Department of
Computer Engineering

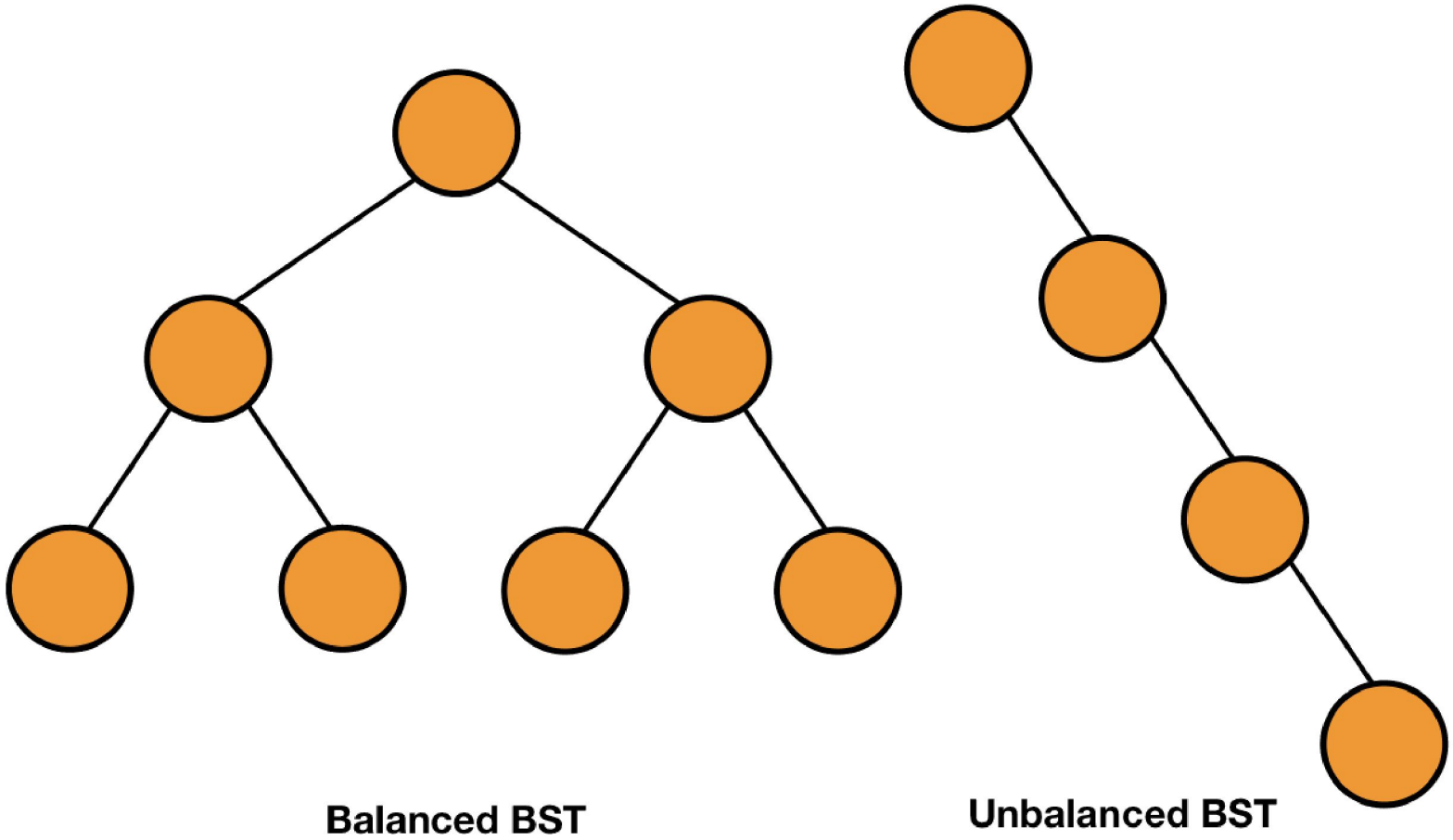
Data Structure & Algorithms

Red Black Trees

Balanced Binary Tree – Review

- A **balanced binary tree** is a binary tree in which the difference between the height of the left subtree and right subtree of each node is not more than 1.

