

RED BLACK TREE

RAJA NANDINI



Topics

- What is Red Black tree?
- Why Red Black trees needed?
- Properties of Red Black trees
- Comparison between Red Black tree and Binary search tree
- Insertion Fixing violations
- Right rotation & Left rotation
- Rotation Cases
- Insertion
- Advantages, Applications
- Q & A
- Recap
- References

What is Red Black tree?

- Red Black tree is a self balancing binary search tree.
- Every node has a color either red or black.
- In Extended, NULL links are replaced by special nodes.

Why Red black trees are needed?

- RB Tree is a specialized version of a BST.
- Because RB tree guarantees time $O(\log n)$ for search, insertion and deletion even in worst case also.
- But BST give $O(n)$ time for search, insertion and deletion in worst case.

Time complexity	Binary Search Tree (search, Insertion & Deletion)	Red Black tree (Search, Insertion & Deletion)
Worst case	$O(n)$	$O(\log n)$
Average case	$O(\log n)$	$O(\log n)$
Best case	$O(1)$	$O(1)$ $O(\log n)$ $O(\log n)$

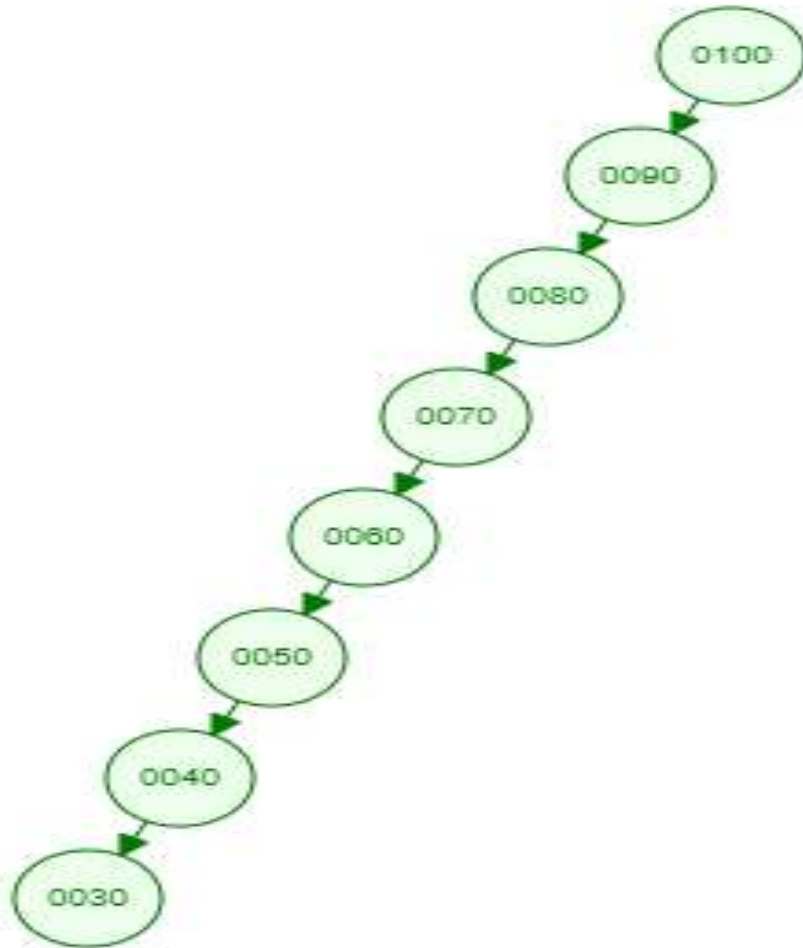
Properties of Red Black Tree

1. Root is always BLACK.
2. All Extended nodes are BLACK.
3. A RED node cannot have RED children , it can have only BLACK children.
4. For each node N , all paths from node N to external nodes contain the same number of BLACK nodes (Same BLACK height) .

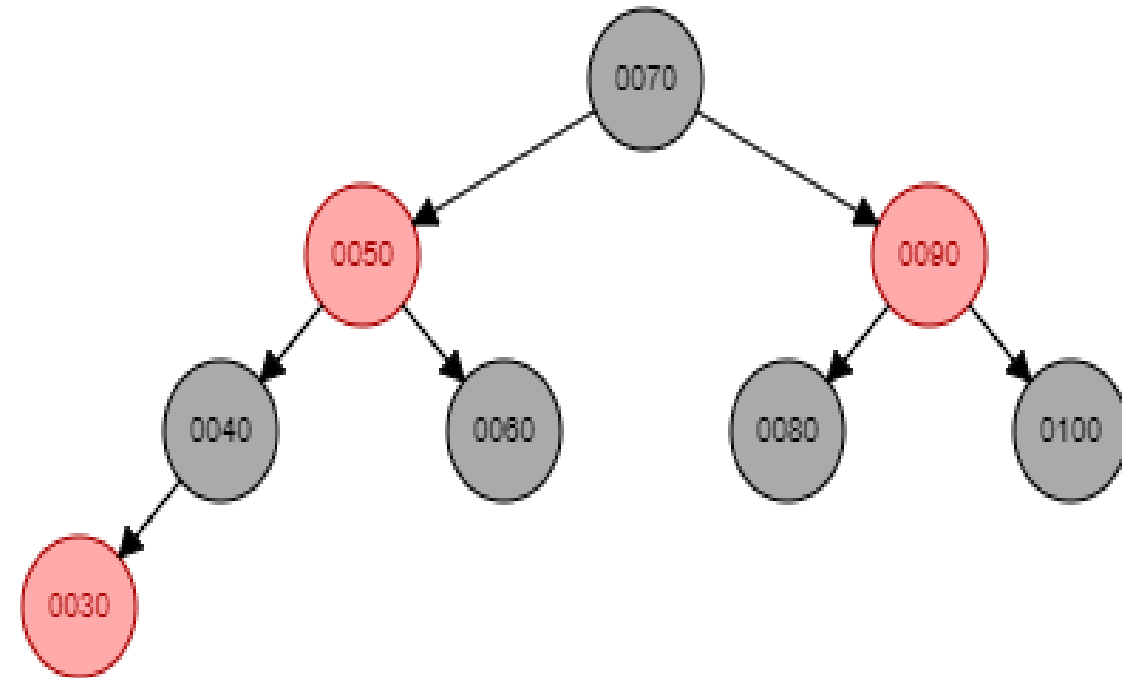
- Both BST and RB trees maintain the BST property.

BST	RB Tree
1) It is not the self balancing tree.	1) It is the self balancing tree.
2) It form long chain of nodes that can cause searches to take linear time.	2) But RB tree guarantees a search operation takes logarithmic time.
3) BST maintain balance by placing nodes based on the values ,as the smallest on the left and largest on the right.	3) Where as in RB tree each node marked with a color and has additional operations defined on it to maintain “balance”

BST vs RB Tree Representation



BST

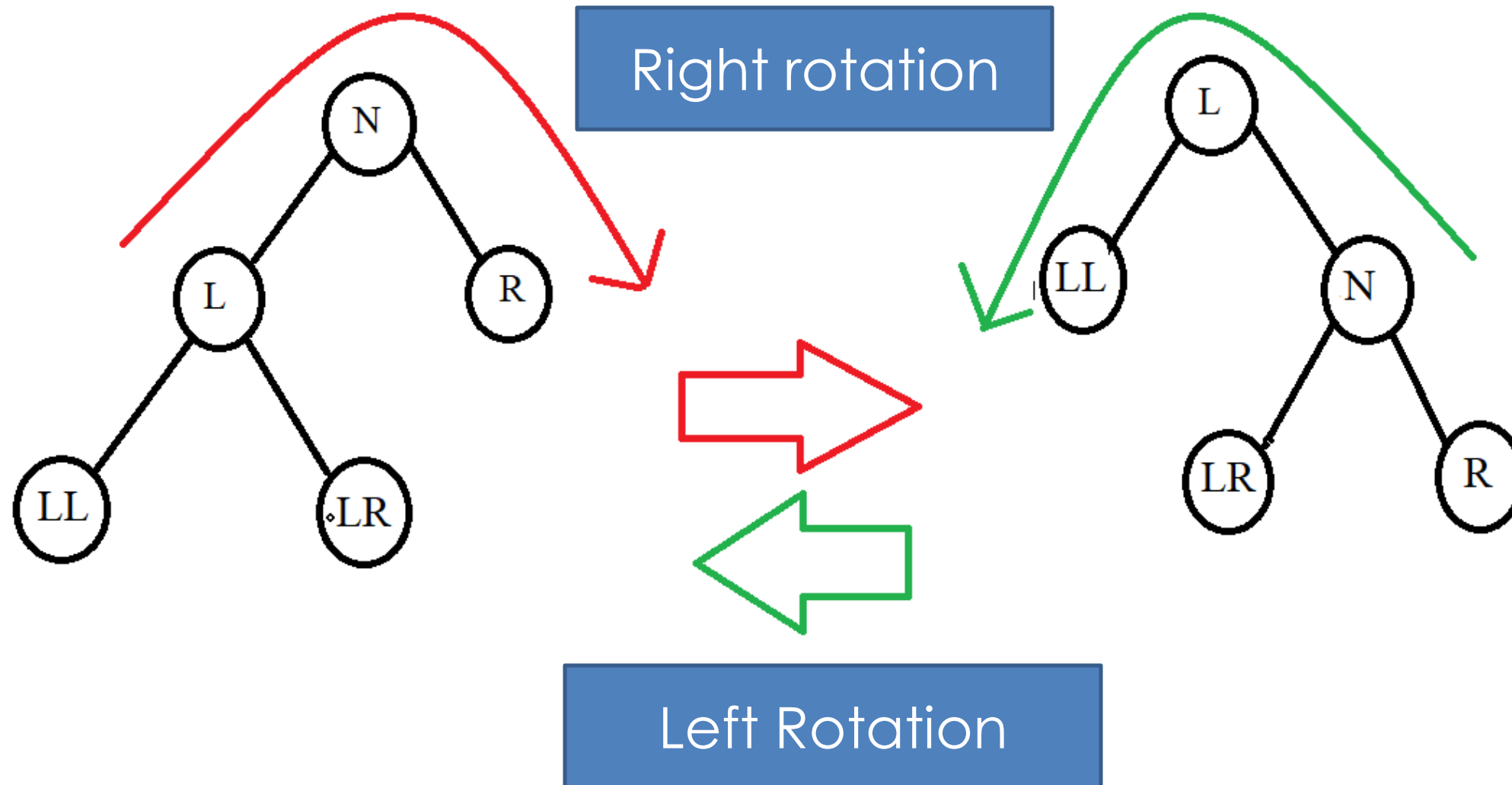


RB Tree

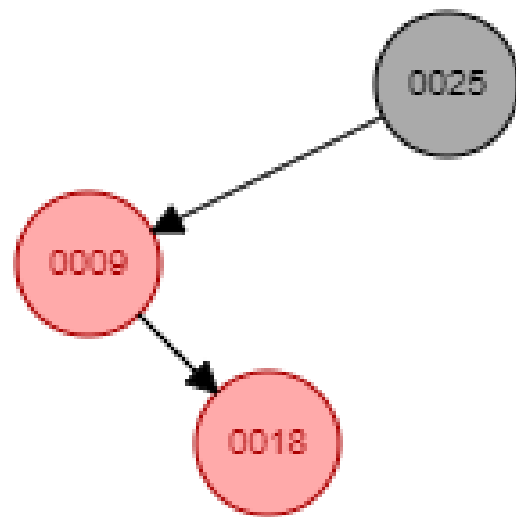
1. If tree is empty ,create new node as root node with color BLACK.
2. If tree is not empty , create new node as leaf node based on BST properties with color RED.
3. If parent of new node is BLACK then exit.
4. If parent of new node is RED , then check the uncle color of new node.
 - a) If uncle color is BLACK or null then do suitable rotation and recolor.
 - b) If uncle color is RED change parent, uncle color to BLACK and grandparent color to RED. If grandparent is root change color of grandparent to BLACK and recheck.

