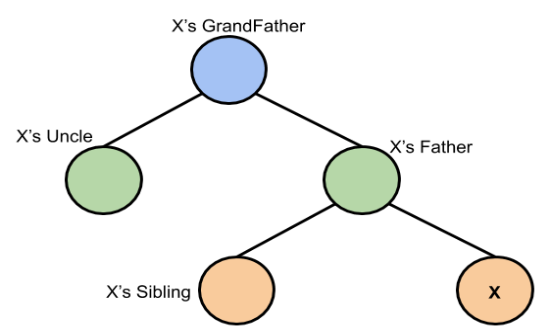
**Rules That Every Red-Black Tree Follows:**

1. Every node has a color either red or black.
2. The root of the tree is always black.
3. There are no two adjacent red nodes (A red node cannot have a red parent or red child).
4. Every path from a node (including root) to any of its descendants’ NULL nodes has the same number of black nodes.
5. All leaf nodes are black nodes.



searchElement (root, value)

node = root;

While (node != null)

If (value == node.value)

return true;

Else If (value < node.value)

node = node.left;

Else If (value > node.value)

node = node.right;

return false;

addElement (root, value)

node = root;

While (node != null)

If (value < node.value)

node = node.left;

Else If (value > node.value)

node = node.right;

Else If (value == node.value)

Return false;

node = new node;

node.value = value;

If (node == root)

node.colour == black;

return true;

Else

node.colour = red;

If (node.parent.colour == black)

Return true;

Else if (node.parent.colour == red)

If (node.uncle.colour == red)

Do

Node.parent.colour = black;

Node.uncle.colour = black;

If (node.grandfather != root)

Node.grandfather.colour = red;

Else

Node.grandfather.colour = black;

Return true;

Node = node.grandfather;

While (node != null && (node.parent.colour == red && node.uncle.colour == red))

If (node.parent.colour == red && node.uncle.colour == black)

If (node.grandparent.left == node.parent && node.parent.left == node) //LL

RightRotation (node);

Else If (node.grandparent.right == node.parent && node.parent.right == node) //RR

LeftRotation (node);

Else If (node.grandparent.left == node.parent && node.parent.right == node) //LR

node.grandparent.left = node;

node.parent.right = node.left;

node.left = node.parent;

node.parent = node.grandparent;

node.parent.parent = node; //change tree to LL form

RightRotation (node.parent); //node at the bottom

Else If (node.grandparent.right == node.parent && node.parent.left == node) //RL

node.grandparent.right = node;

node.parent.left = node.right;

node.right = node.parent;

node.parent = node.grandparent;

node.parent.parent = node; //change tree to RR form

LeftRotation (node.parent); //node at the bottom

Return true;

RightRotation (node)

If (node.grandparent.parent.left == node.grandparent)

node.grandparent.parent.left = node.parent

Else if (node.grandparent.parent.right == node.grandparent)

node.grandparent.parent.right = node.parent

node.grandparent.left = node.parent.right;

node.parent.right = node.grandparent;

node.parent.parent = node.grandparent.parent;

node.grandparent.parent = node.parent;

If (node.parent.parent == null)

Node.parent = root;

LeftRotation (node)

If (node.grandparent.parent.left == node.grandparent)

node.grandparent.parent.left = node.parent

Else if (node.grandparent.parent.right == node.grandparent)

node.grandparent.parent.right = node.parent

node.grandparent.right = node.parent.left;

node.parent.left = node.grandparent;

node.parent.parent = node.grandparent.parent;

node.grandparent.parent = node.parent;

If (node.parent.parent == null)

Node.parent = root;

deleteElement (root, value)

node = root;

While (node != null)

If (value < node.value)

node = node.left;

Else If (value > node.value)

node = node.right;

Else If (value == node.value)

Delete (node);

Return false;

Delete (node)

If (node.colour == red || node.child.colour == red)

Node = node.child;

Node.colour = black;

Return true;

Else if (node.colour == black && node.child.colour == black) //null is also black

Node = node.child;

Node.colour = double black

If (node.colour == double black)

If (node.sibling == black && node.sibling.child == red)

If (node.parent.left == node.sibling && node.sibling.left == node.sibling.child)

RightRotation (node.sibling.child);

Else if (node.parent.right == node.sibling && node.sibling.right == node.sibling.child)

LeftRotation (node.sibling.child);

Else if (node.parent.left == node.sibling && node.sibling.right == node.sibling.child)

node.sibling.parent.left = node.sibling.child;

node.sibling.right = node.sibling.child.left;

node.sibling.child.left = node.sibling;

node.sibling.child.parent = node.sibling.parent;

node.sibling.parent = node.sibling.child; //change tree to LL

RightRotation (node.sibling); //node at the bottom

Else if (node.parent.right == node.sibling && node.sibling.left == node.sibling.child)

node.sibling.parent.right = node.sibling.child;

node.sibling.left = node.sibling.child.right;

node.sibling.child.right = node.sibling;

node.sibling.child.parent = node.sibling.parent;

node.sibling.parent = node.sibling.child; //change tree to RR

LeftRotation (node.sibling); //node at the bottom

Else If (node.sibling == black && node.sibling.child == black)

Do

Node.colour = black;

Node.sibling.colour = red;

If (node.parent.colour == black)

Node.parent.colour = double black;

Else If (node.parent.colour == red)

Node.parent.colour = black;

Node = node.parent;

While (node.parent.colour == double black)

Return;

Else if (node.sibling == red)

If (node.parent.left == node.sibling)

Node.sibling.colour = black;

Node.sibling.right.colour = red;

RightRotation(node.sibling.left)

Else if (node.parent.right == node.sibling)

Node.sibling.colour = black;

Node.sibling.left.colour = red;

LeftRotation(node.sibling.right)

Else if (node.child == root)

Node = node.child;

Node = black;

Return true;