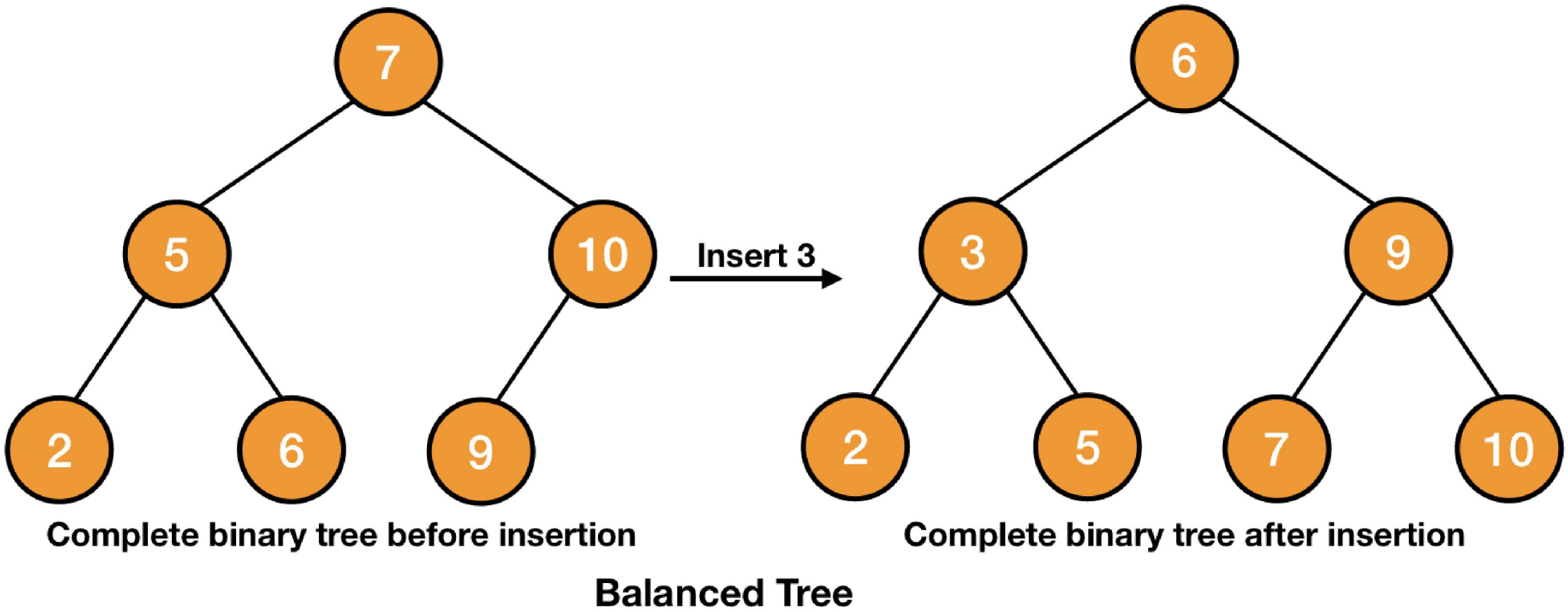
Balanced Binary Tree – Review



Balanced Binary Tree – Review

- **Self-Balancing Binary Search Trees** are *height-balanced* binary search trees that automatically keep height as small as possible when insertion and deletion operations are performed on tree. The height is typically maintained in order of Log n so that all operations take O(Log n) time in average.
- A **red-black** tree is a kind of self-balancing binary search tree.