Rotations

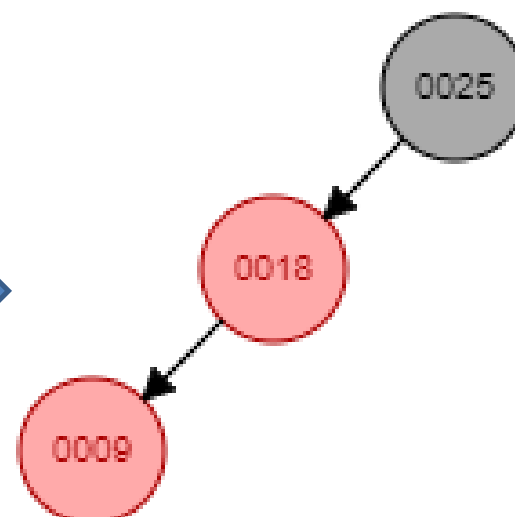
Right rotation & Left rotation



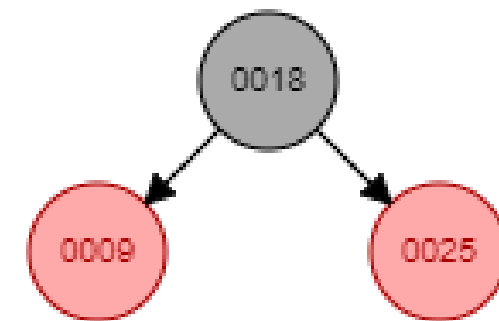
Rotation Cases



L - a

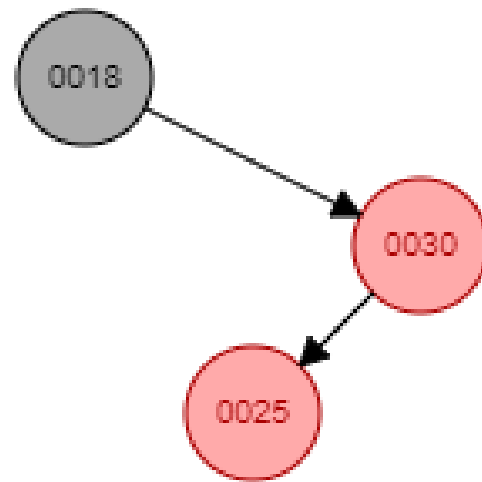


L - b

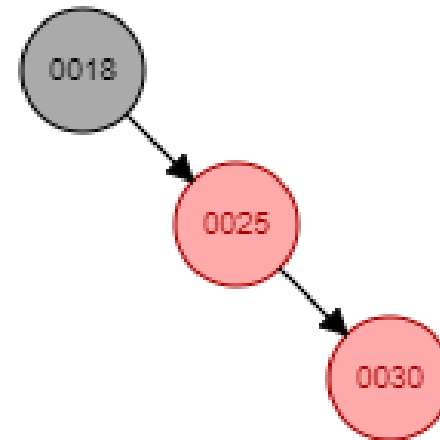


L

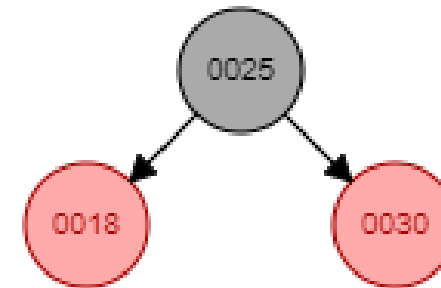
Rotation Cases Contd.



R - a



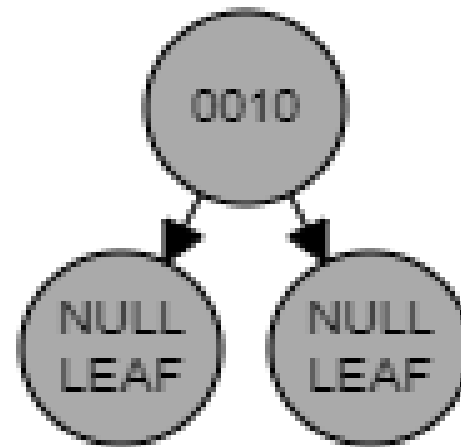
R - b



R

Insertion -Example

1. Insert (root, 10)

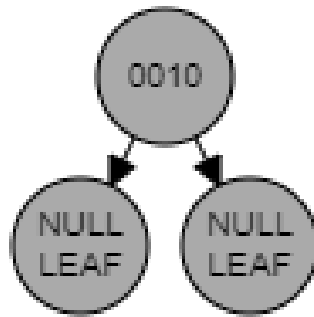


NV-No Violations

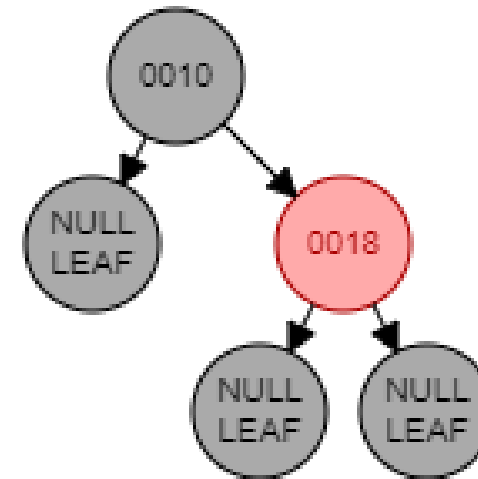
NV

Insertion -Example

2. Insert(root, 18)



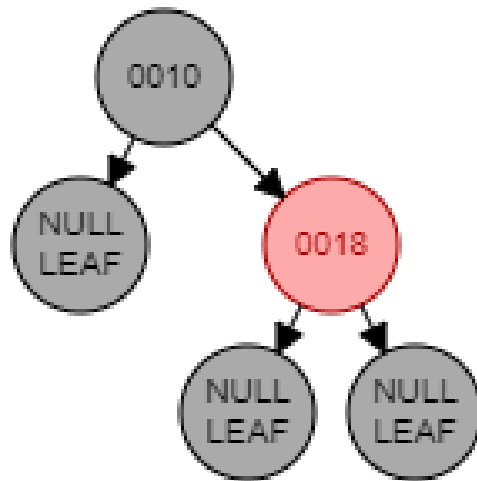
Before - NV



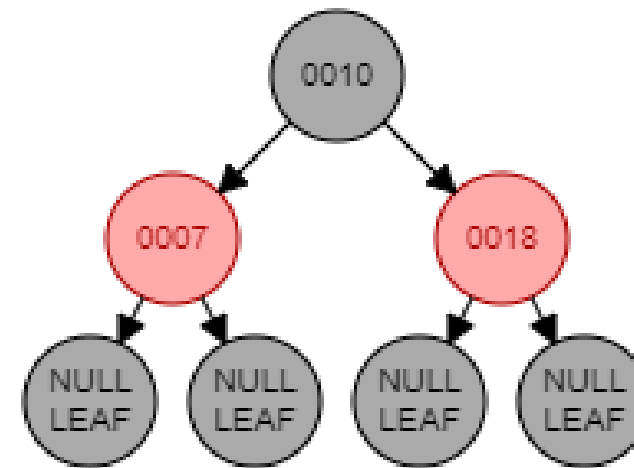
After - NV

Insertion -Example

3. insert(root, 7)



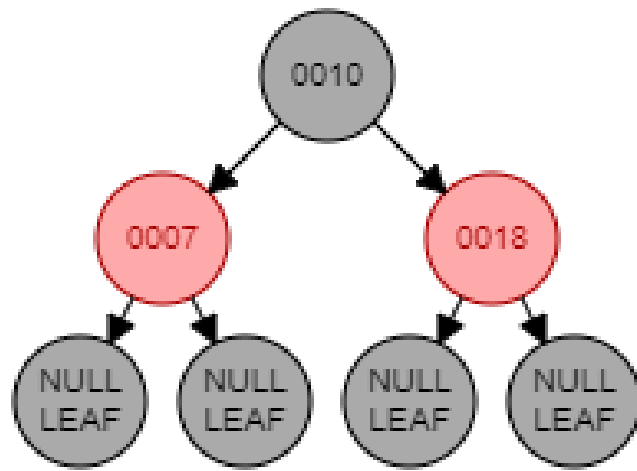
Before - NV



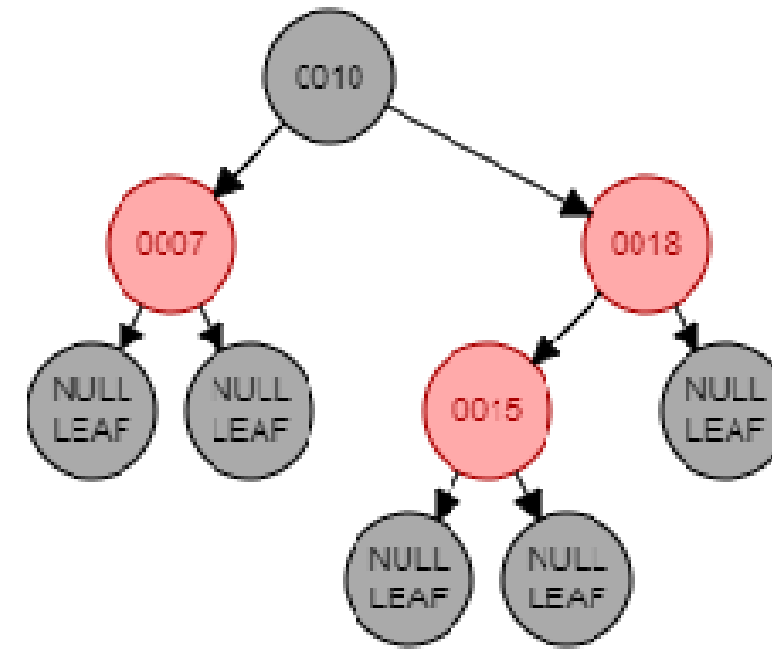
After - NV

Insertion -Example

4. insert(root, 15)

