Data Structures

AVL Insertion

Mostafa S. Ibrahim
Teaching, Training and Coaching since more than a decade!

Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / Msc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)

Insertion

- Insertion follows a very similar process to what you have learned already
- But, we need to fix any corruption (i.e. |BF| > 1) immediately
- To do so, at the end of our insertion function, we will need to both update the height, and call the balance function, which will check if |BF|>1 or not
- We also need to update the left and right subtrees, as they might have been changed
- We will follow the code from the last homework (rewritten BST with extra node struct)

Insertion: before

- This is the original insertion code (before making changes)
- Observe: it assumes tree nodes are never rebalanced (changed)
- We need to balance the nodes, and ensure flexibility in our code

```
def insert(self, val):
    def process(current, val):
        if val < current.val:
            if not current.left:
                current.left = Node(val)
            else:
                process(current.left, val)
        elif val > current.val:
            if not current.right:
                current.right = Node(val)
            else:
                process(current.right, val)
        # Elise - already exists
```

Insertion: after

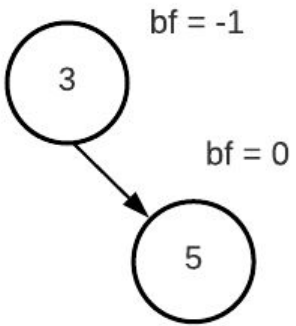
- 3 critical code changes:
- 1) current.left = process(current.left, val)
  - To update left if its tree rotated
  - The same for the right node
- 2) update_height and balance the node in the end
- 3) update the root as it may be changed

```
def insert(self, val):
    def process(current, val):
        if val < current.val:
            if not current.left:
                current.left = Node(val)
            else:
                # ** change left. update left as it might be balanced
                current.left = process(current.left, val)
        elif val > current.val:
            if not current.right:
                current.right = Node(val)
            else:
                current.right = process(current.right, val)
        # Else - already exists
        # ** update/balance
        current.update_height()
        return self.balance(current)

    if not isinstance(val, list):
        val = [val]
    for item in val:
        # ** update the root
        self.root = process(self.root, item)
```

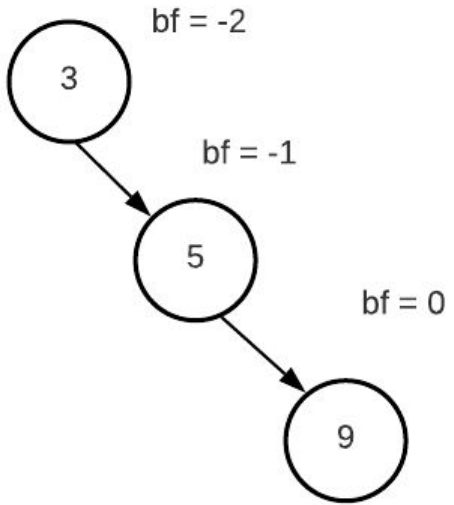
Insert: 3, 5, 9, 1, 0, 2, 6, 10, 7, 4, 8

- Let’s insert the first 2 values
- No problems so far



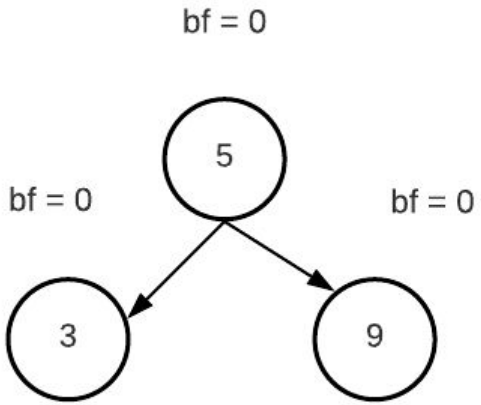
Insert: 3, 5, 9, 1, 0, 2, 6, 10, 7, 4, 8