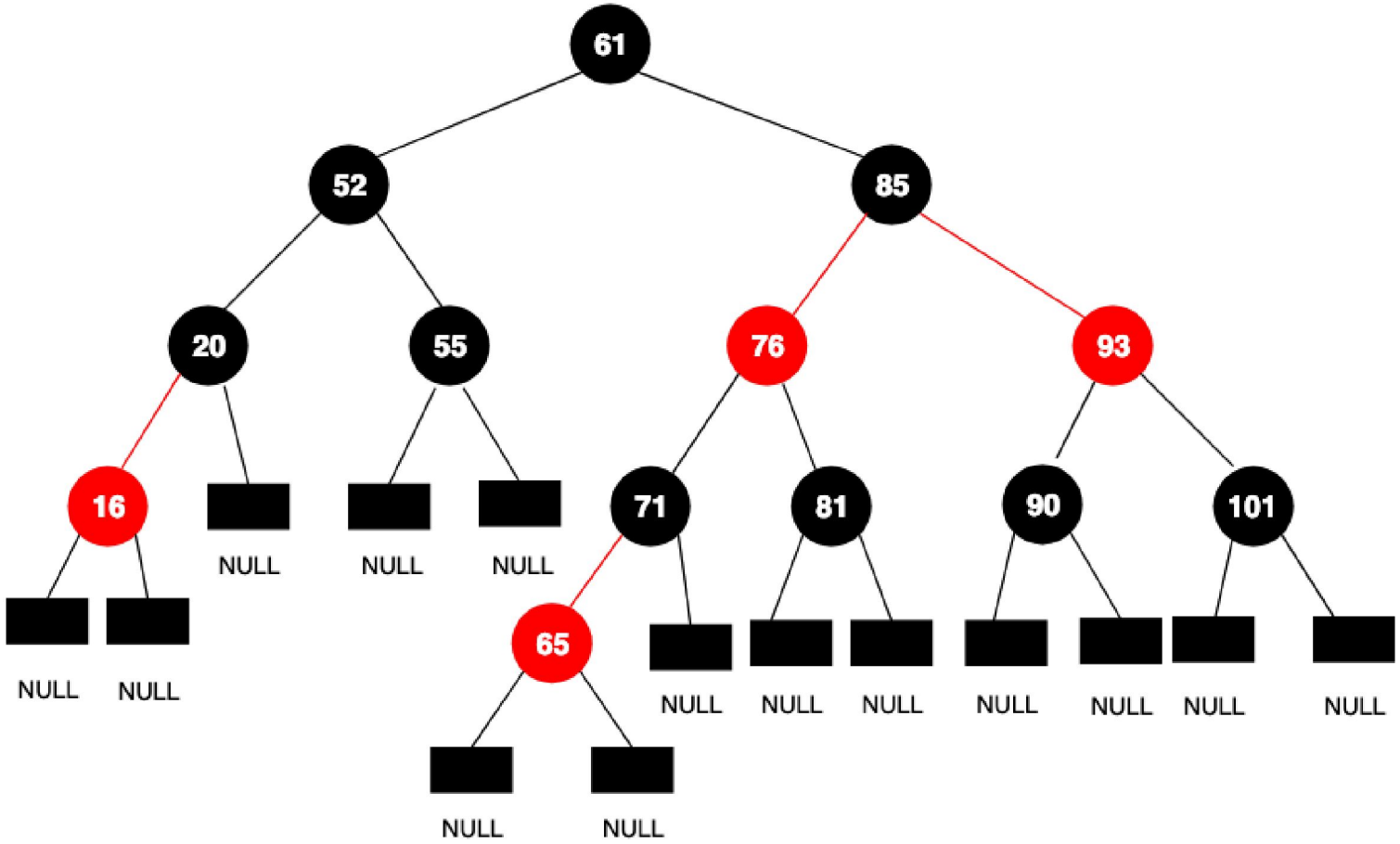
Balanced Binary Tree – Review



Properties of Red-Black Trees

1. Every node is colored either red or black.
2. Root of the tree is black.
3. All leaves are black.
4. Both children of a red node are black i.e., there can't be consecutive red nodes.
5. All the simple paths from a node to descendant leaves contain the same number of black nodes.