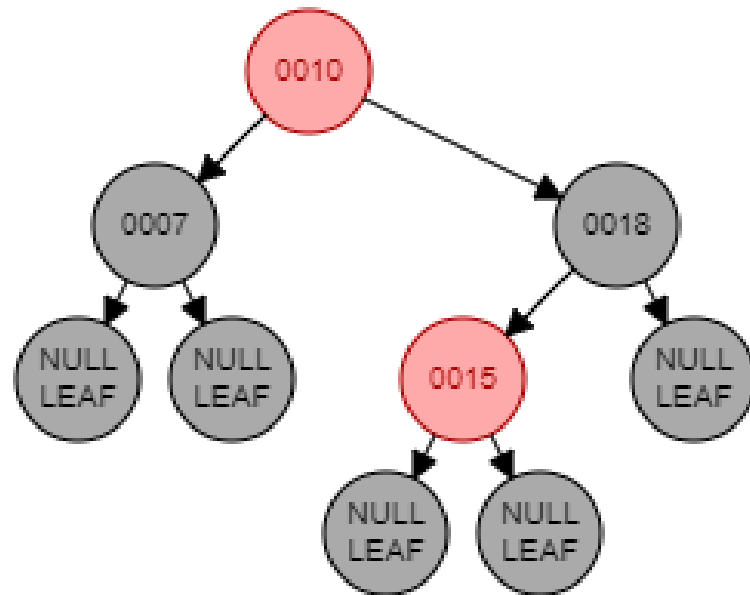


Before - NV

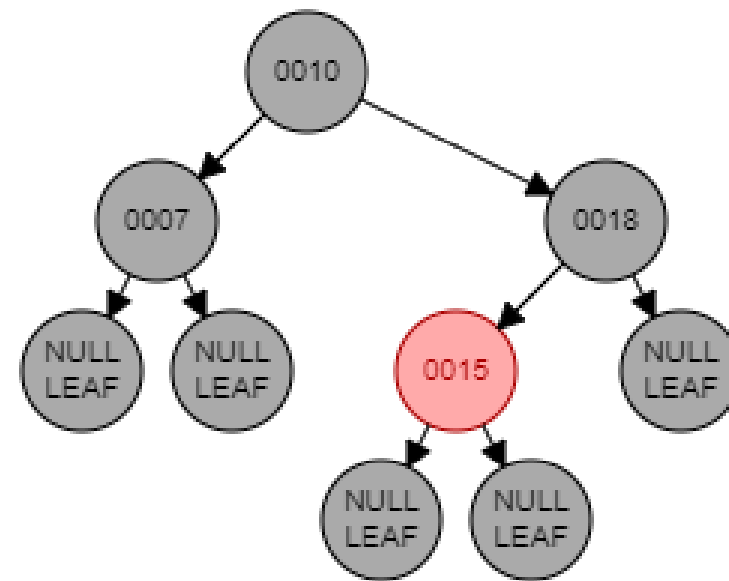


✓

Insertion -Example



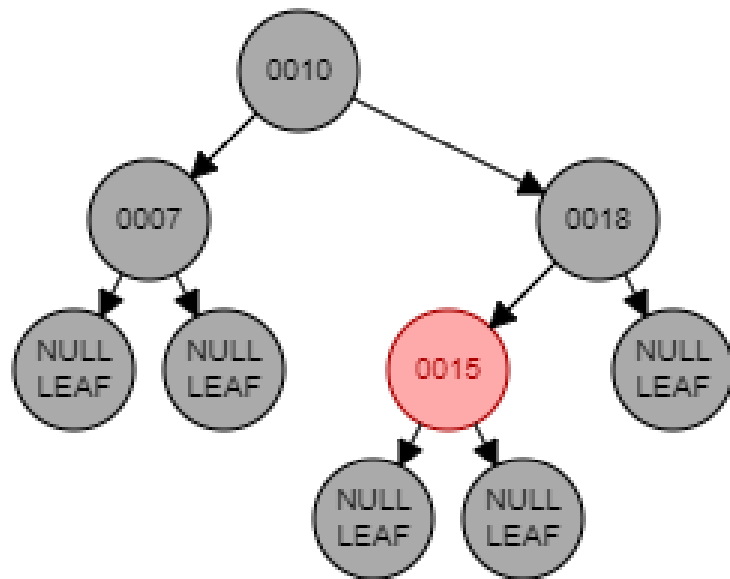
V



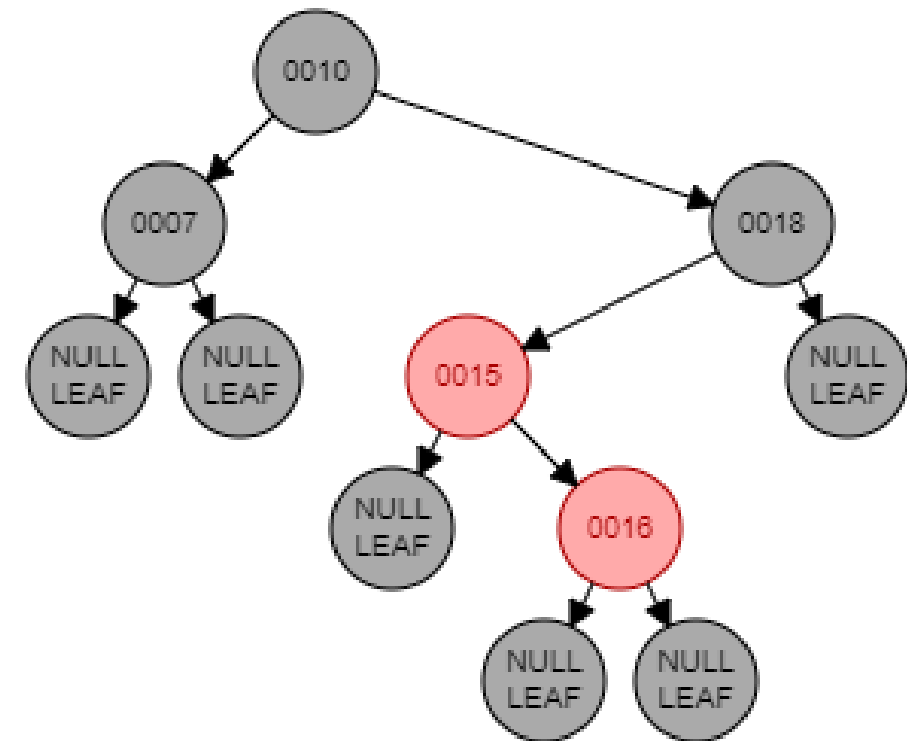
After - NV

Insertion -Example

5. insert(root, 16)



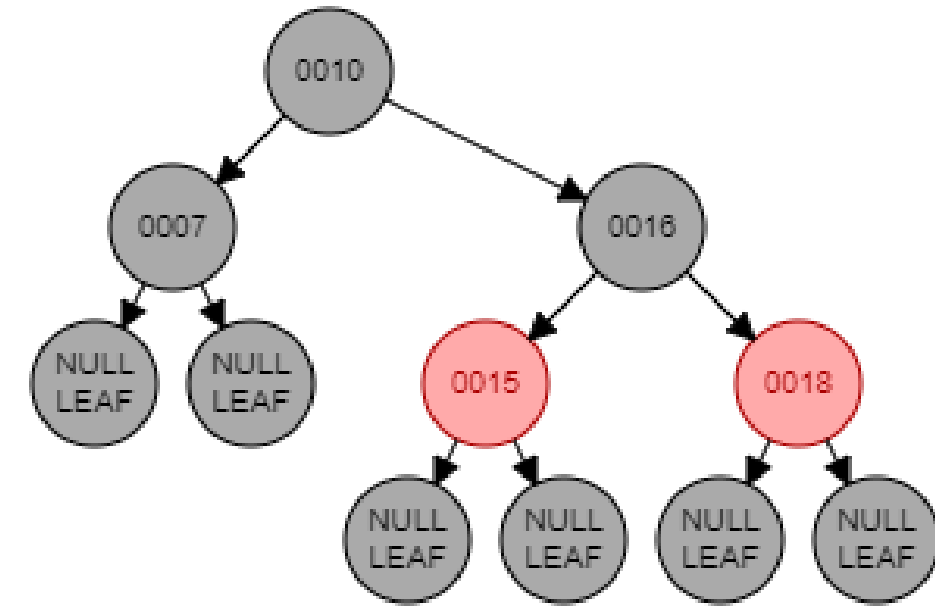
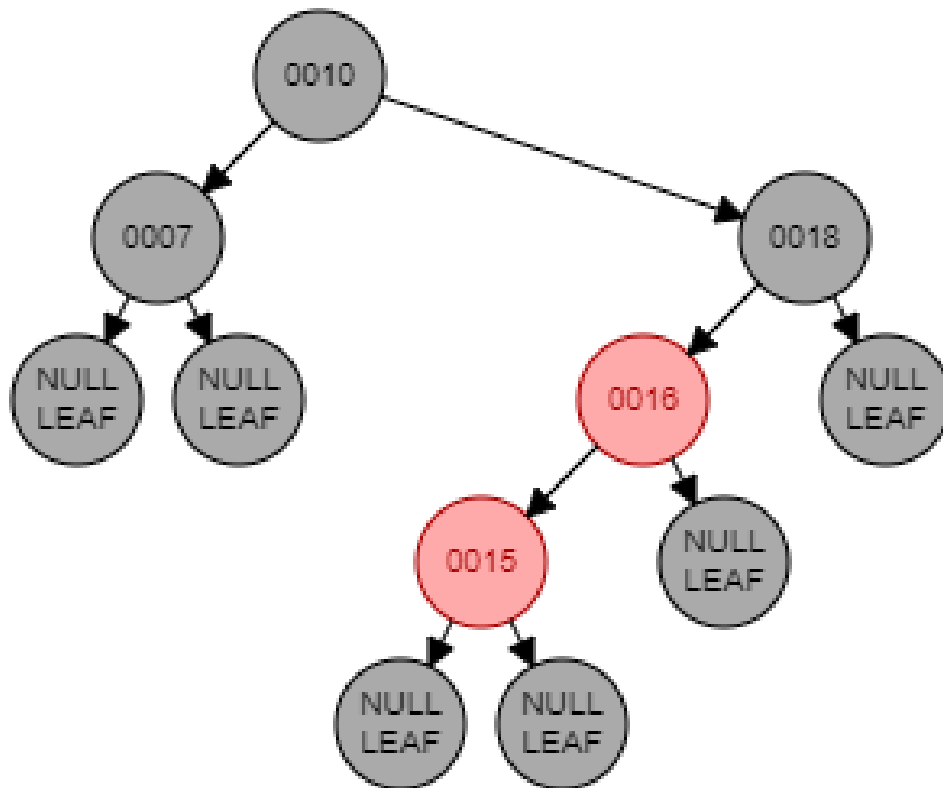
Before - NV



✓

Insertion -Example

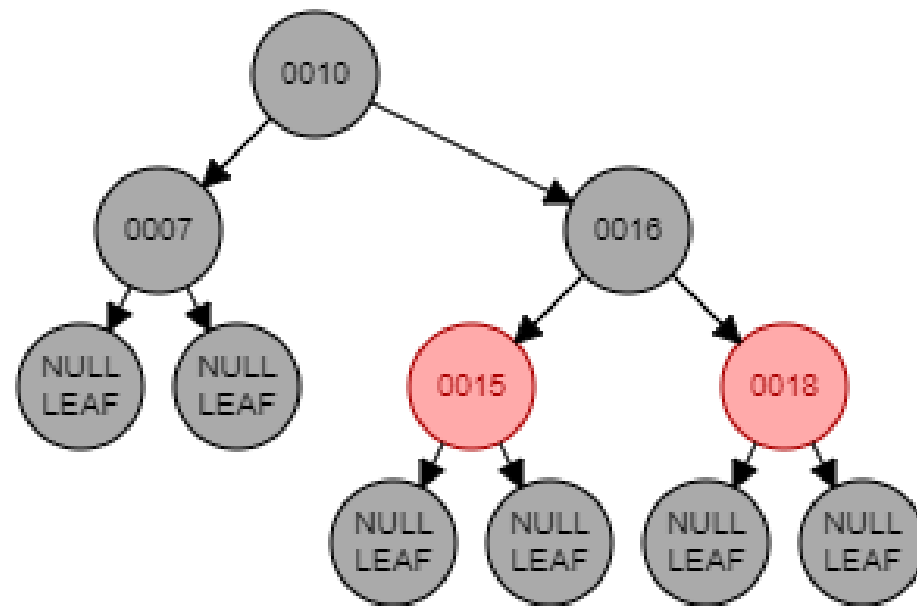
insert(root, 16) Contd.



