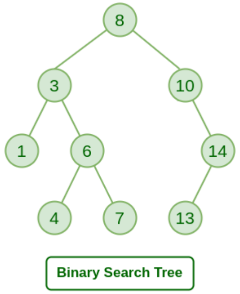
TODO BEFORE

To read Before #ToBeReady

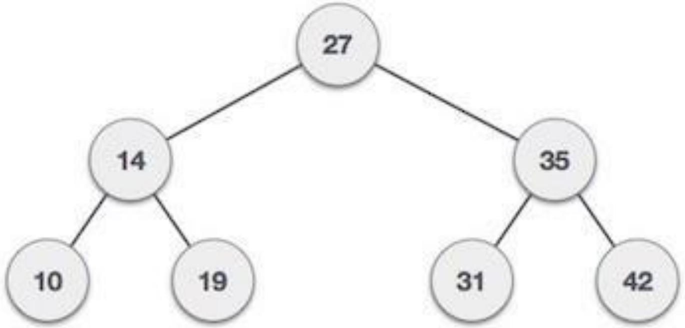
https://github.com/aish21/Algorithms-and-Data-Structures

Great doc covering
All ADTS

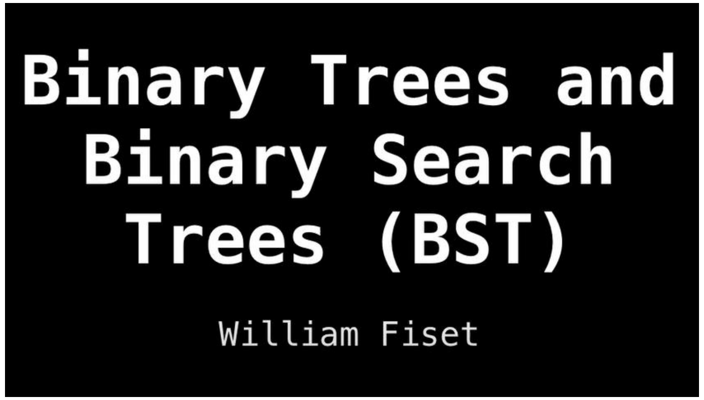
✓ Introduction to BST



✓ Binary Search Tree



✓ Binary Trees

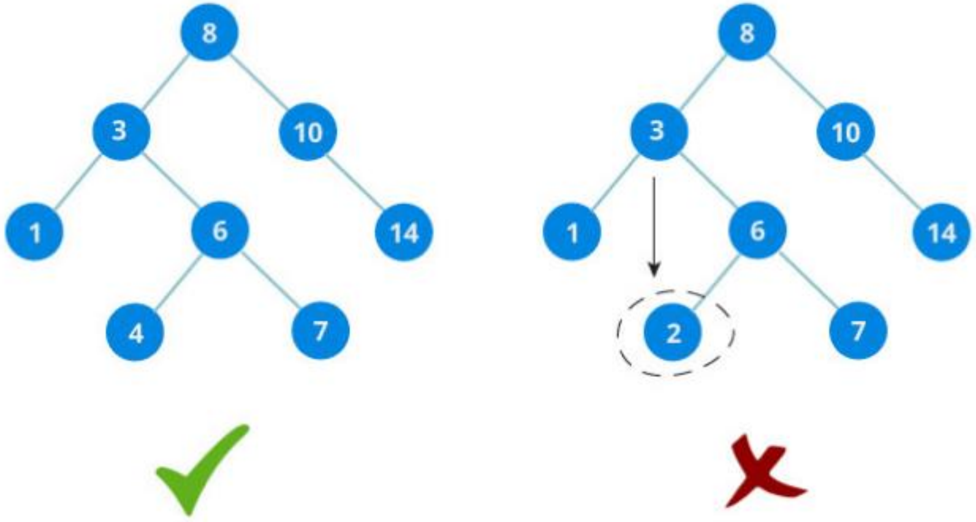


Binary Trees and Binary Search Trees (BST)

William Fiset

ADVANCED ALGORITHM

W8-S1 – Binary Search Tree (BST)