Properties of Red-Black Trees

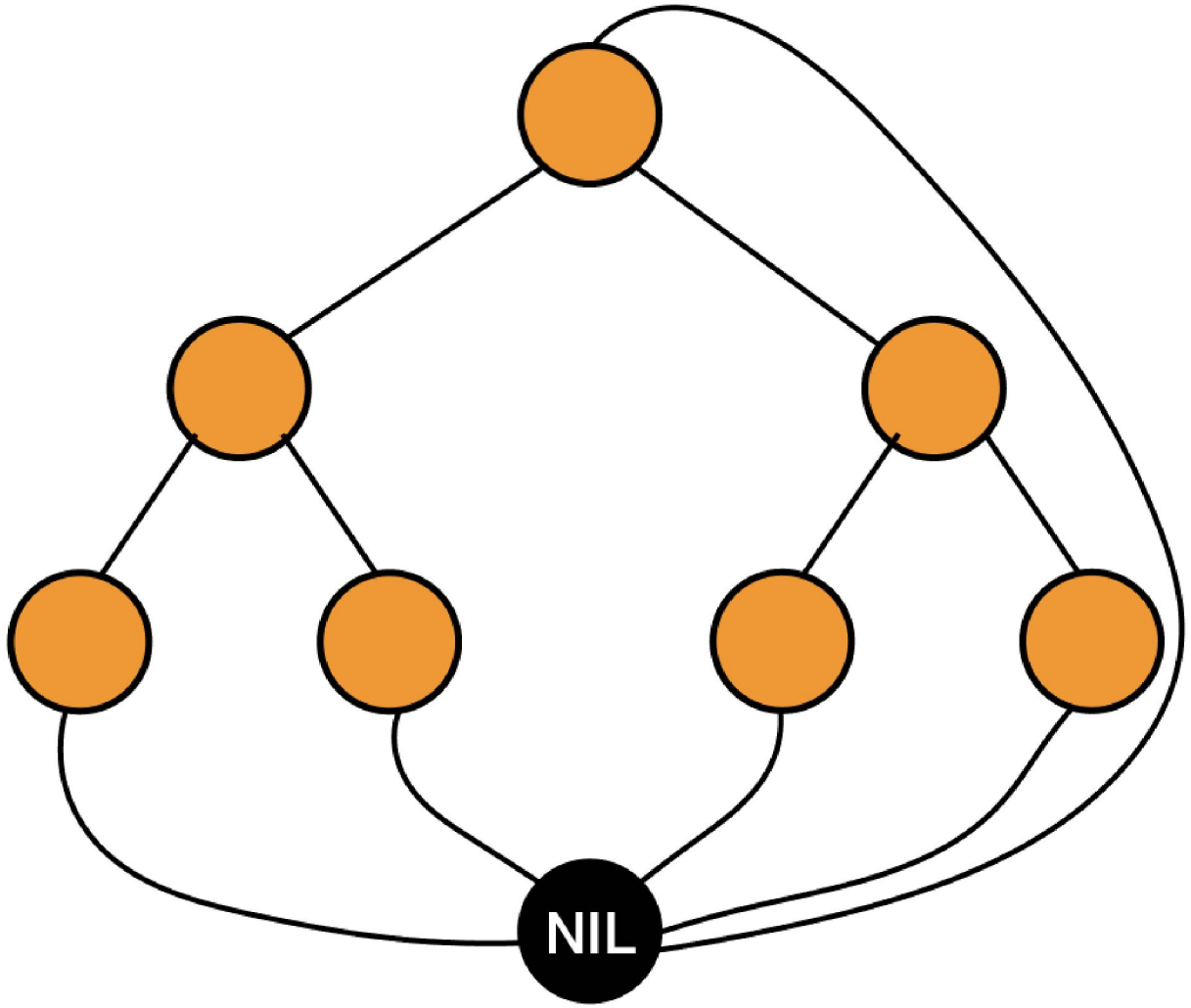


Properties of Red-Black Trees

- Since all the leaves are black, we have used blank nodes or NULL/NIL for them as shown in the above picture. We can also use only one node to represent all NIL with black color and any arbitrary values for the parent, data, left, right, etc. as shown below.

Properties of Red-Black Trees

So instead of NULL, we are using an ordinary node to represent (NIL) to represent it. This technique will save a lot of space for us.

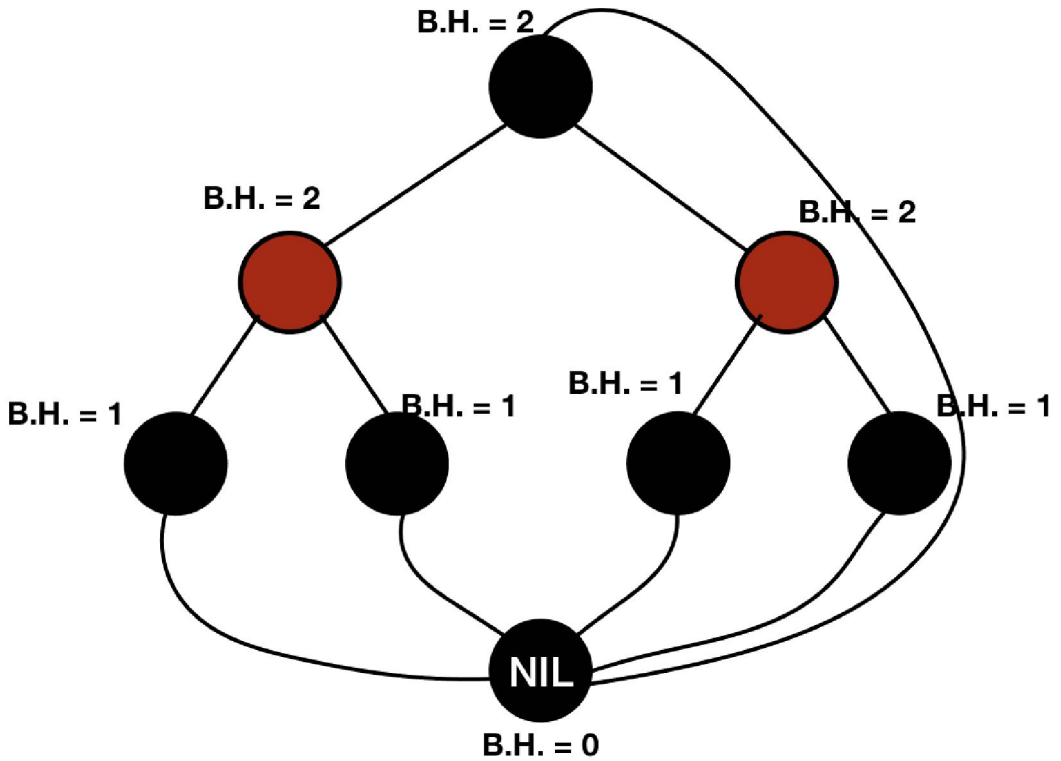


Black Height of Red-Black Tree

- Black height is an important term used with red-black trees. It is defined as number of black nodes on any simple path from a node x to a leaf (not including it). Black height of any node x is represented by $bh(x)$.
- According to property 5, the number of black nodes from a node to any leaf is the same. Thus, the black height of any node counted on any path to any leaf will be the same.
- Look at the picture given below with black height of nodes