- With 9, node(3) is unbalanced
- Right-Right case
- Do a left rotation at 3



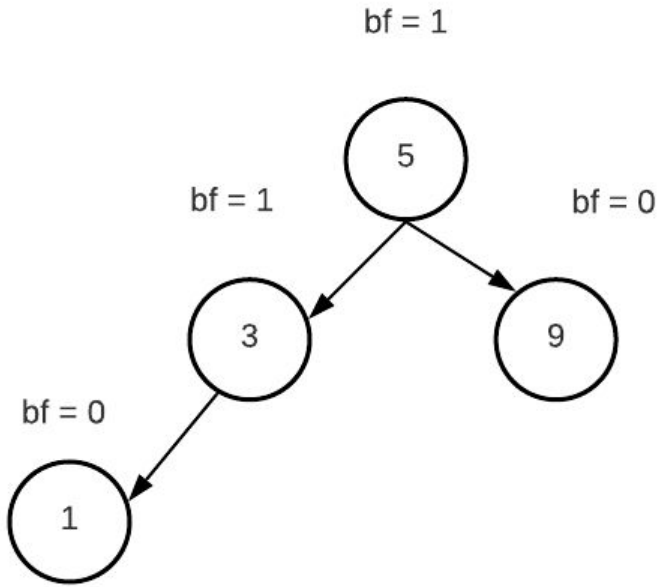
Insert: 3, 5, 9, 1, 0, 2, 6, 10, 7, 4, 8

- It's fixed now!



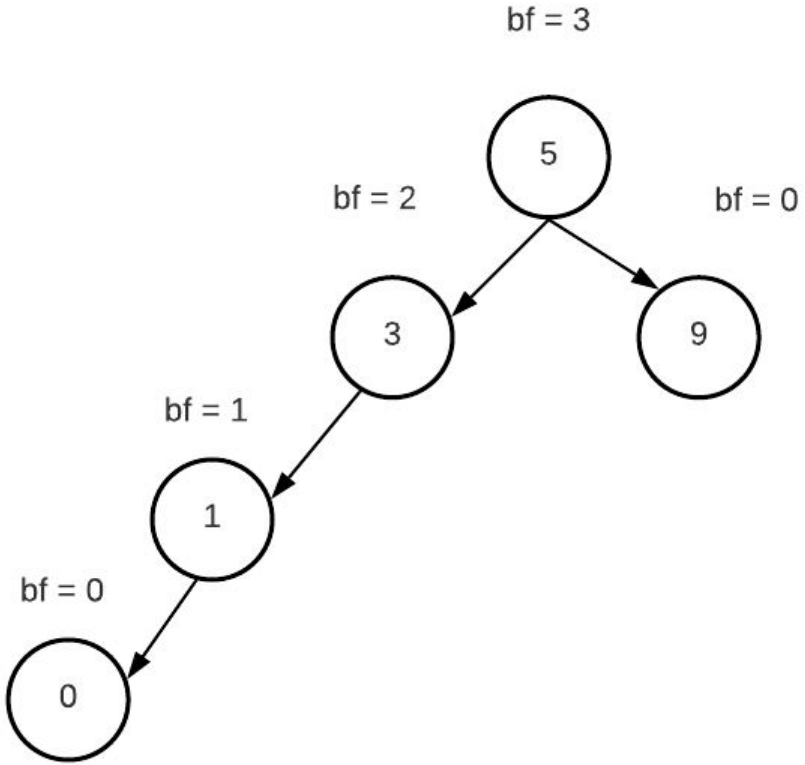
Insert: 3, 5, 9, 1, 0, 2, 6, 10, 7, 4, 8