CADT
IDT

🥇 Objectives for today 🥇

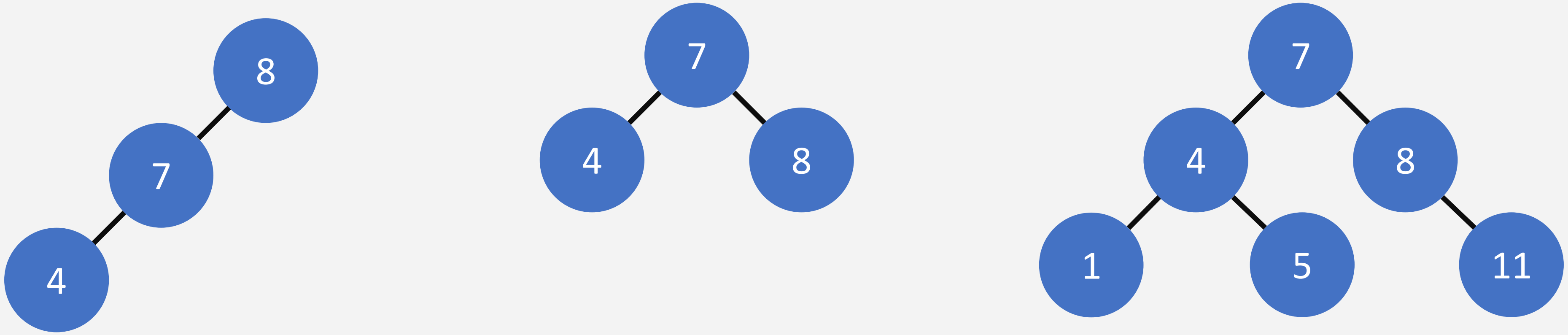
- ✓ Understand **Binary Search Trees (BST)**
- ✓ Perform **BST Operations**
- ✓ Identify BST operation **complexities**

EXPLAIN

What is a **Binary Search Tree** (BST)?

A **binary tree** that satisfies those 2 **invariants**:

- Left subtree has smaller elements
- Right subtree has bigger elements

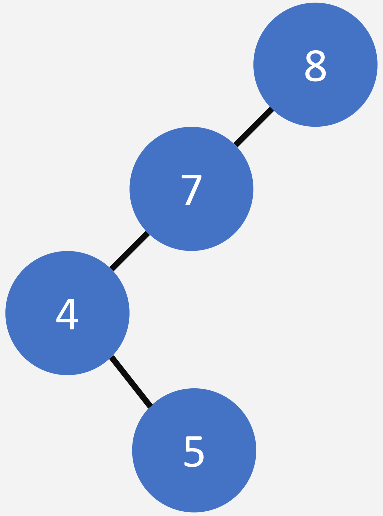


EXPLAIN

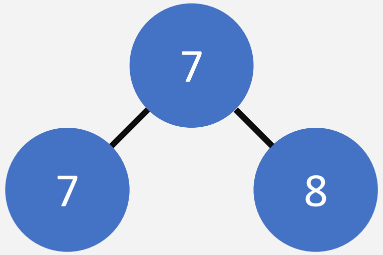
Binary Search Tree (BST)

Is this a valid BST ?