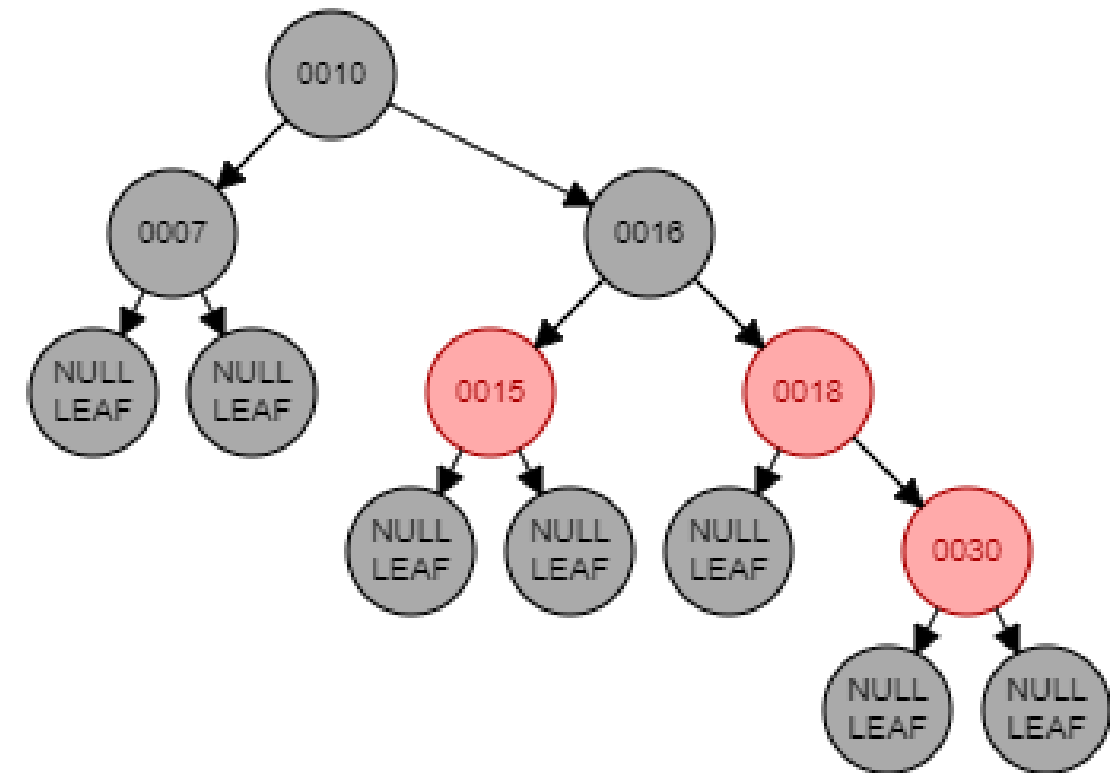
After - NV

Insertion -Example

6. Insert(root, 30)



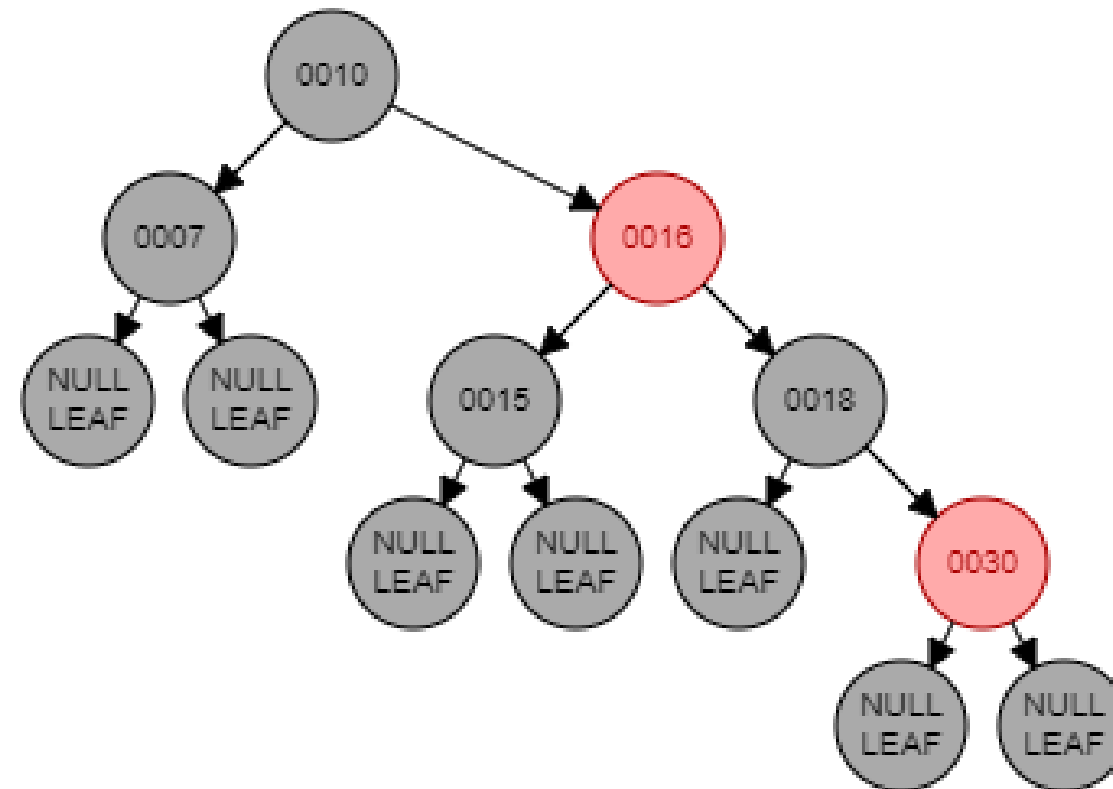
Before - NV



V

Insertion -Example

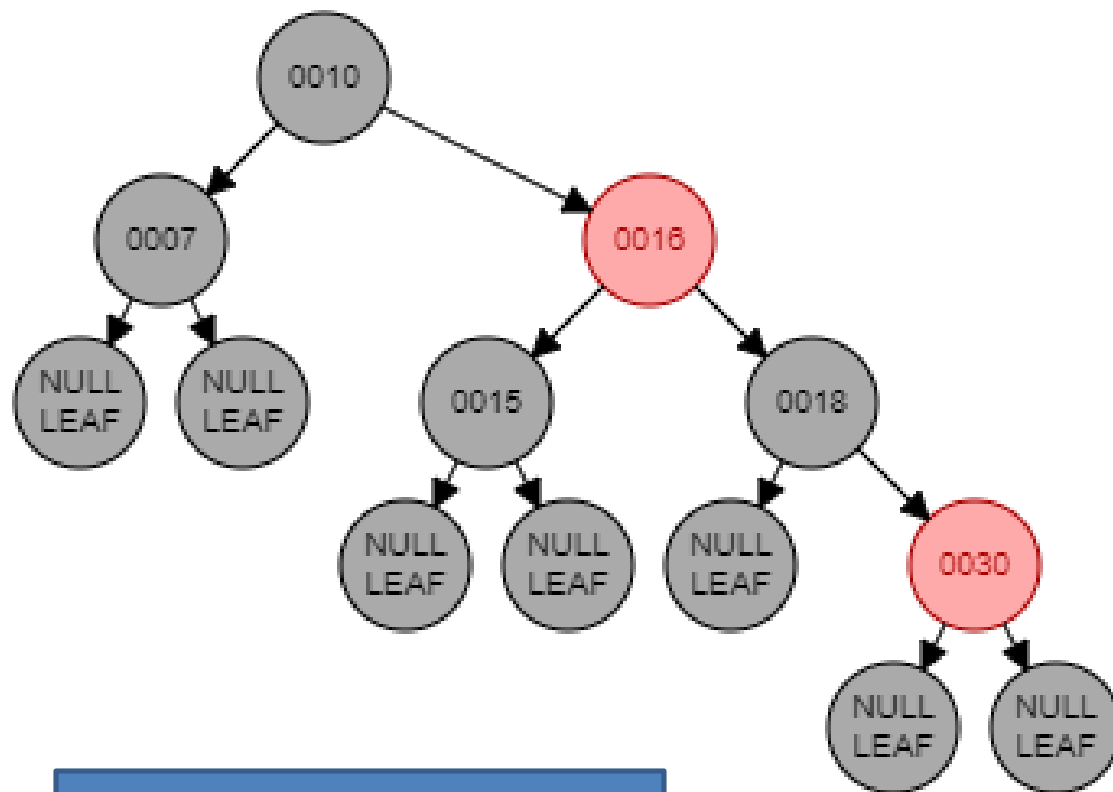
Insert(root, 30) Contd.



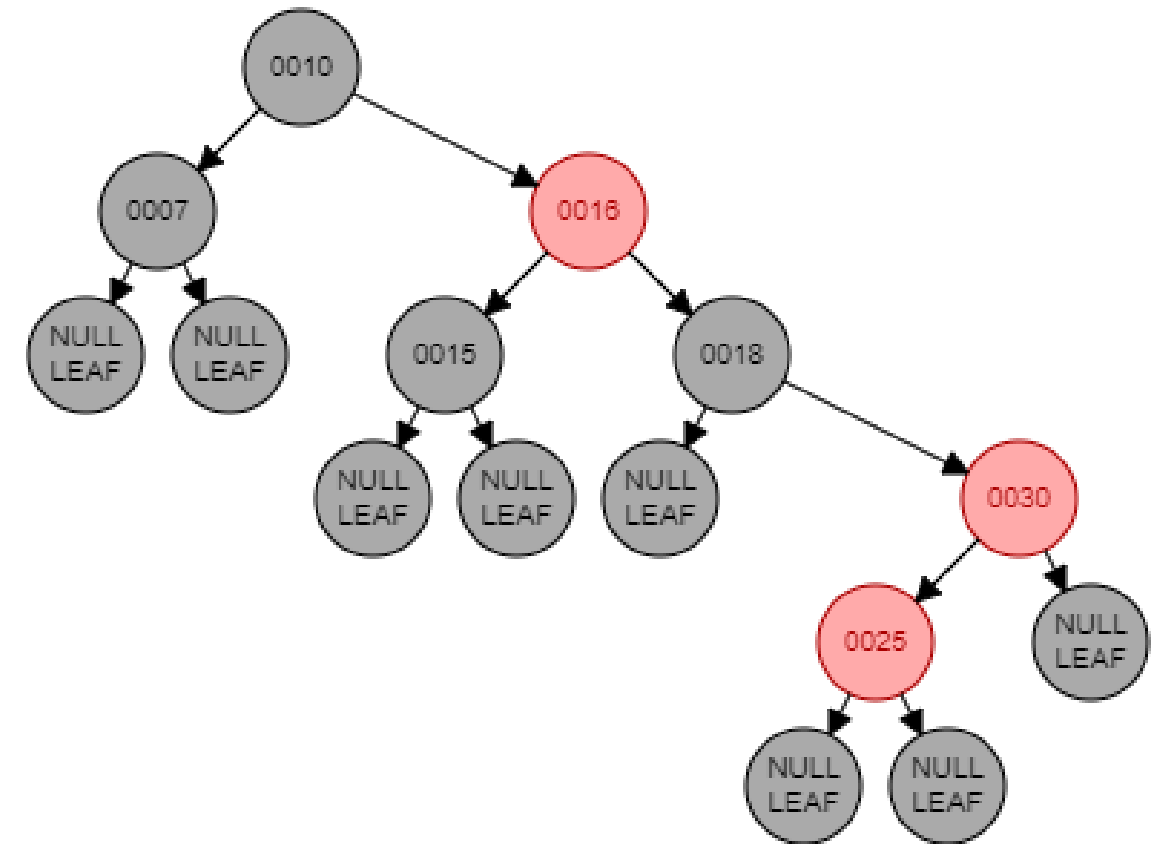
After - NV

Insertion -Example

7. insert(root, 25)



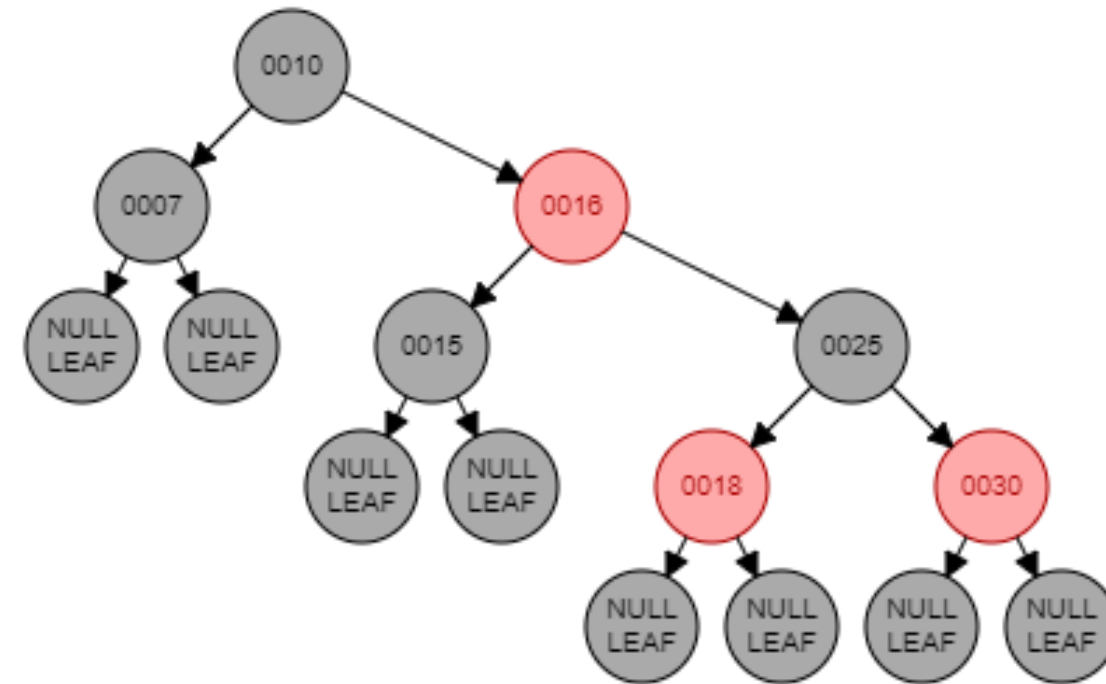
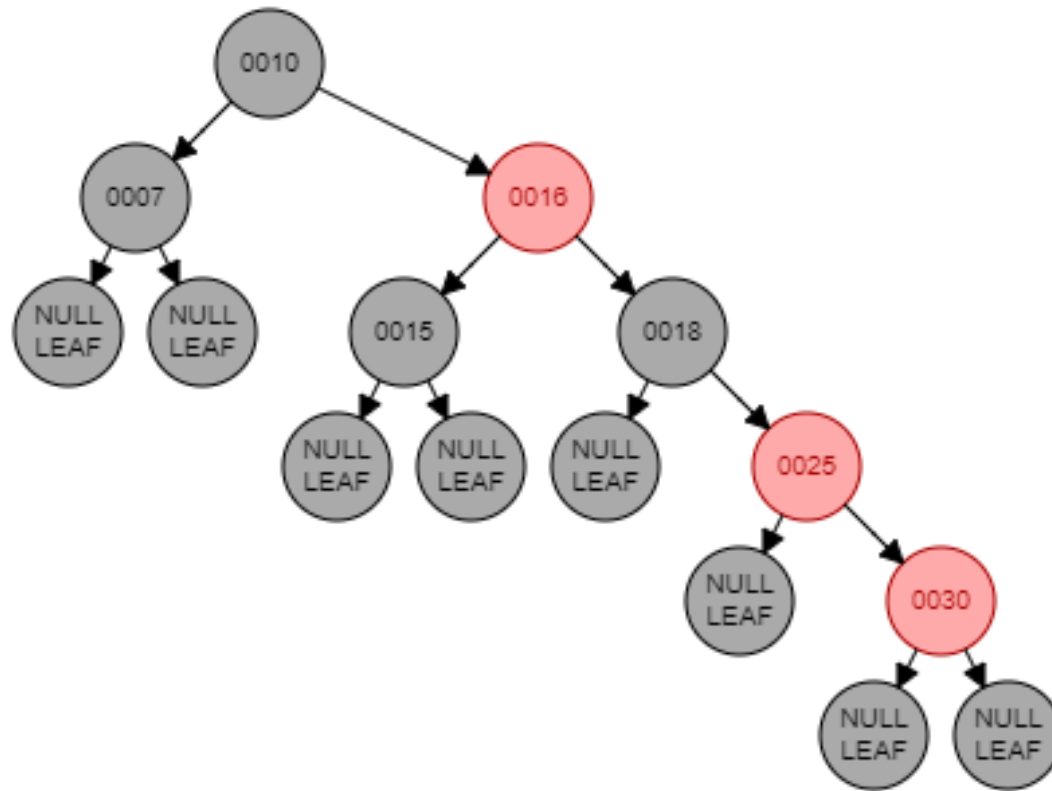
Before - NV



✓

Insertion -Example

insert(root, 25) Contd.