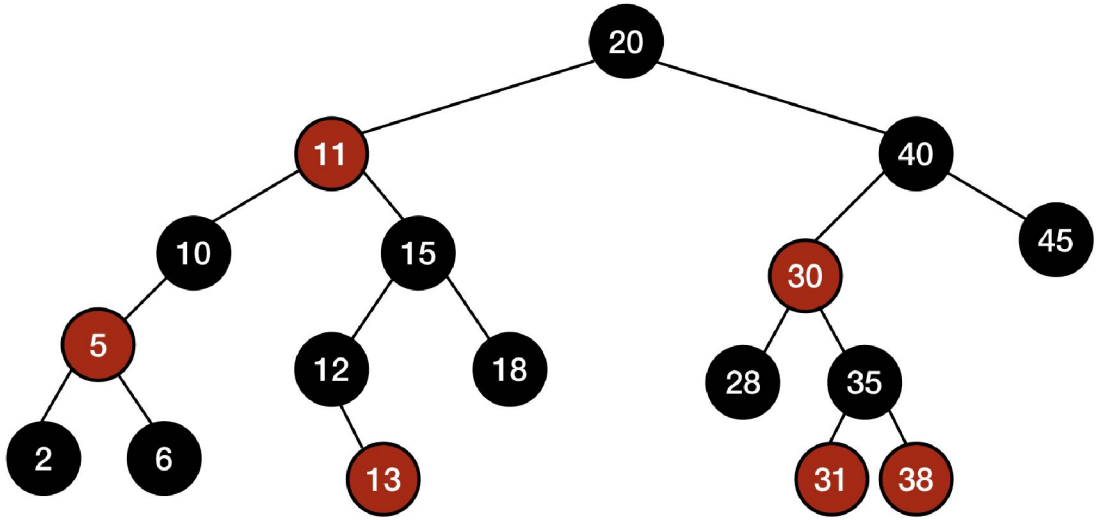
Black Height of Red-Black Tree – Example

- Black height of the leaf (NIL) is 0 because we exclude the node for which we are counting the black height. Root has a black height of 2 because there are 2 black nodes (excluding the root itself) on a path from the root to leaf.



Black Height of Red-Black Tree – Example

- We have omitted the leaves while representing this tree and we are going to follow the same pattern in this entire chapter. You should keep in mind that there is a NIL node representing the leaves in each example.



Proof of height of red-black tree is O(lg(n))

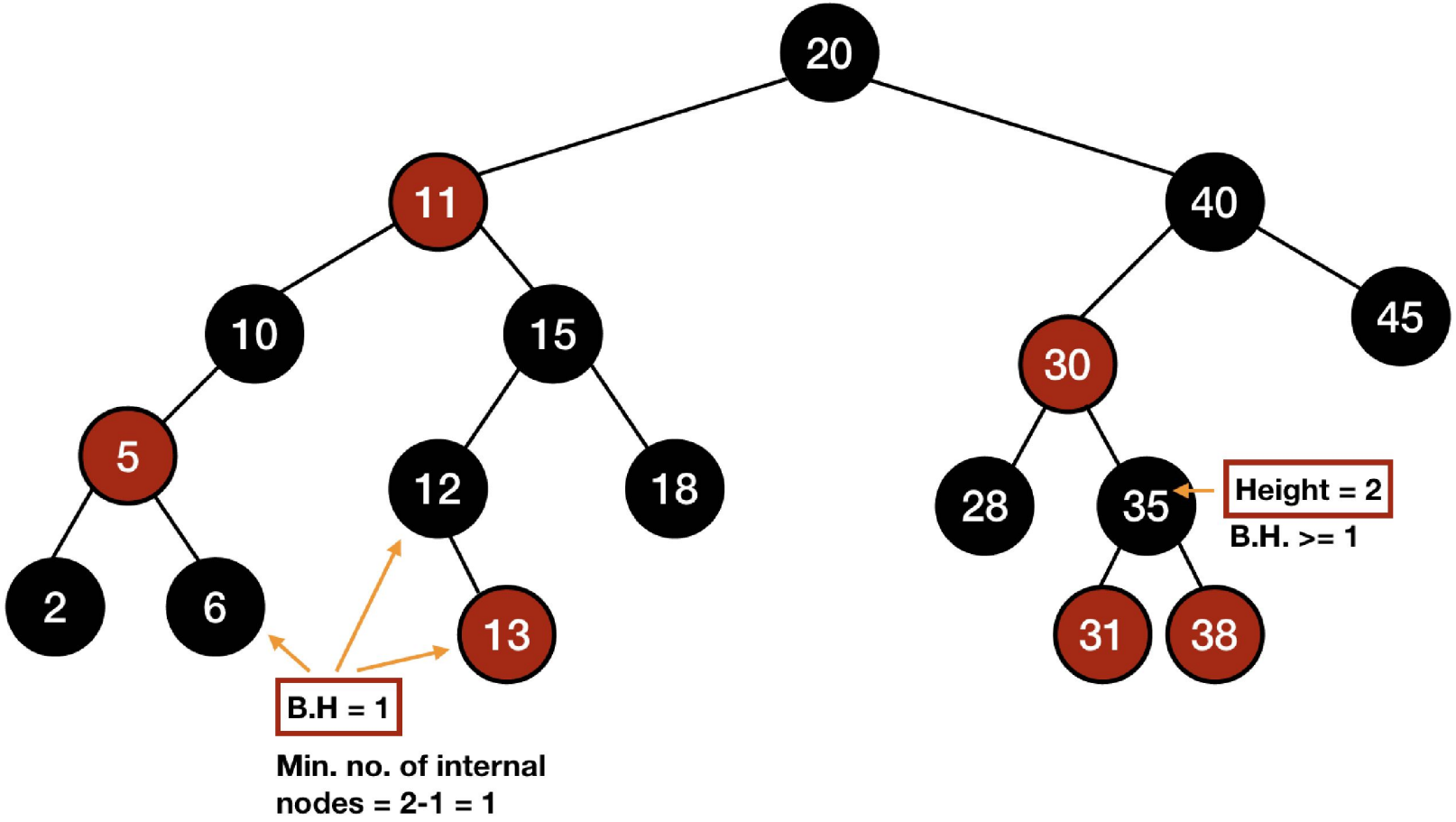
- A binary search tree following the above 5 properties is a **red-black** tree. We also mentioned that basic operation of a binary search tree such as search can be done in $O(\log n)$ worst-case time on a red-black tree. To prove this, we will first prove that **a binary search tree following the above properties (thus, a red-black tree) with n internal nodes can have a maximum height of** $2\log(n+1)$