- We're still balanced after inserting 1



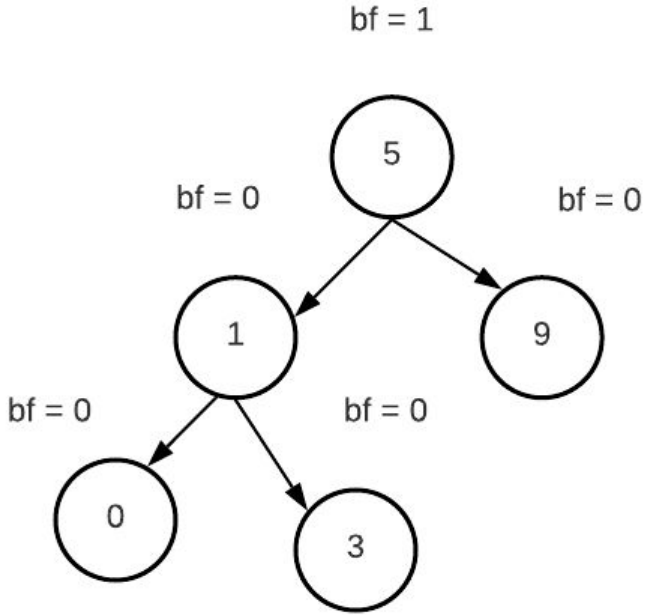
Insert: 3, 5, 9, 1, 0, 2, 6, 10, 7, 4, 8

- With 0, node(3) is unbalanced
- Left-Left case
- Do a right rotation at 3



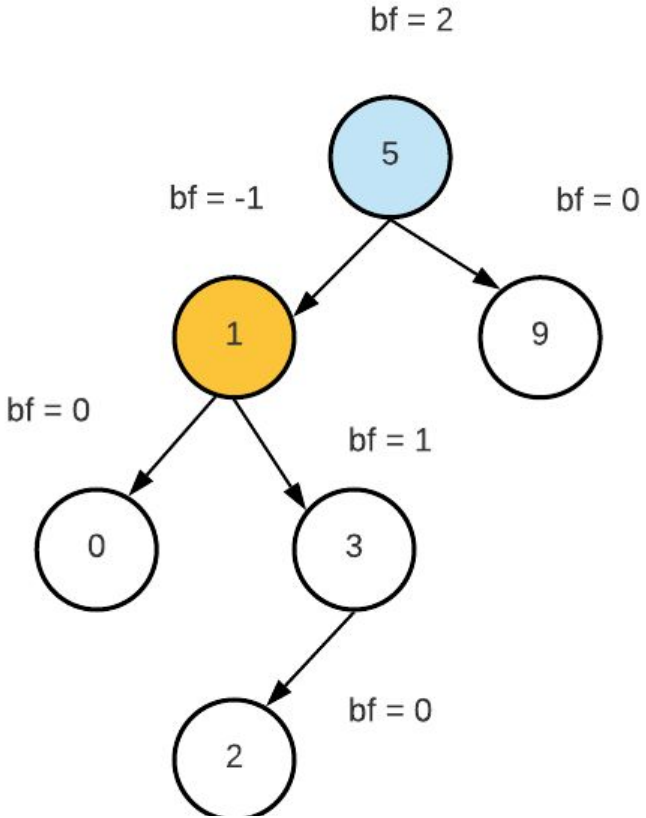
Insert: 3, 5, 9, 1, 0, 2, 6, 10, 7, 4, 8

- It's fixed now!



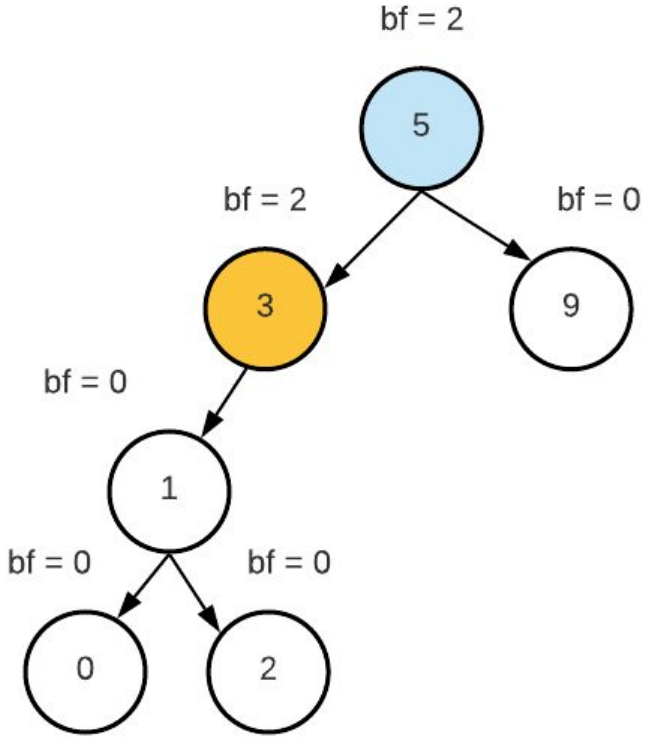
Insert: 3, 5, 9, 1, 0, 2, 6, 10, 7, 4, 8