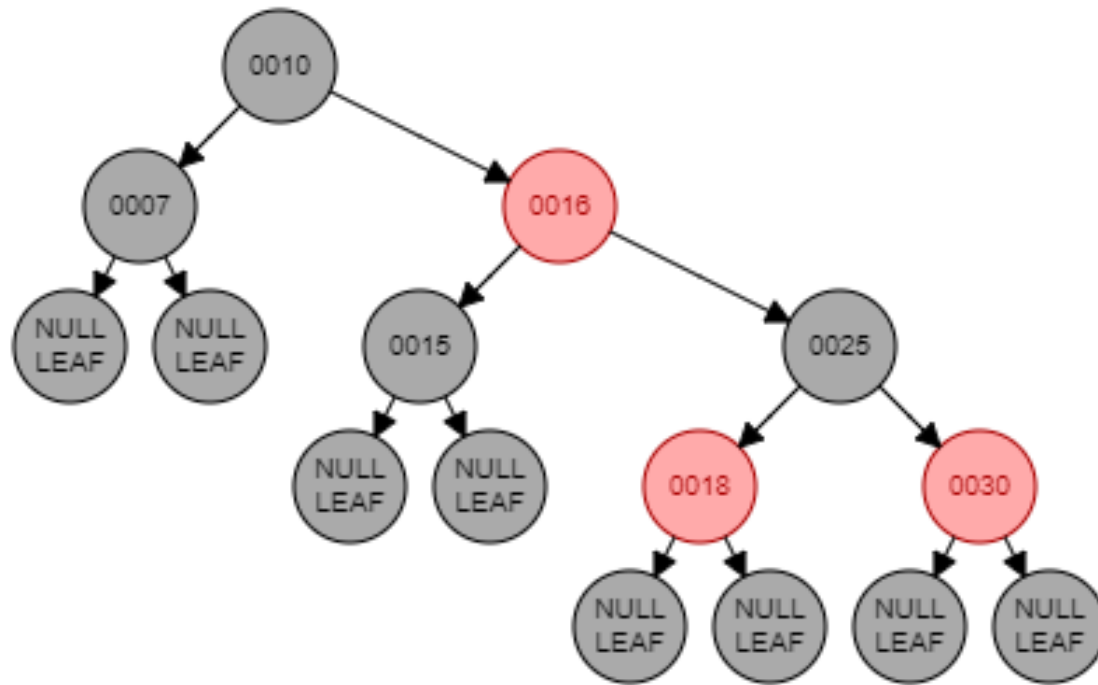


V

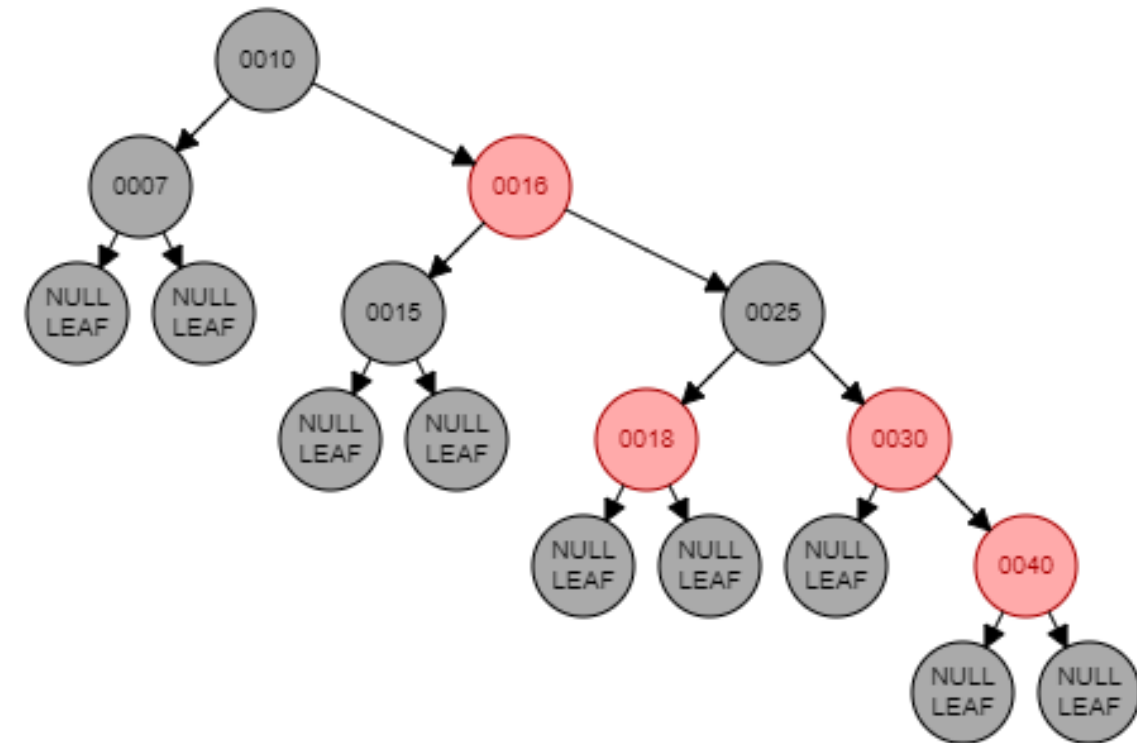
After - NV

Insertion -Example

8. Insert(root,40)



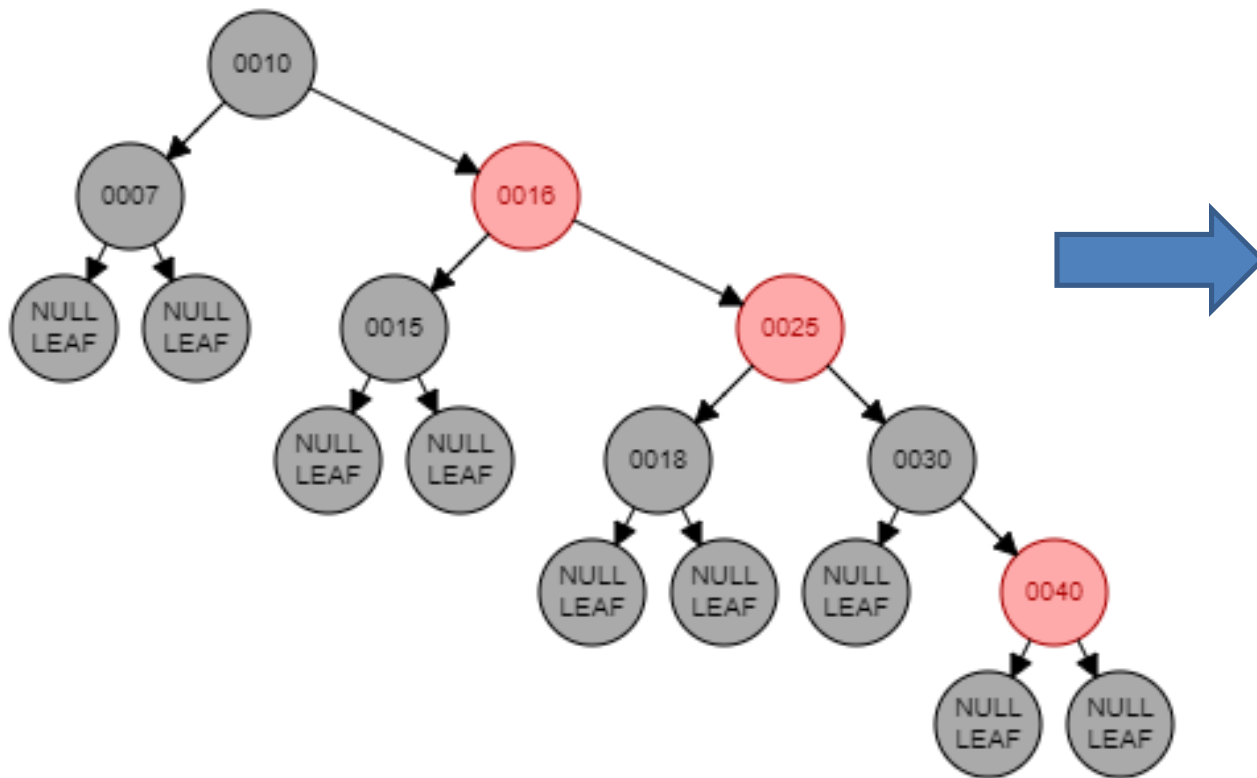
Before - NV



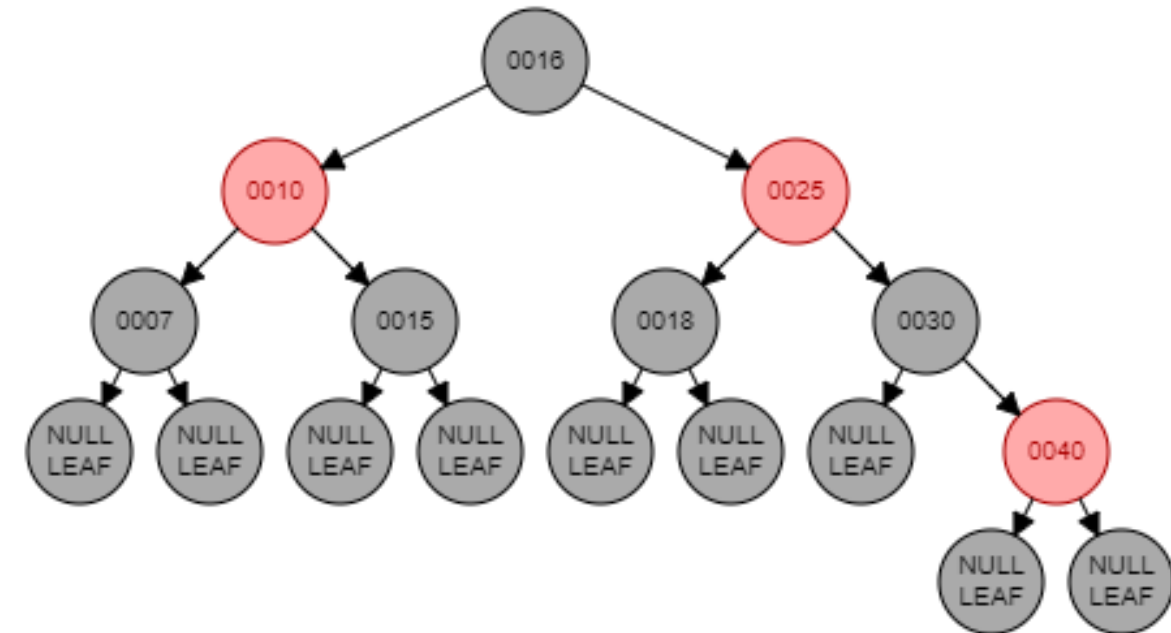
✓

Insertion -Example

Insert(root,40) Contd.