Proof of height of red-black tree is O(lg(n))

In order to do so, we need to prove the following statements first:

1. A subtree rooted at any node x has **at least** $2^{bh(x)} - 1$ internal nodes.
2. Any node x with height $h(x)$ has $bh(x) \geq \frac{h(x)}{2}$

Proof of height of red-black tree is O(lg(n))

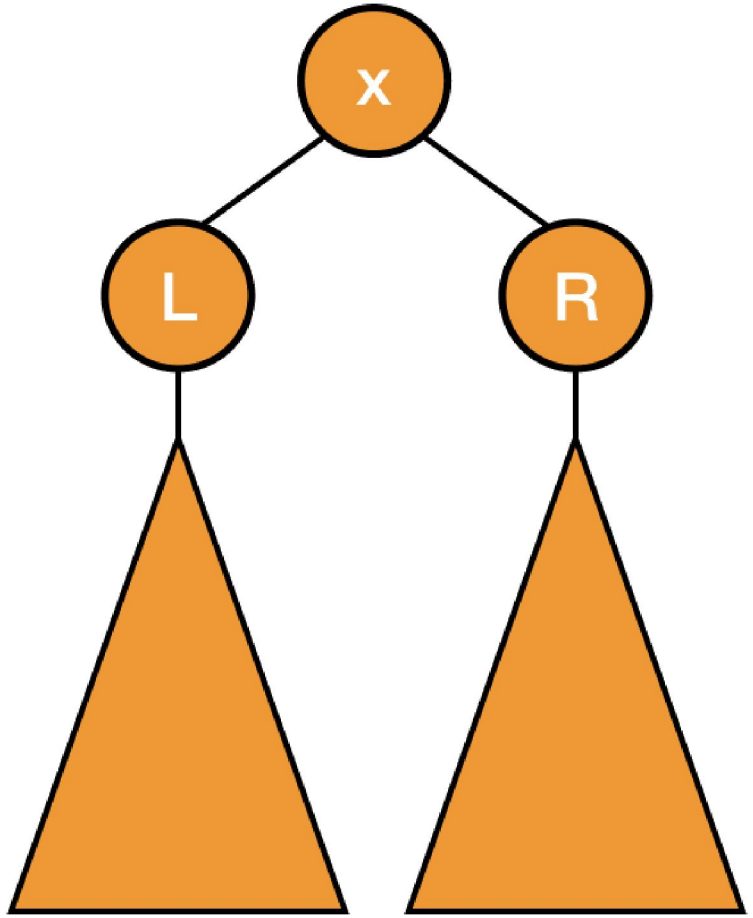


Proof of height of red-black tree is O(lg(n))

We are going to prove the first statement by the method of induction. The base case will be when x is 0 i.e., x is a leaf. According to the statement, number of internal nodes are $2^0 - 1 = 0$. Since x is a leaf, this statement is true in the base case.

Now, consider a node x with two children l and r.