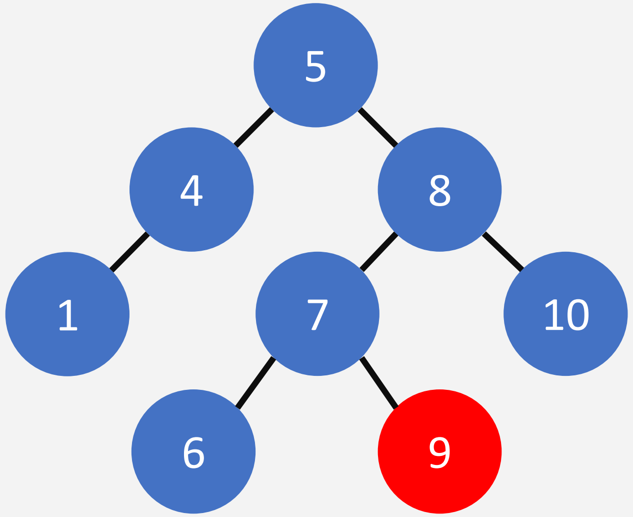
valid



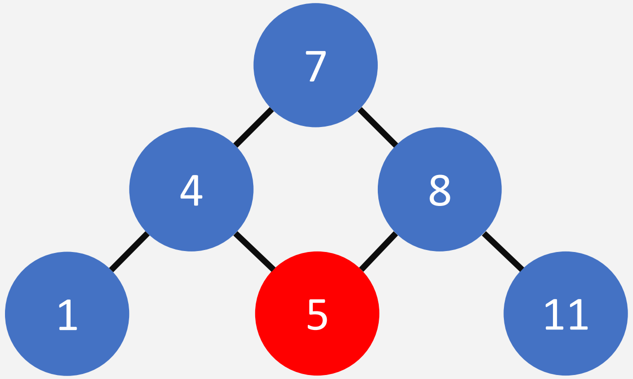
Allow duplicate value or not?



Invalid (9 > 8)



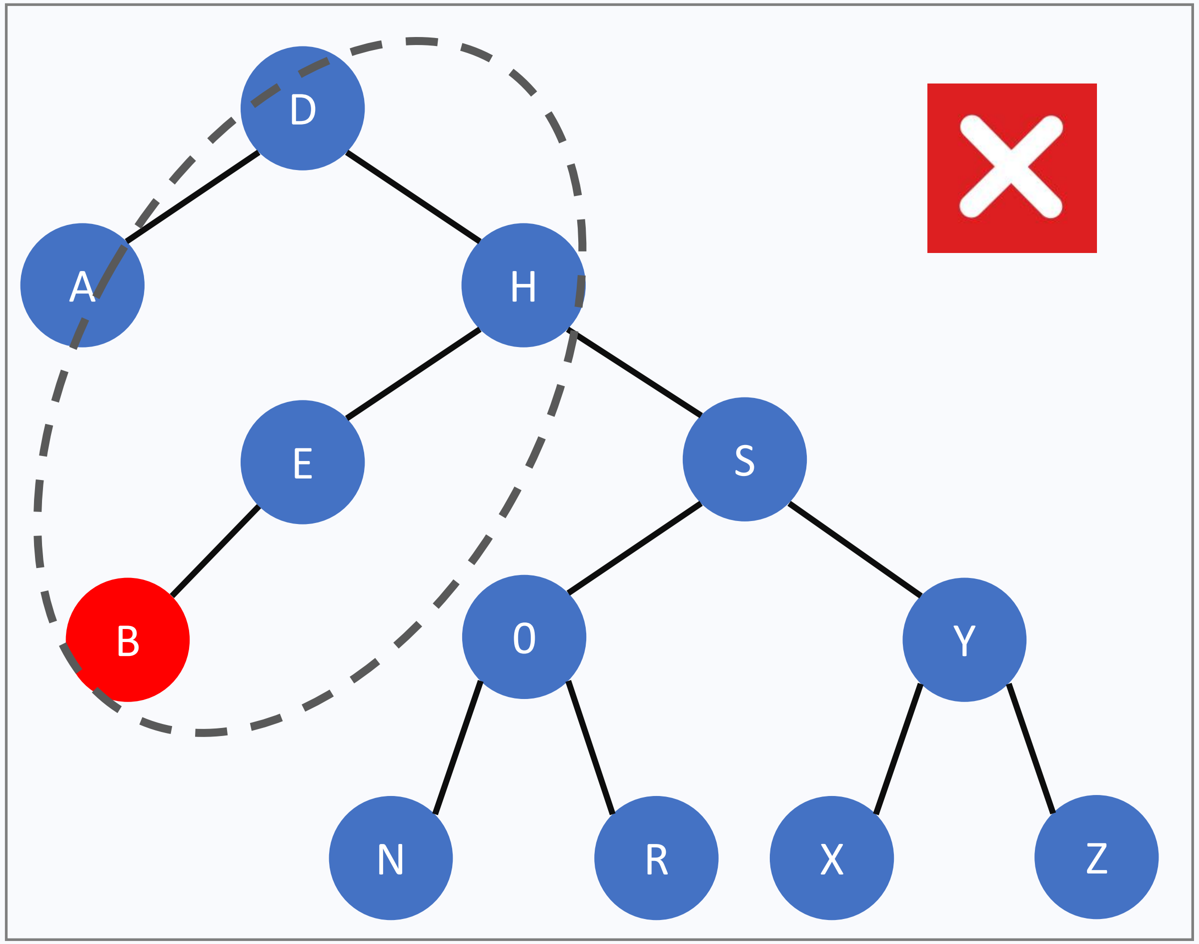
Invalid (not a tree structure)