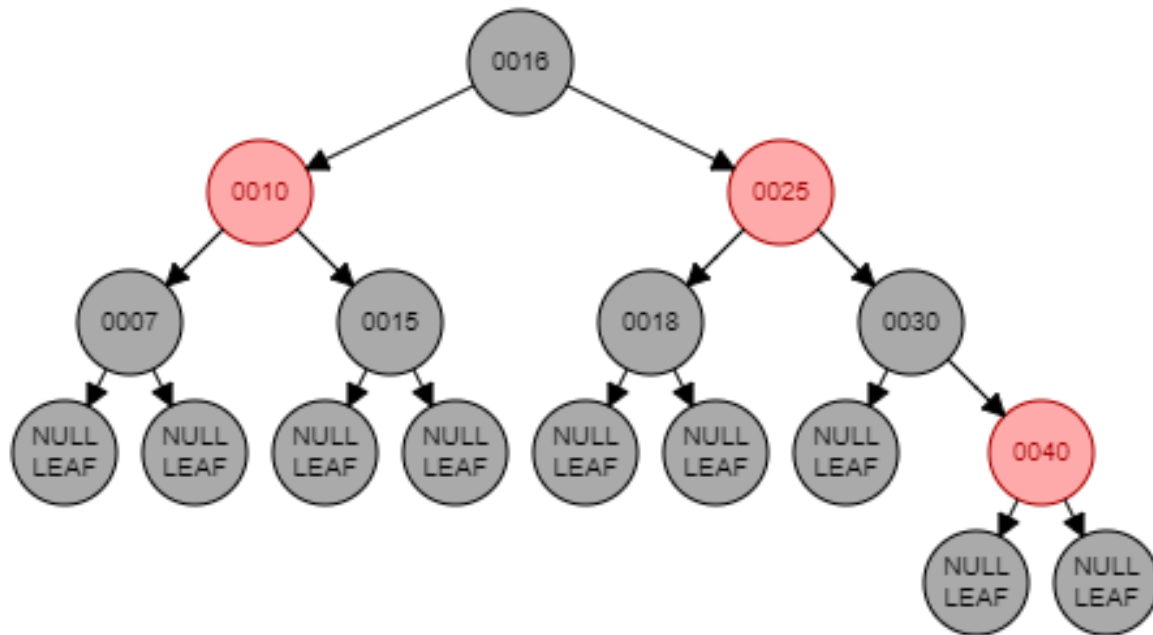
✓



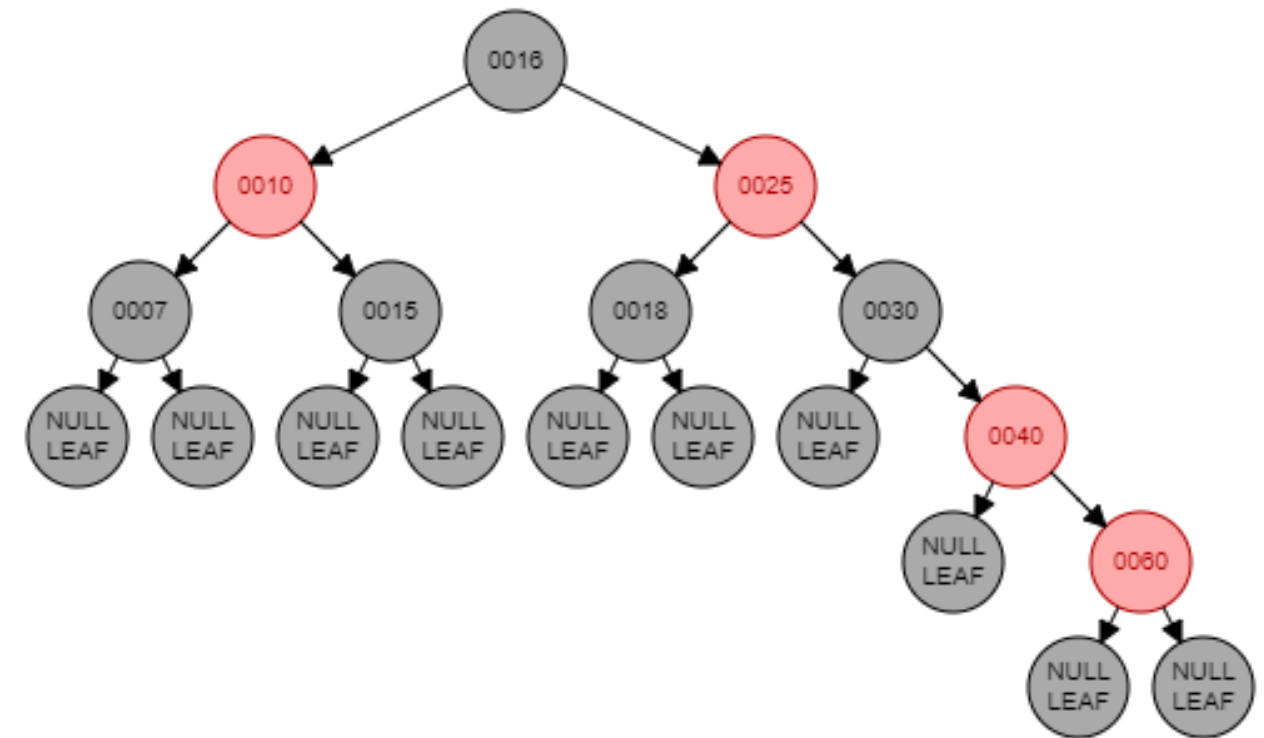
After - NV

Insertion -Example

9. Insert(root, 60)



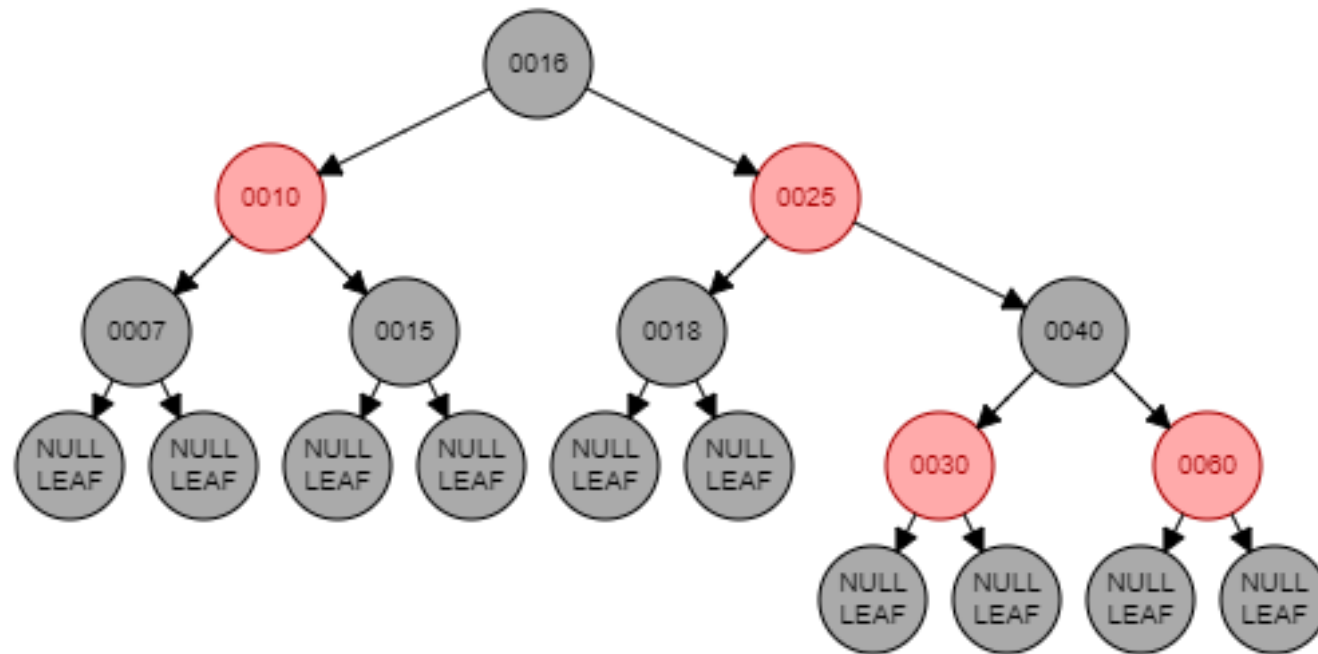
Before



V

Insertion -Example

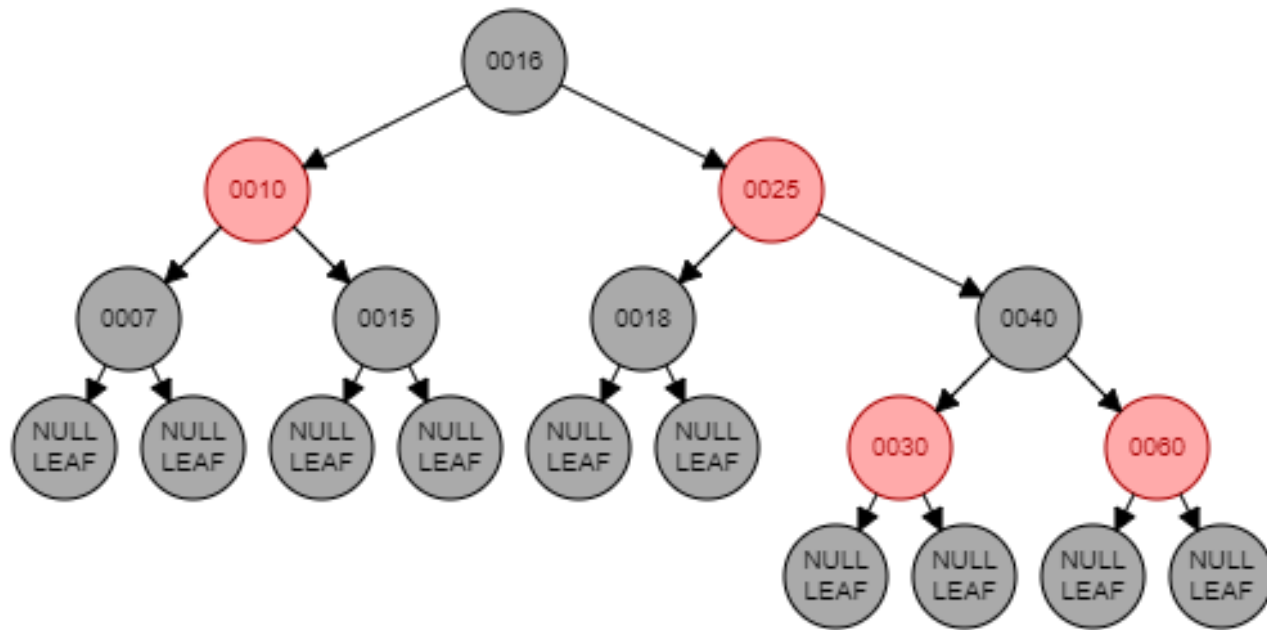
Insert(root, 60) Contd.