- With 2, node(5) is unbalanced
- Left-Right case
  - Bf: 2 -1
- First, carry out left_rotation(1)
  - This pushes 1 down and 3 up
  - A=0, B=2, C=Null
    - So B's parent will change from 3 to 1
- **Then** perform right_rotation(5)



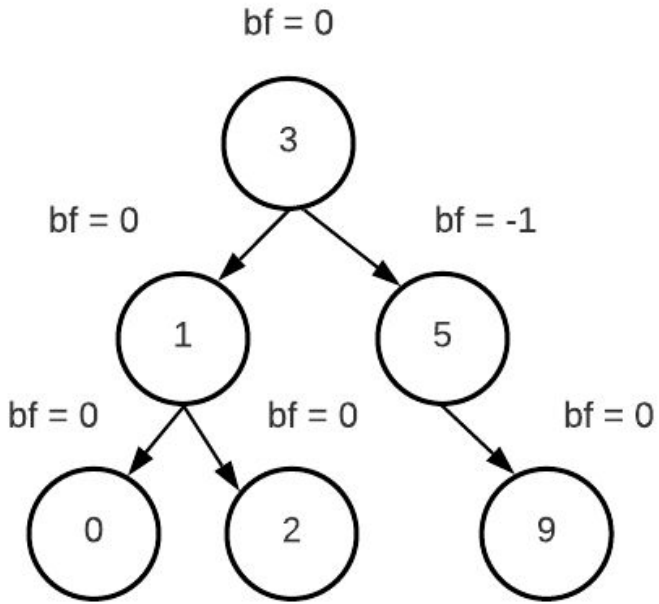
Insert: 3, 5, 9, 1, 0, 2, 6, 10, 7, 4, 8

- 3 is now a left-left case
  - Observe: bf(3) = 2
  - Don’t let that confuse you
- What **remains** is to perform right_rotation(5



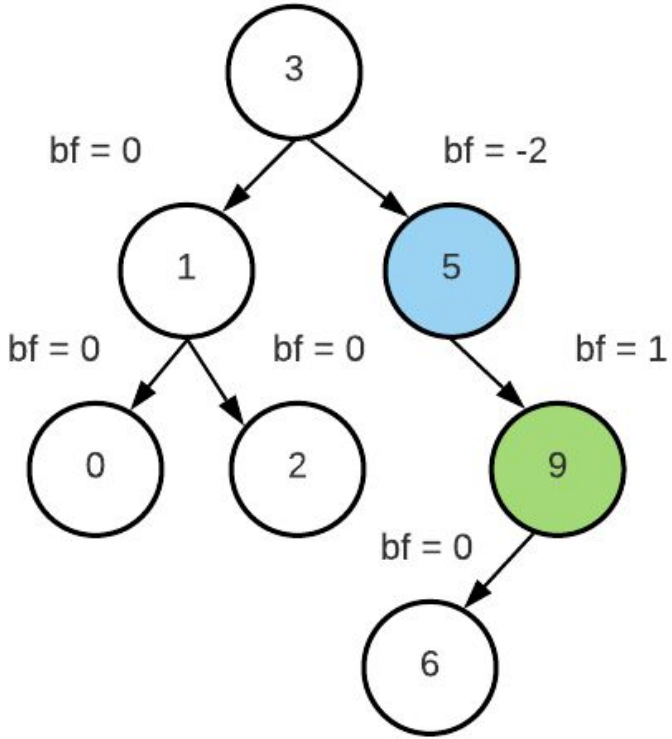
Insert: 3, 5, 9, 1, 0, 2, 6, 10, 7, 4, 8