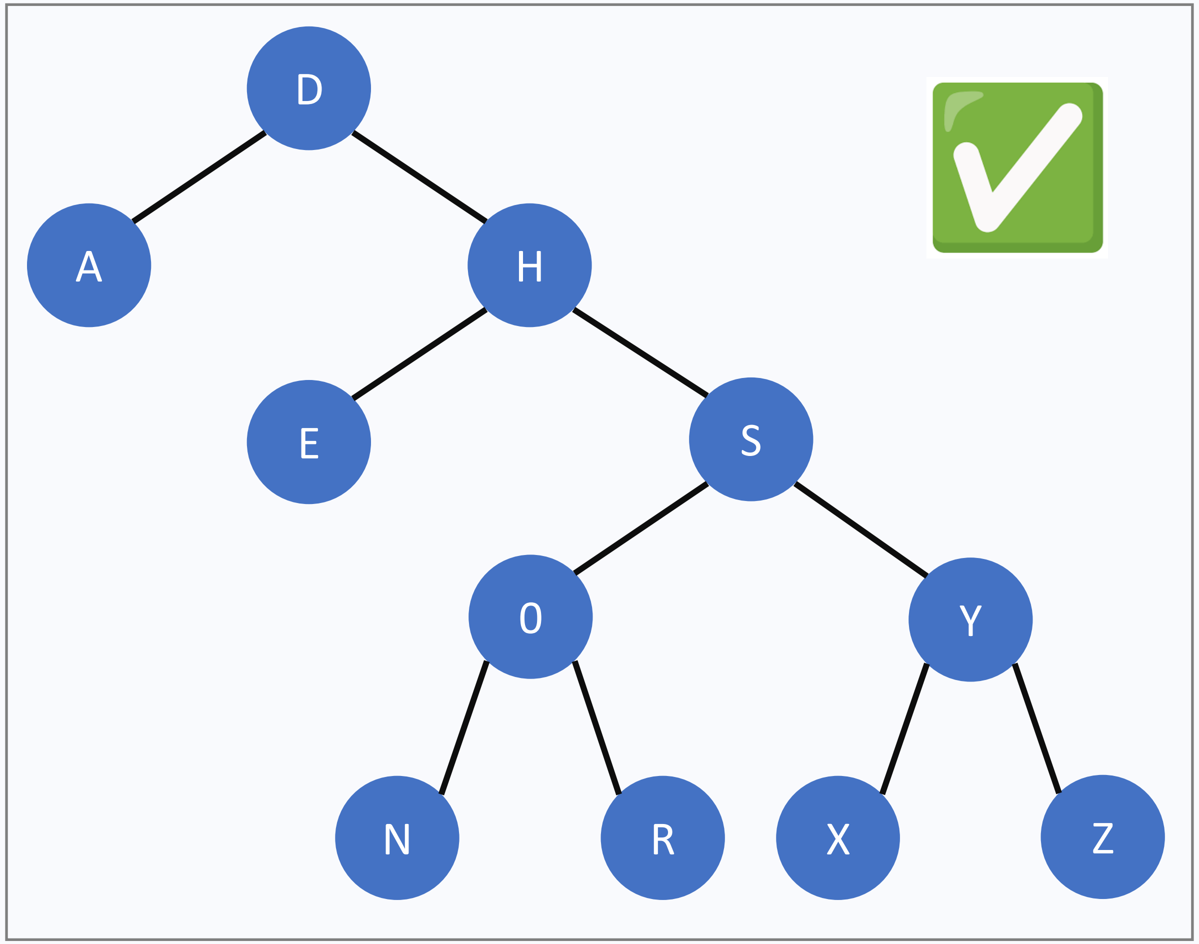
EXPLAIN

Binary Search Tree

A **BST** is not limited to only using numbers. Any data that can be ordered can be placed inside a **BST**.



Node B is in the wrong order



EXPLAIN

When and where are BST **used**?