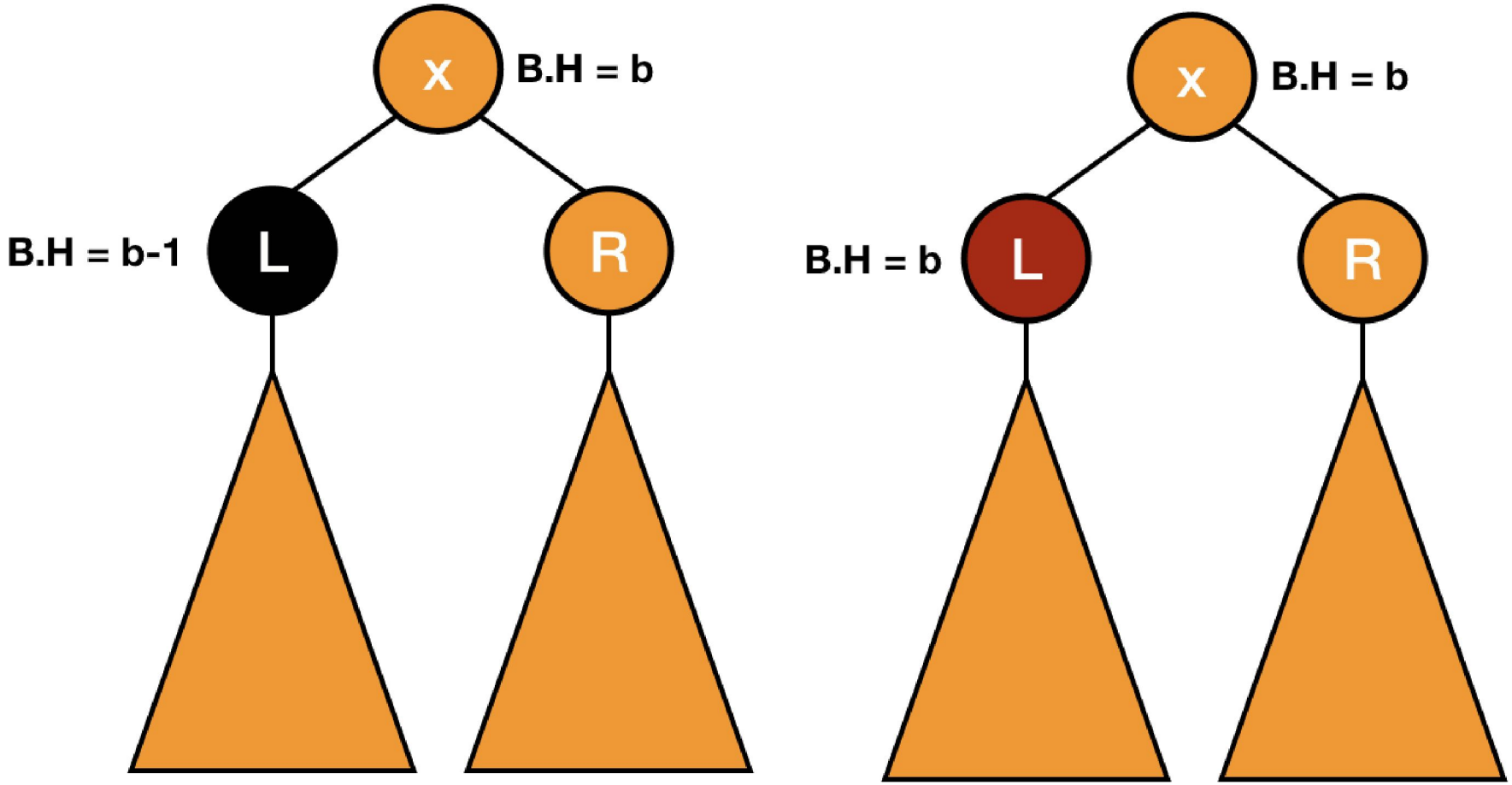
Proof of height of red-black tree is O(lg(n))



Proof of height of red-black tree is O(lg(n))

- Let $bh(x) = b$. Now if the color of the child is red, then its black height will also be b. However, if the color of the child is black, then its black height will be $b - 1$.
- According to the inductive hypothesis, child must have at least $2^{b-1} - 1 = 2^{bh(x)} - 1$ internal nodes.