- It's fixed now



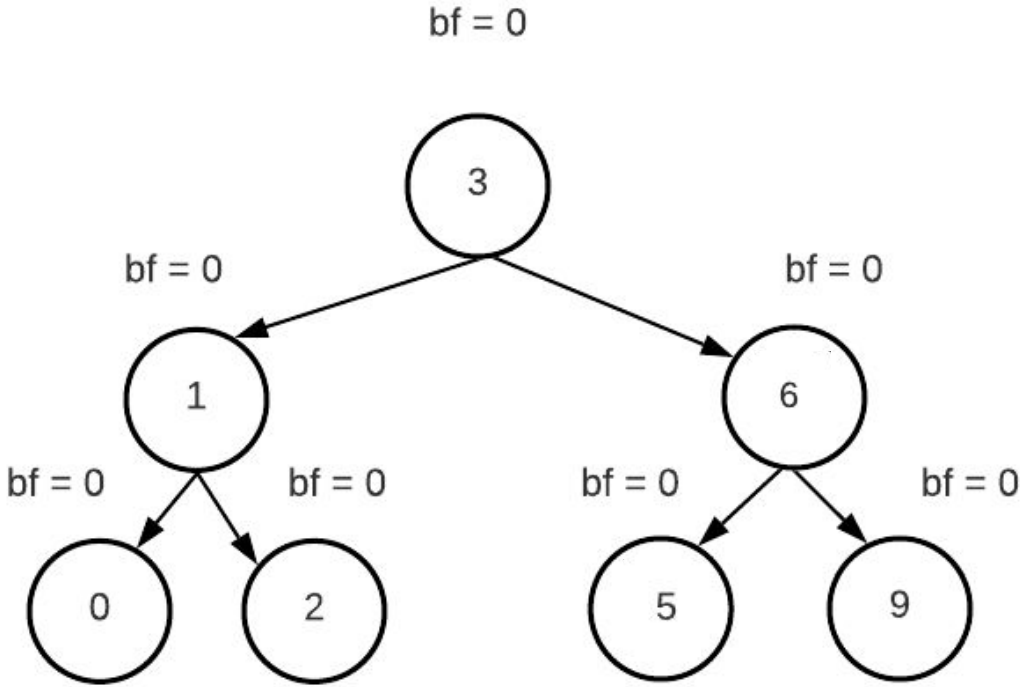
Insert: 3, 5, 9, 1, 0, 2, 6, 10, 7, 4, 8

- After inserting 6, we have a right-left case
  - Bf: -2, 1
- Perform right-rotation(9) to convert to right-right
- Then carry out left-rotation(5)
- As there are no children (A/B/C), this is easier to visualize