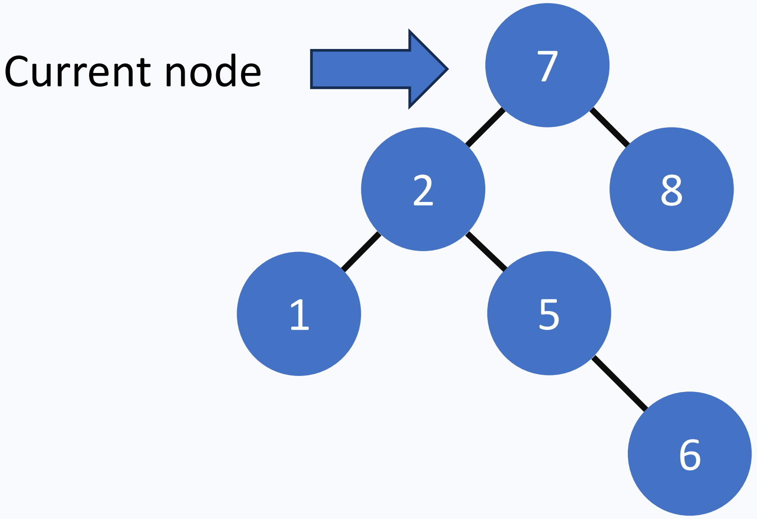
Just few examples !

- ✓ Efficient searching and data retrieval
- ✓ Dynamic sets of data
- ✓ Auto-Complete and Spell Check
- ✓ Decision Tree Algorithm
- ✓ Hierarchical Data Structures

EXPLAIN

Searching in BST

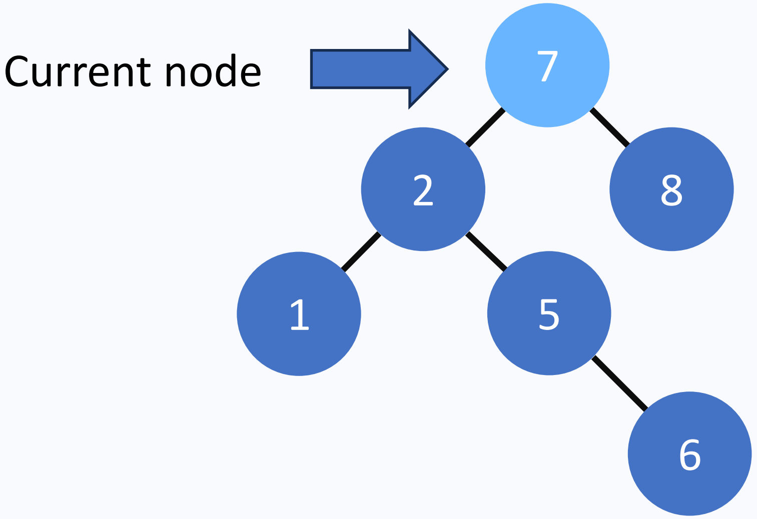
We want to search for the number **6,** We start at the **root**.



