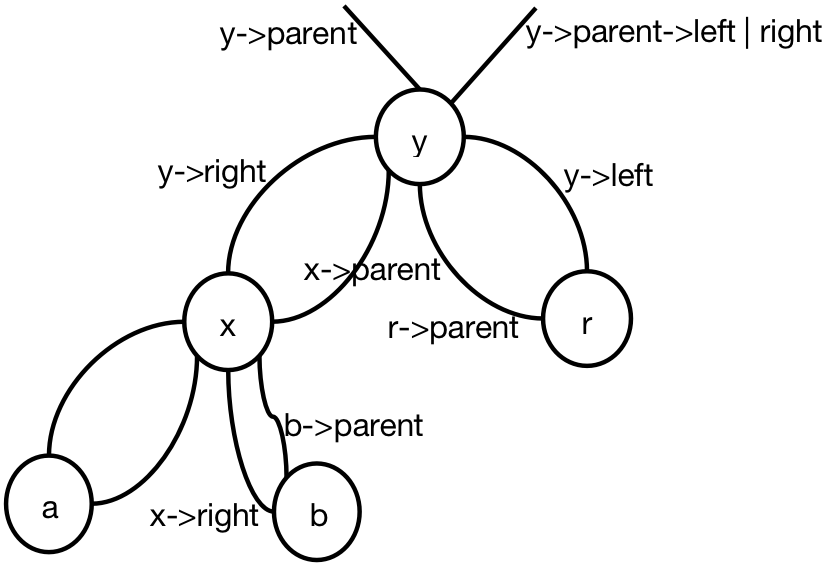
Red Black Tree – Rotation

*Description:*

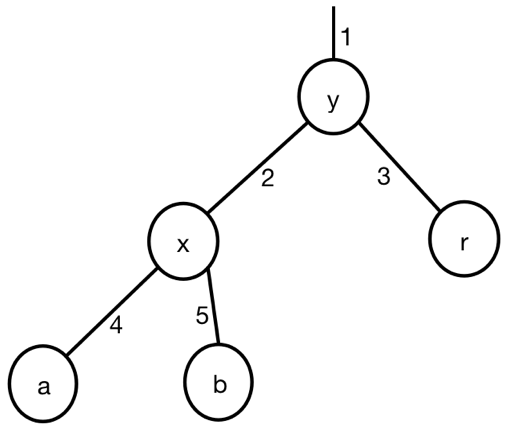
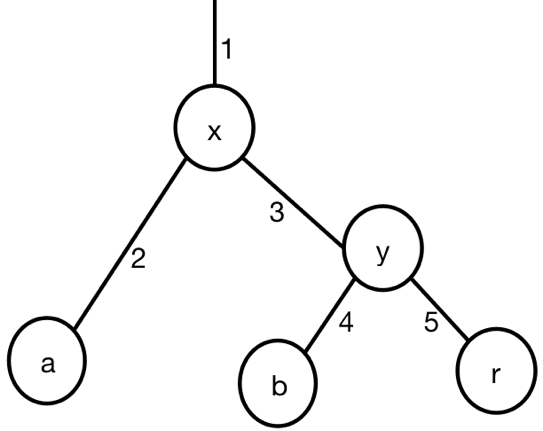
Query Tree Operation TREE-INSERT and TREE-DELETE has been operated on the Red Black Tree with n key nodes, and runtime cost equals to O(logn). However, such two operations have corrupted the structure of Red Black Tree, in order to keep the inner structure of Red Black Tree, we need to do some changes on Color and Pointer of Nodes.

*Rule:*

It is informed that “Changes on Pointer of Nodes” stands for “Rotation”, which means through using *Rotation of Pointers of Red Black Tree*, therefore we keep the Standard Property of Red Black Tree.



In the following pictures, two kinds of rotation operations have been given: one is *Left Rotation* and another is *Right Rotation*.

Right Rotation

Left Rotation

For Node structure, there have four fields that need to be considered, including:

* node.value
* *node.left*
* *node.right*
* *node.parent*

There does not have any changes on node.value, since it would never change even any rotation operations. However, since we can not generate the dangling pointers for each node, so when operating any node rotation, we need to consider each pointers in Red Black Tree.

Both Right Rotation(T, y) and Left Rotation(T, x), all following fields need to be changed:

* x->parent | y->parent
* x->right | y->right
* x->right->parent | y->right->parent
* y->parent | x->parent
* y->left | x->left
* y->parent->left or right | x->parent->left or right

*Code:*

void Left\_Rotation(T, x) {

y = x->right;

x->right = y->left;

if (y->left != T.nil) {

y->left->parent = x;

}

y->parent = x->parent;

if (x == x->parent->left) {

x->parent->left = y;

} else if (x == x->parent->right) {

x->parent->right = y;

} else if (x->parent == T.nil) {

T.root = y;

}

x->parent = y;

y->left = x;

}

void Right\_Rotation(T, y) {

x = y->left;

y->left = x->right;

if (x->right != T.nil) {

x->right->parent = y;

}

x->parent = y ->parent;

if (y == y->parent->left) {

y->parent->left = x;

} else if (y == y->parent->right) {

y->parent->right = x;

} else if (y->parent == T.nil) {

T.root = x;

}

y->parent = x;

x->right = y;

}