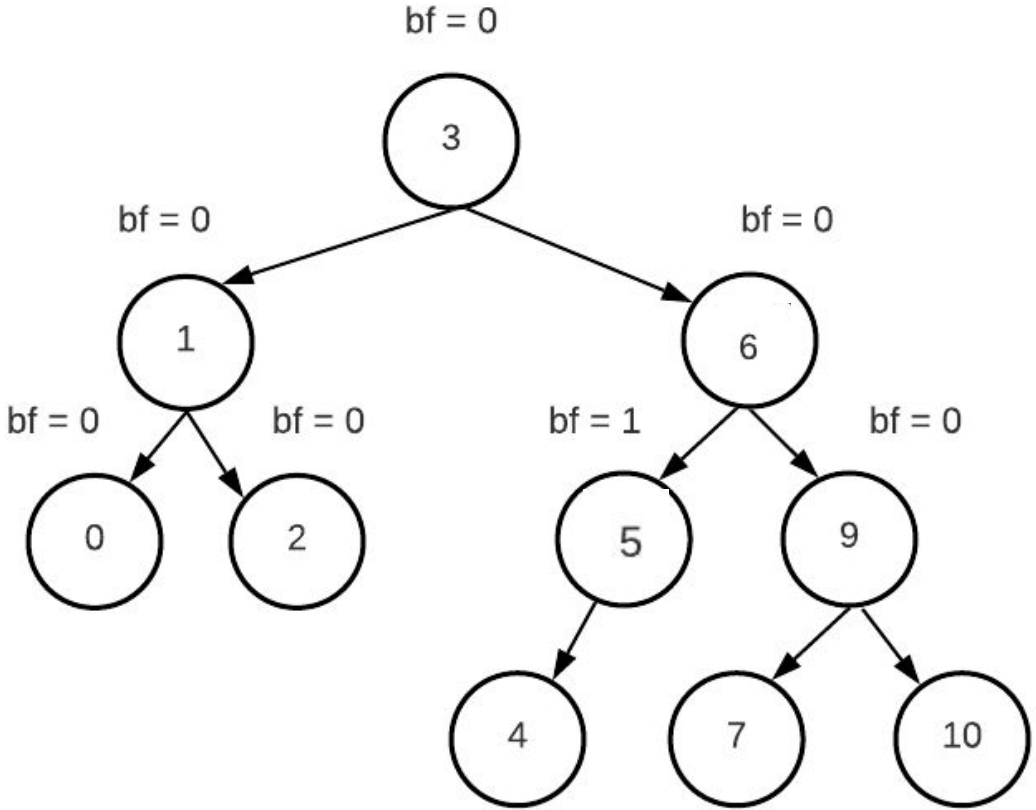
Insert: 3, 5, 9, 1, 0, 2, 6, 10, 7, 4, 8

- It's fixed now



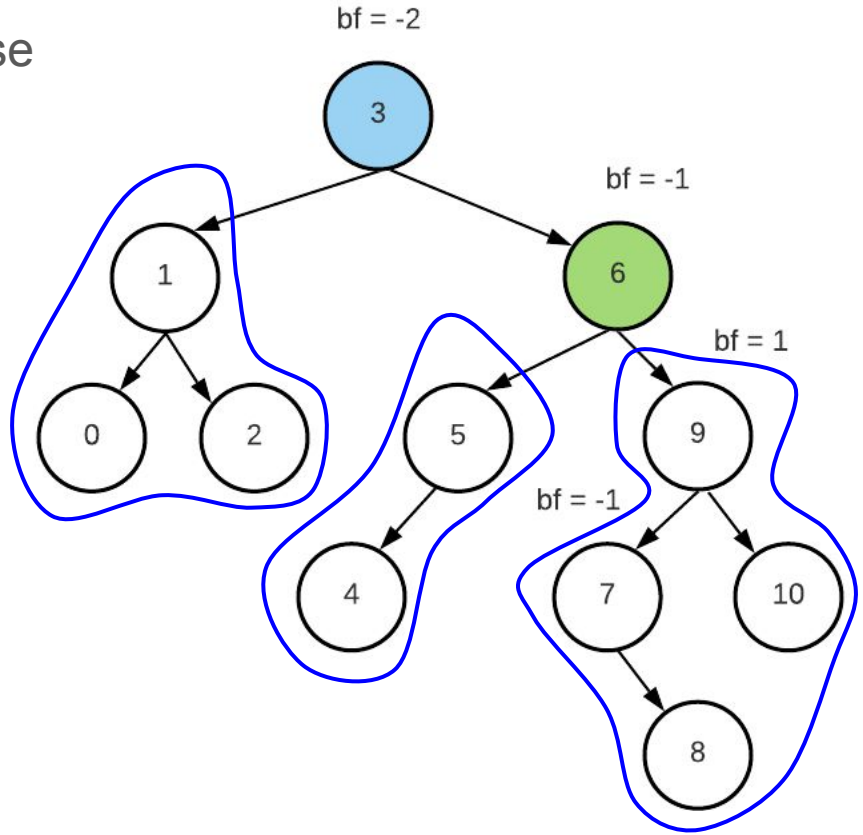
Insert: 3, 5, 9, 1, 0, 2, 6, 10, 7, 4, 8

