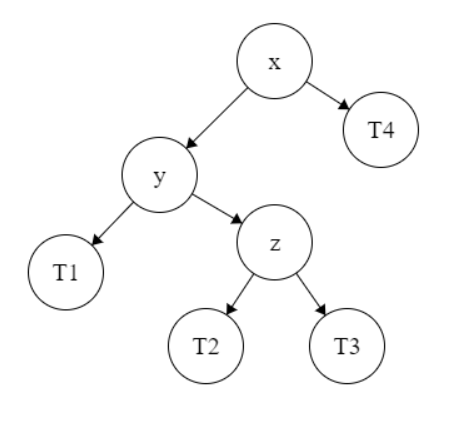


Balanced (right rotate x)

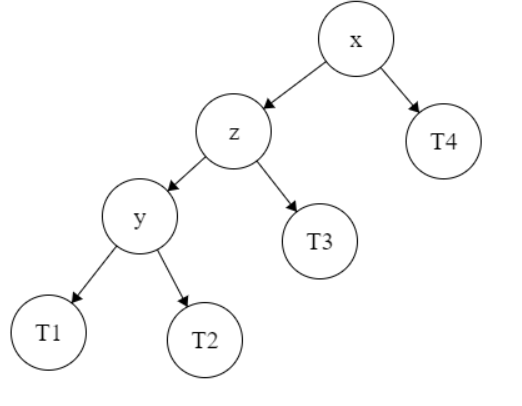


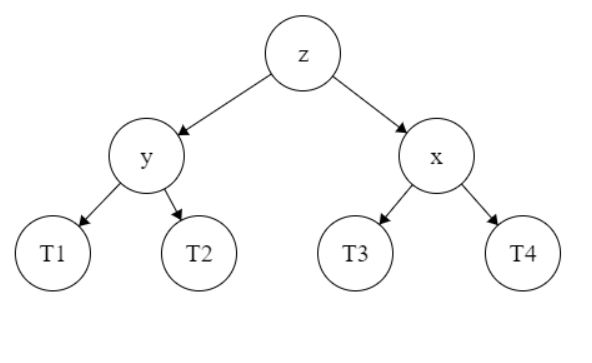
Left Right Case

Unbalanced

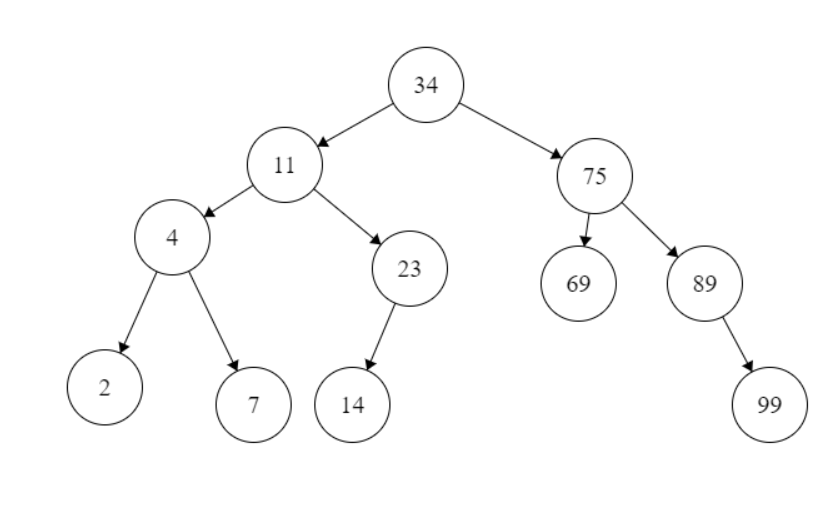


Balanced (left rotate y & right rotate x)

left rotate y  right rotate x



Insert following keys into an initially empty AVL tree. Discuss the cases of balancing that appear. Keys are: 4, 23, 11, 89, 34, 2, 7 14, 75, 69, 99, 80.



RR case -> rotate to left node

RL case -> rotate to right node

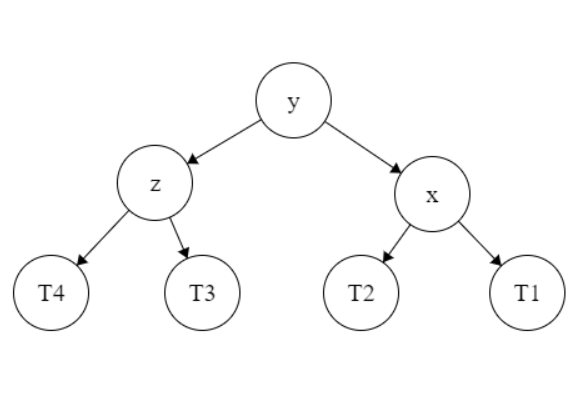
rotate to left node

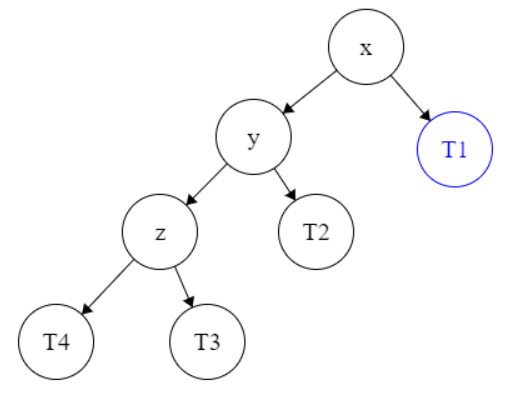
LL case -> rotate to right node

LR case-> rotate to left node

rotate to right node

implement: rotateLeft()

 rotateRight()



y becomes root

x becomes right child of y

T2 becomes left child of x

rotateRight (Node x) {

        Node y = x.left;

        Node T2 = y.right;

        y.right = x;

        x.left = T2;

        return y;

    }