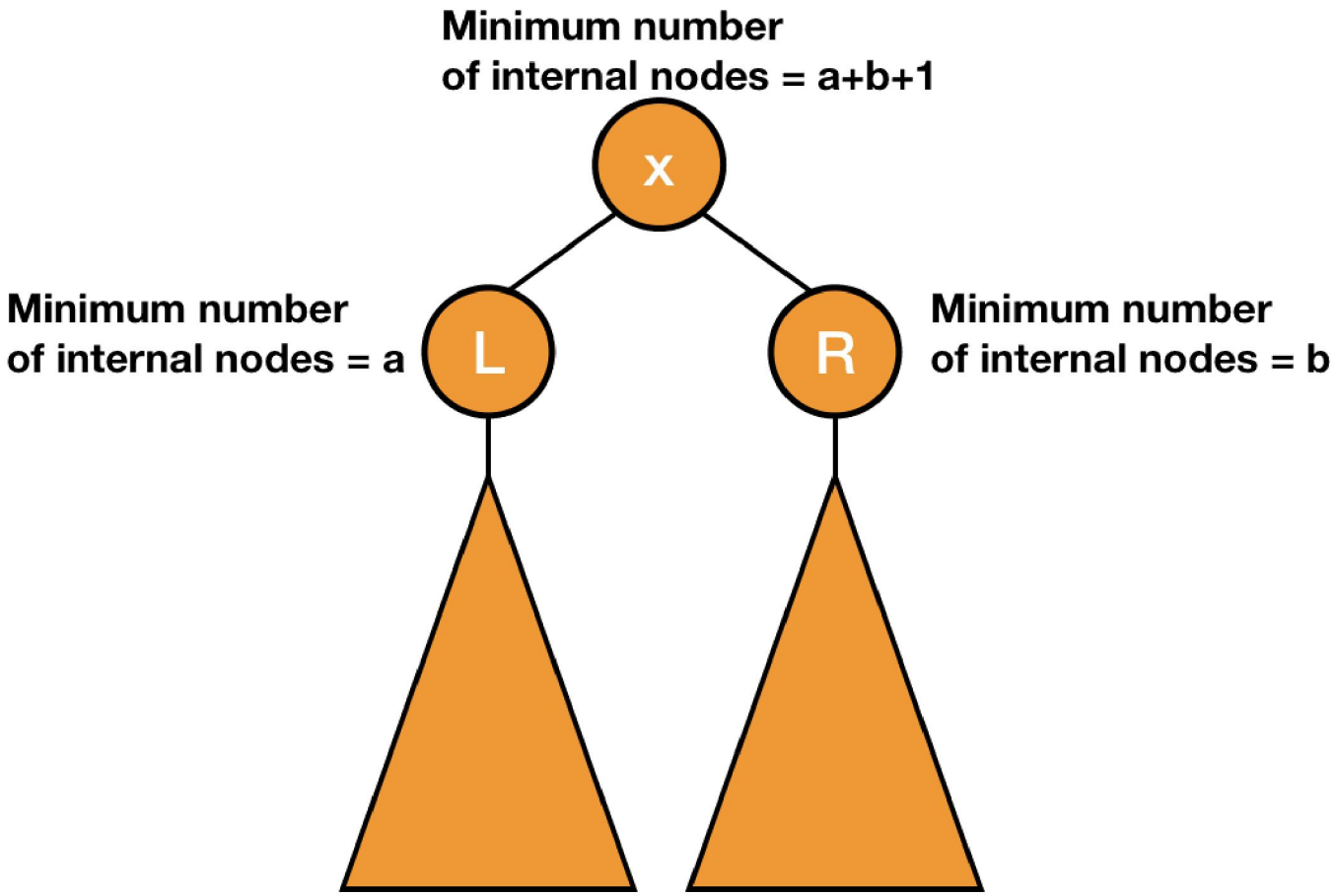
Proof of height of red-black tree is O(lg(n))



Proof of height of red-black tree is O(lg(n))

- We have assumed that the inductive hypothesis is true for the child, now we need to show that it is also true for the parent i.e., node x and hence completing the proof.
- The node x must have at least $1 +$ minimum number of internal nodes that can be present on the right child $+$ minimum number of internal nodes that can be present on the left child.

Proof of height of red-black tree is O(lg(n))



Proof of height of red-black tree is O(lg(n))

$$2^{bh(l)-1} + 2^{bh(r)-1} + 1$$

- Internal nodes of $x \geq 2^{bh(l)-1} + 2^{bh(r)-1} + 1$
- or, Internal nodes of $x \geq 2 * (2^{bh(x)-1}) - 1$
- or, Internal nodes of $x \geq 2^{bh(x)} - 1$
- We assumed it to be true for the child and it is also true for the node x, so the statement is proved.

Proof of height of red-black tree is O(lg(n))

- Let's prove the second statement.
- Since the leaves are black and there can't be two consecutive red nodes, so the number of red nodes can't exceed the number of black nodes on any simple path from a node to a leaf. Therefore, we can also say that at least half of the nodes on any simple path from a root to a leaf, not including the node, must be black.
- It means that $bh(x) \geq \frac{h(x)}{2}$
- Thus, the second statement is also true.

Proof of height of red-black tree is O(lg(n))

- Acoording to the second statement, $bh(\text{root}) \geq \frac{h(\text{root})}{2}$ where h is the height of the tree and n is number of internal nodes.
- Using statement 1, $\text{n} \geq 2^{bh(root)} - 1$
- Or, $\text{n} \geq \frac{h}{2} - 1 (bh(root) \geq \frac{h}{2})$
- Or, $\text{n} + 1 \geq \frac{h}{2}$
- Taking log on both sides,
- $\log n + 1 \geq \frac{h}{2}$ or, $h \leq 2 \log n + 1$

Proof of height of red-black tree is O(lg(n))

- Thus, the height of a red-black tree is $O(\log n)$.
- we know that basic operations require $O(h)$ time in a binary search tree and we have proved that the height h is $O(\log n)$. Thus, all those operations can be done in $O(\log n)$ time in a **red-black** tree.
- We need to maintain the properties of a red-black tree while inserting or deleting any node.