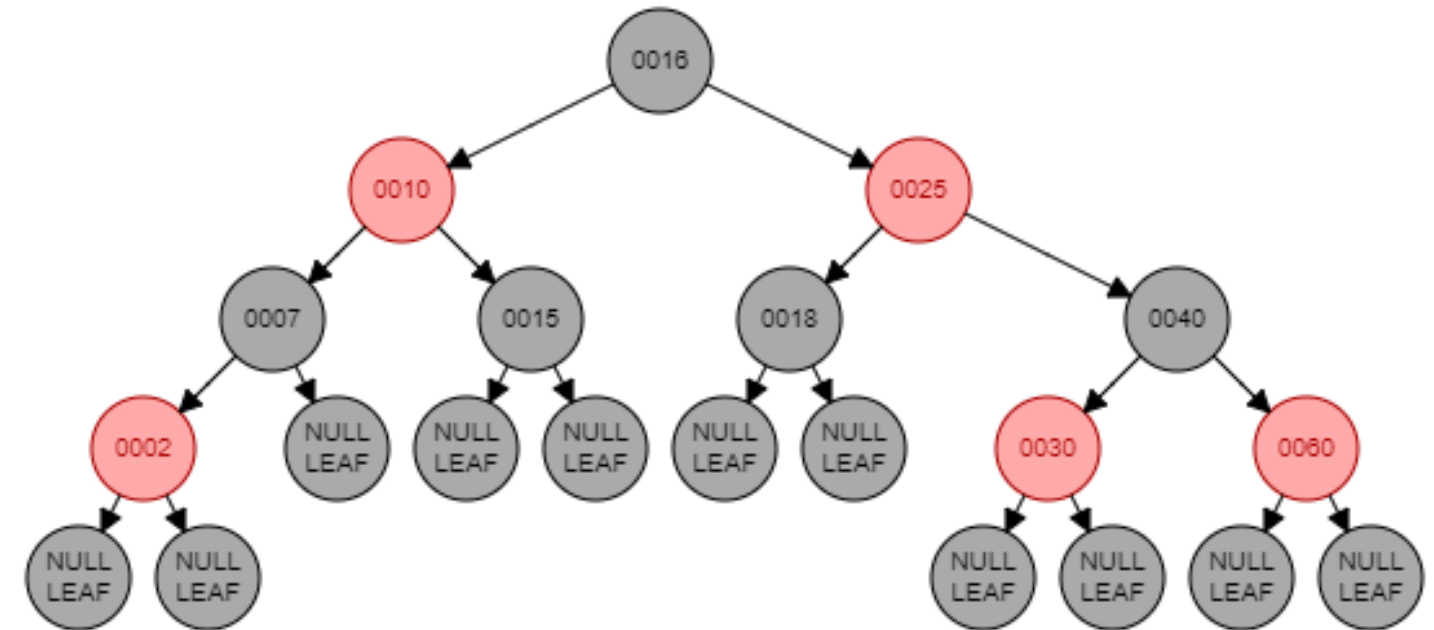
After - NV

Insertion -Example

10. Insert(root, 2)



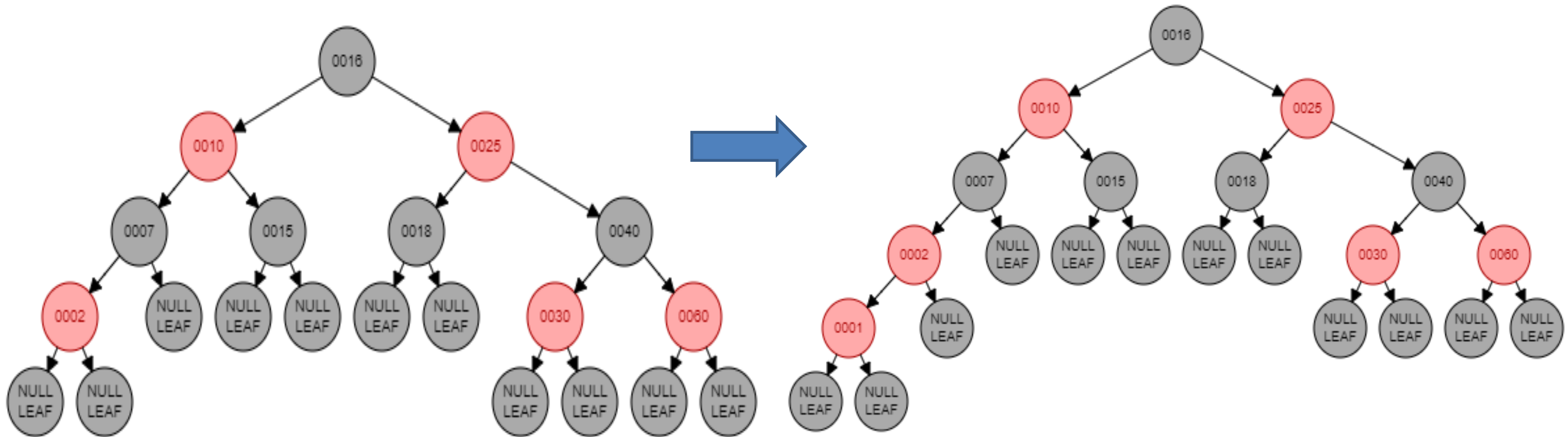
Before - NV



After - NV

Insertion -Example

11. Insert(root, 1)

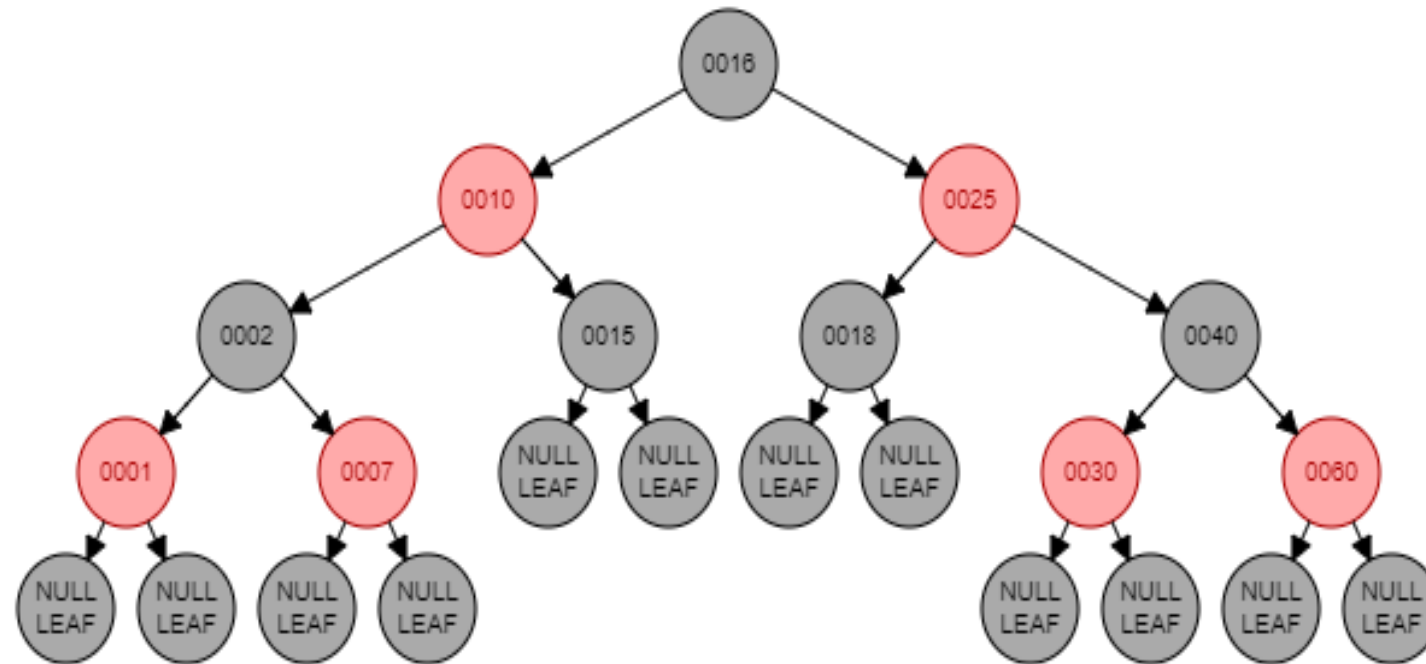


Before - NV

V

Insertion -Example

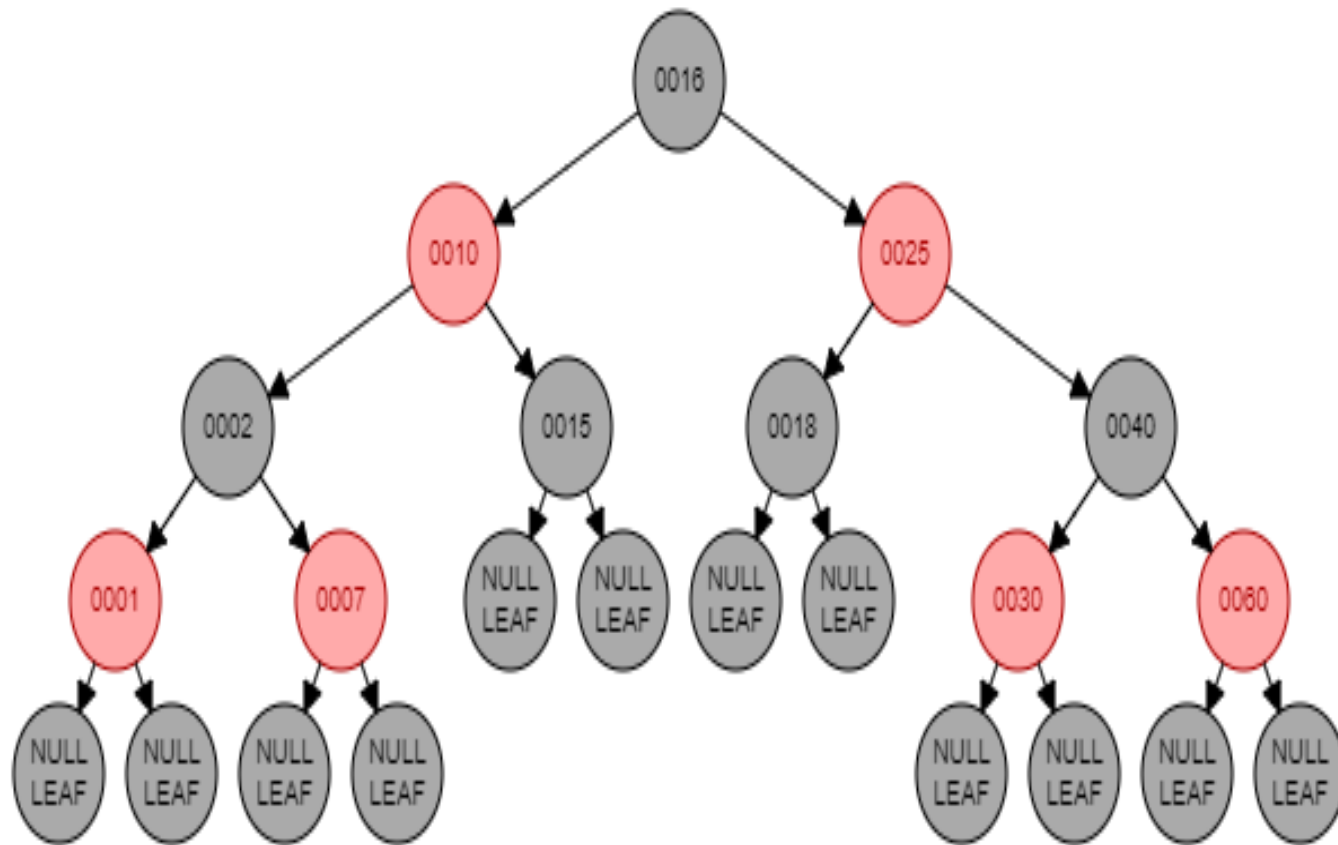
Insert(root, 1) Contd.