- The next three values (10, 7, 4) slot into place without incident

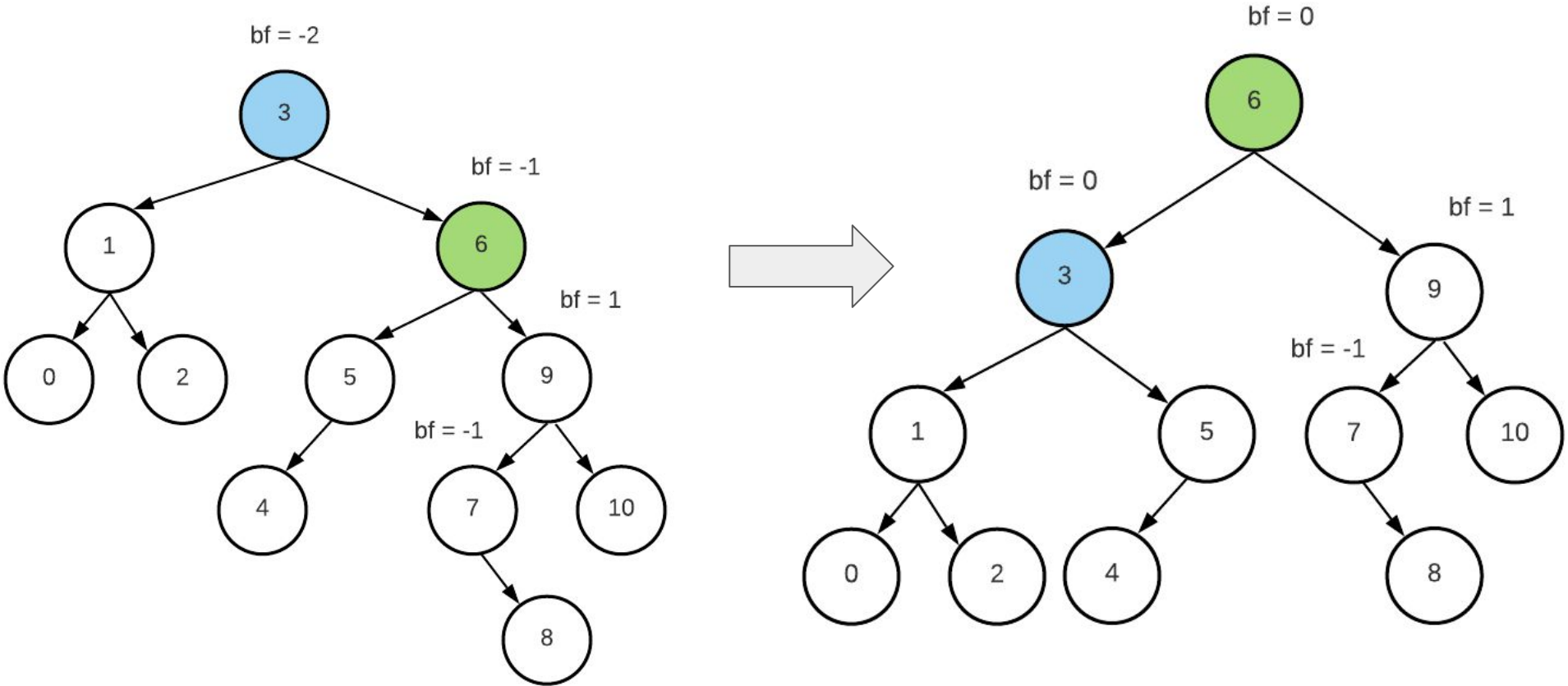


Insert: 3, 5, 9, 1, 0, 2, 6, 10, 7, 4, 8

- After inserting 8, we have a right-right case
  - Bf: -2, -1
- Left-rotation(3)
  - Pushes both 3 down and 6 up
  - A = subtree(1)
  - B = subtree(5)
    - It will change from left of 6 to right of 3
  - C = subtree(9)



Insert: 3, 5, 9, 1, 0, 2, 6, 10, 7, 4, 8



"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."