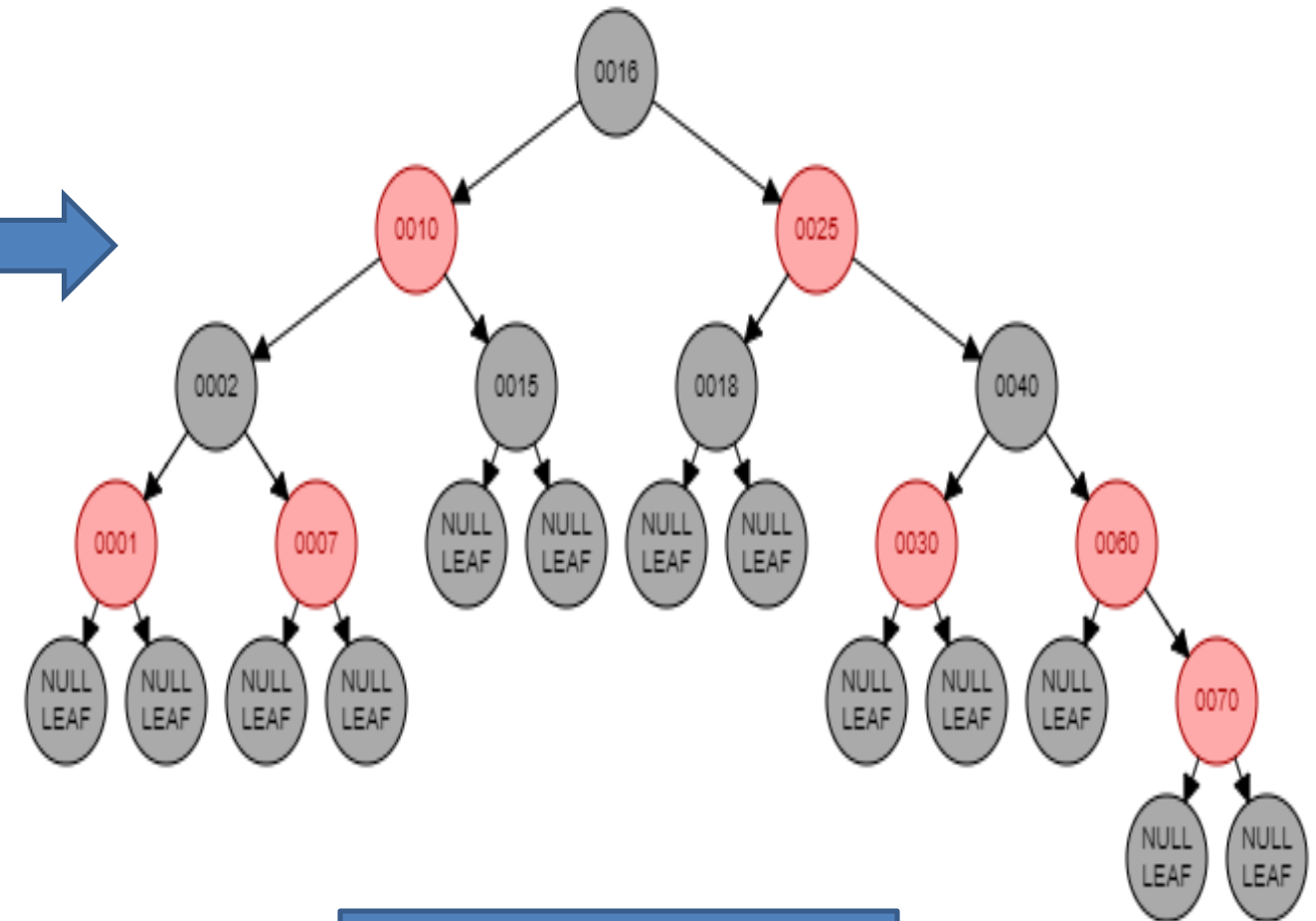
After - NV

Insertion -Example

12. Insert (root, 70)



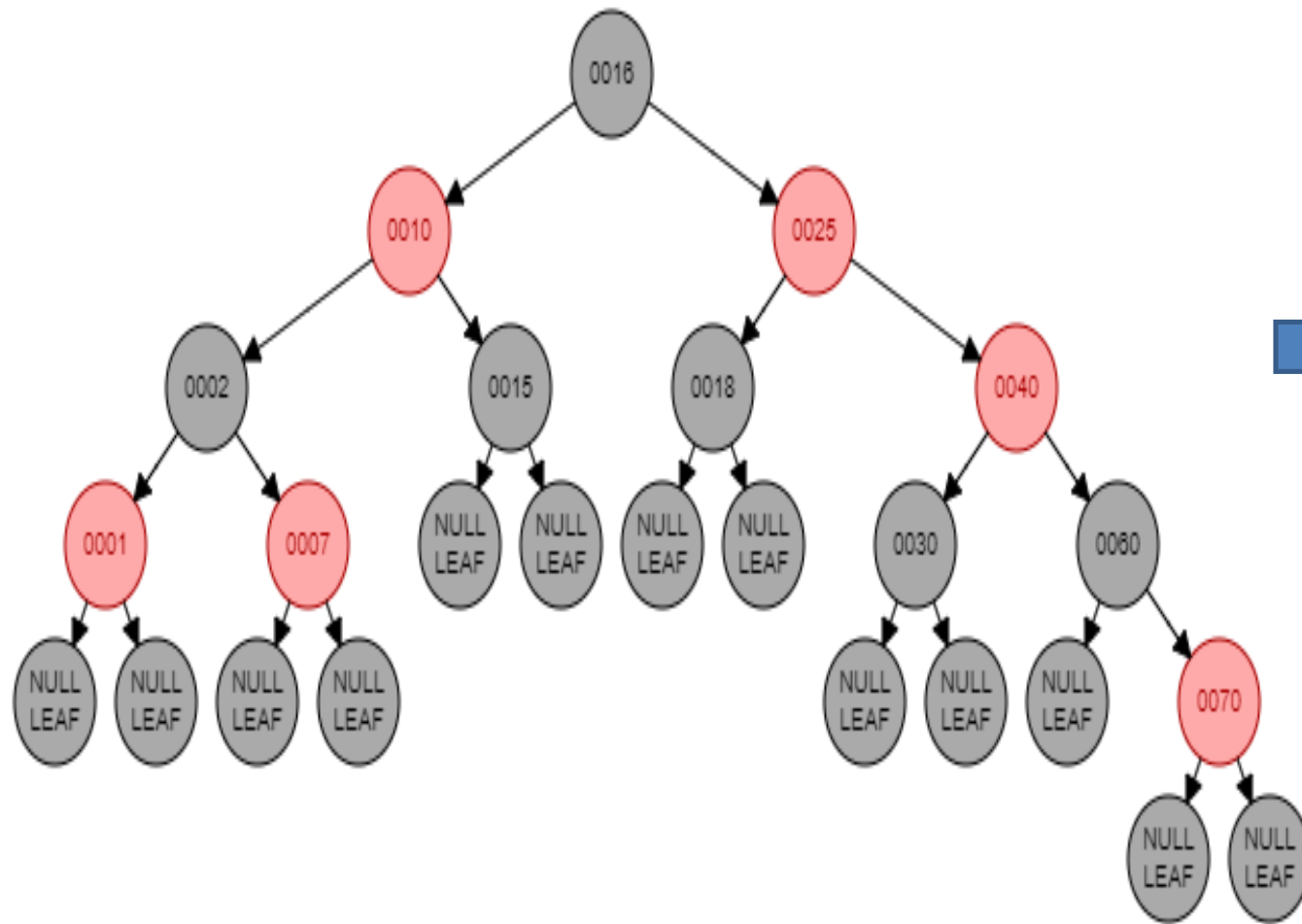
Before - NV



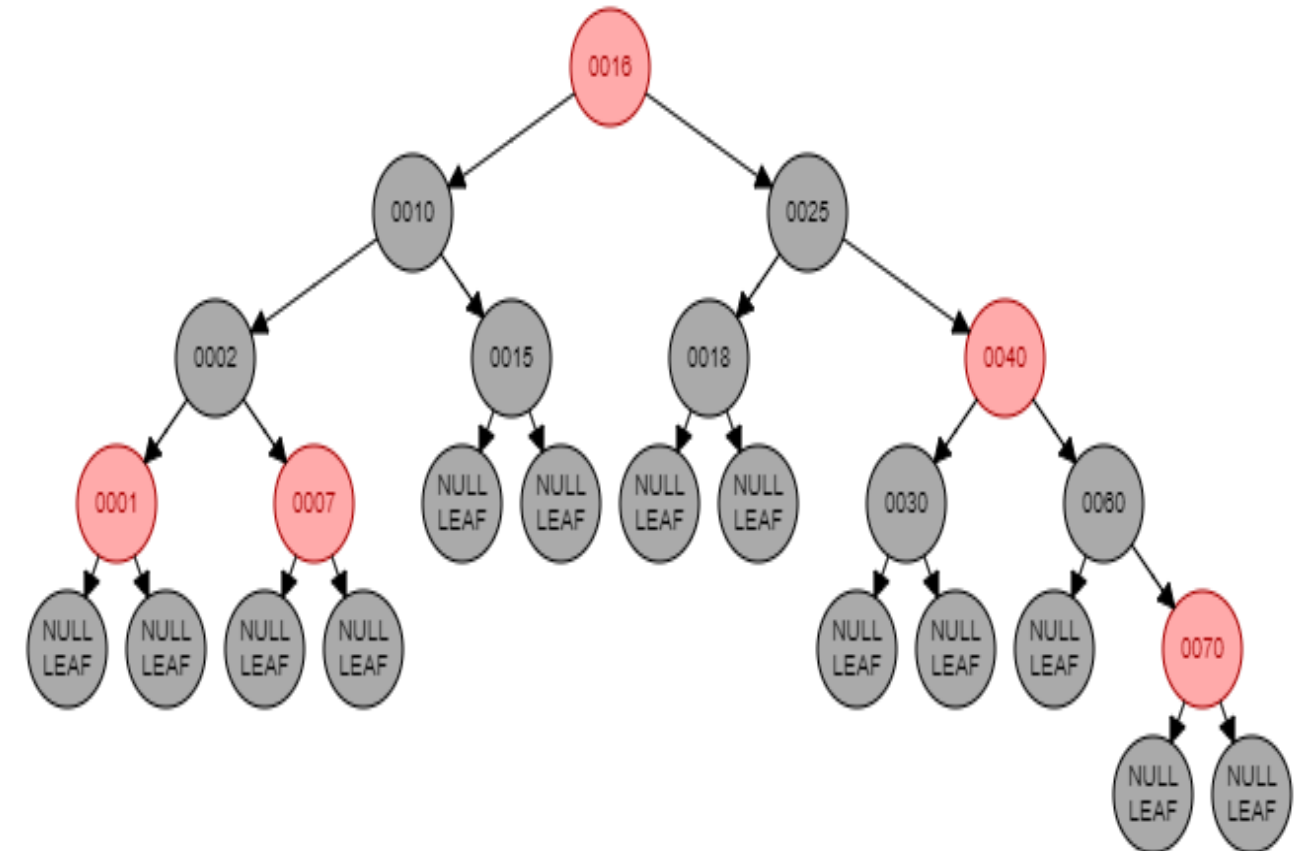
✓

Insertion -Example

Insert (root, 70) Contd.



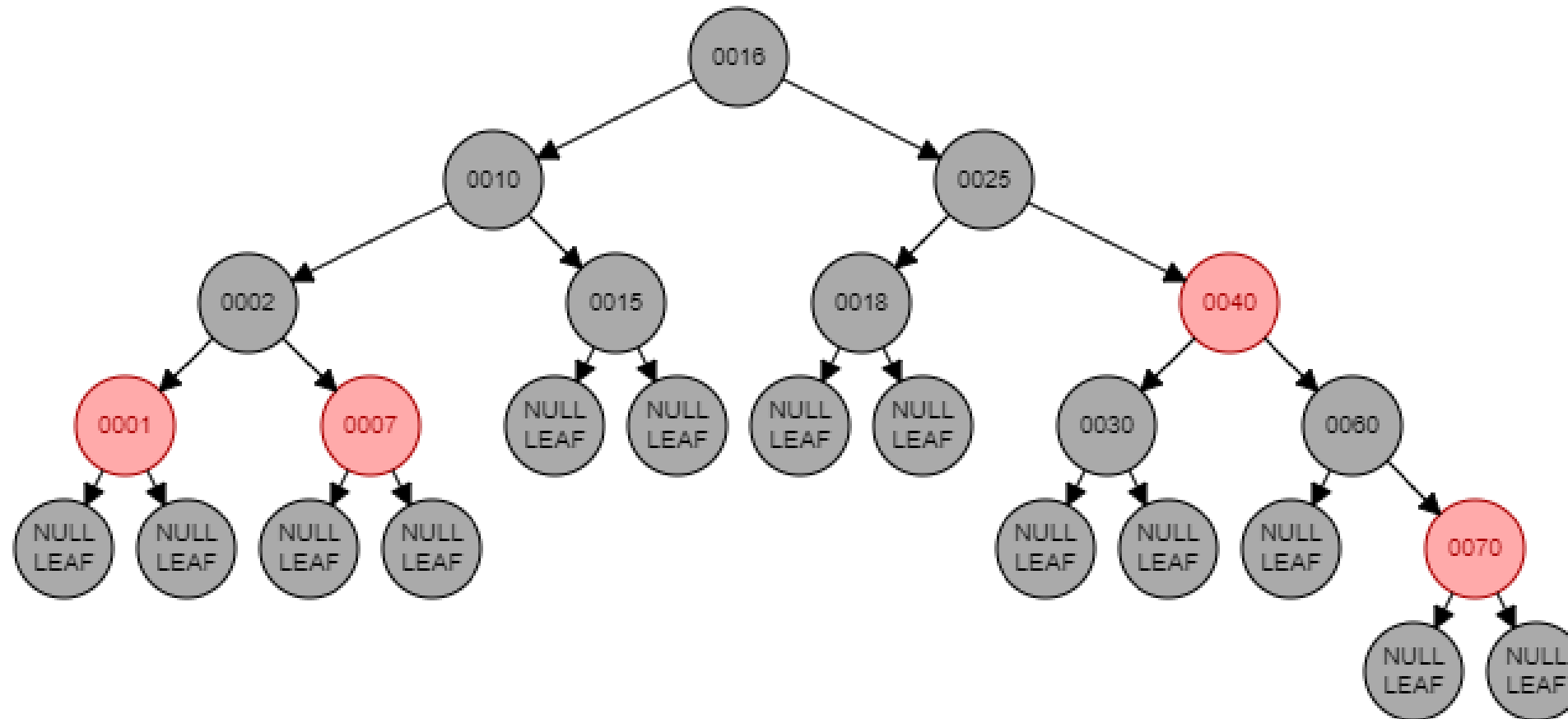
✓



✓

Insertion -Example

Insert (root, 70) Contd.



After - NV

Advantages

- RB Trees are useful when we need insertion or deletion relatively frequent.
- Maximum in 2 steps we can do balance.
- These are self balancing so these operations are guaranteed to be $O(\log(n))$.
- All the advantages of Binary search trees.

Applications

- Most of the self-balancing BST library functions like map and set in C++ use Red Black Tree.
- It is used in Databases.
- It is also used for searching words inside dictionaries, searching web.
- Red Black tree usually for fast element searches.
- In real world Red Black is one of the Application .They are common in the Linux kernel.
- For Example in a process scheduler or for keeping track of the virtual memory segment for a process.

➤ **Q :- Is it possible to have all black nodes in a RB Tree?**

A :- Yes it is possible.

a) A RB Tree with only one node .

b) By deleting nodes in RB Tree .

➤ **Q :- Why we have to add red node every time in RB tree instead of black?**

A :- If we insert red node , by using RB tree properties we can remove violations and make the tree balance.

But if we add the black node ,no violation rules is there in rb tree .So that black height will differ and tree does not balanced.

Recap

- What is Red Black tree and Why it is?
- Properties / Rules of Red Black tree.
- Distinguish between RB Tree and BST
- Insertion Fixing Violations
- Rotations
- Rotation Cases
- Insertion
- Advantages
- Applications
- Q & A
- Recap
- References




References

- <https://www.geeksforgeeks.org/red-black-tree-set-1-introduction-2/>
- <https://stepik.org/lesson/31445/step/5>
- <https://www.codesdope.com/course/data-structures-red-black-trees-insertion/>
- <https://www.cs.usfca.edu/~galles/visualization/RedBlack.html>

Thank You!



India: Bengaluru, Hyderabad | US: California

   | www.globaledgesoft.com

FAIRNESS • LEARNING • RESPONSIBILITY • INNOVATION • RESPECT