EXPLAIN

Searching in BST

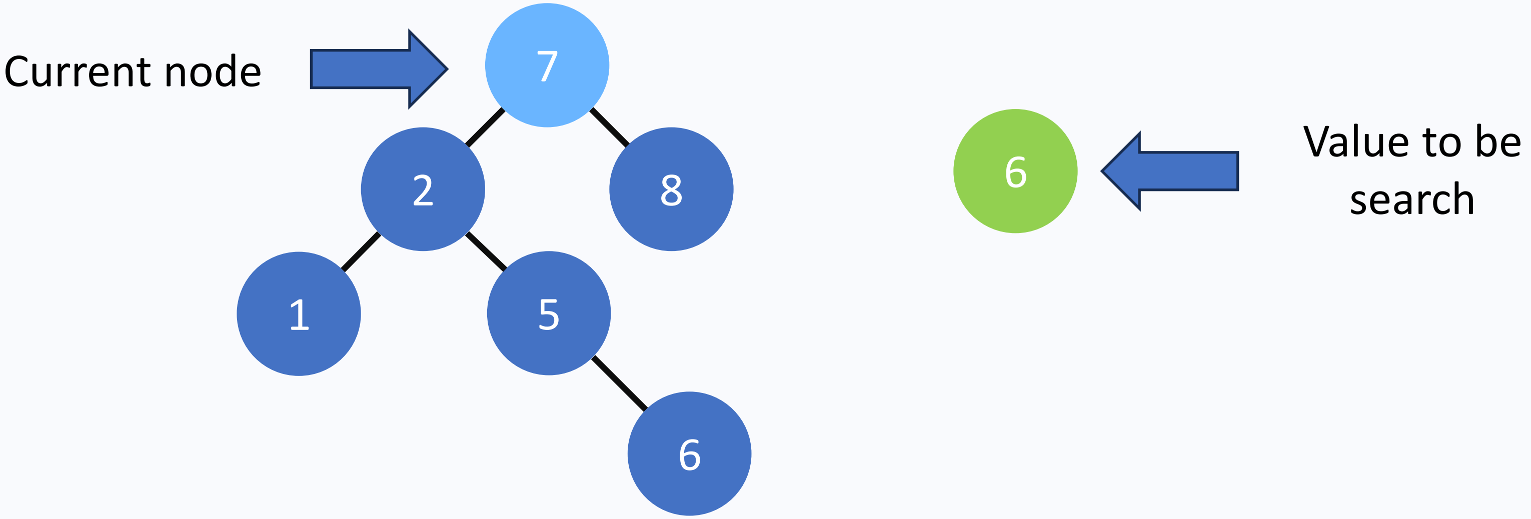
First we **compare** the **value** with root, if root greater than value go **left**, if lower go **right**



EXPLAIN

Searching in BST

First we **compare** the value with root, if root greater than value go **left**,
if lower go **right**

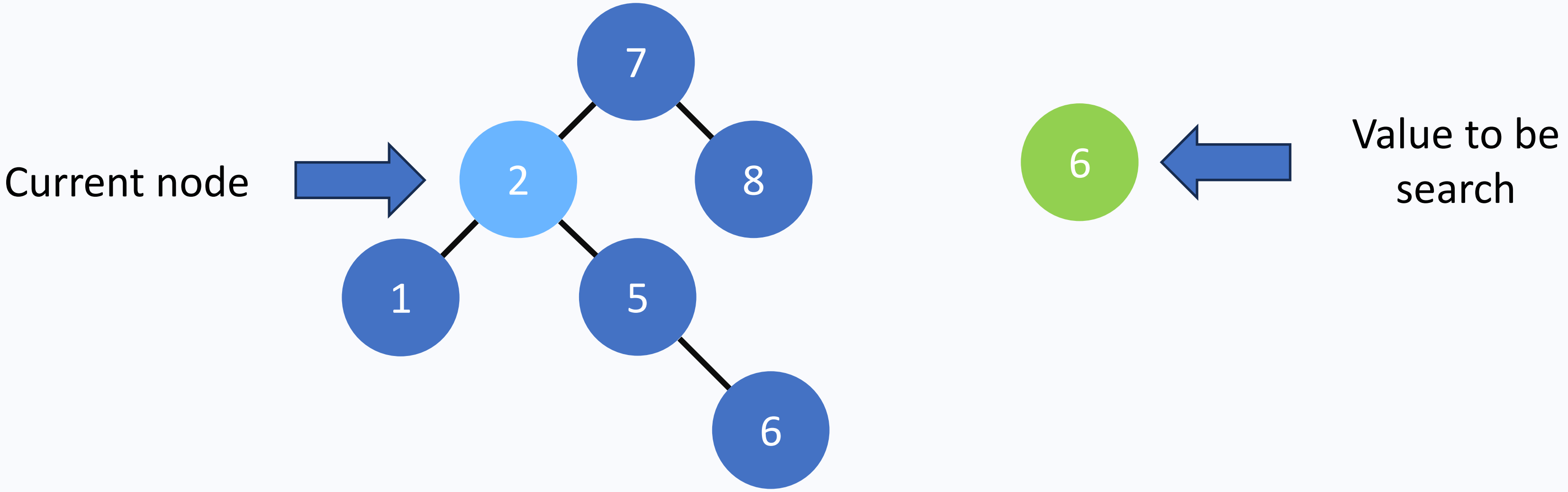


Start with root

EXPLAIN

Searching in BST

First we **compare** the **value** with root, if root greater than value go **left**, if lower go **right**



Current node

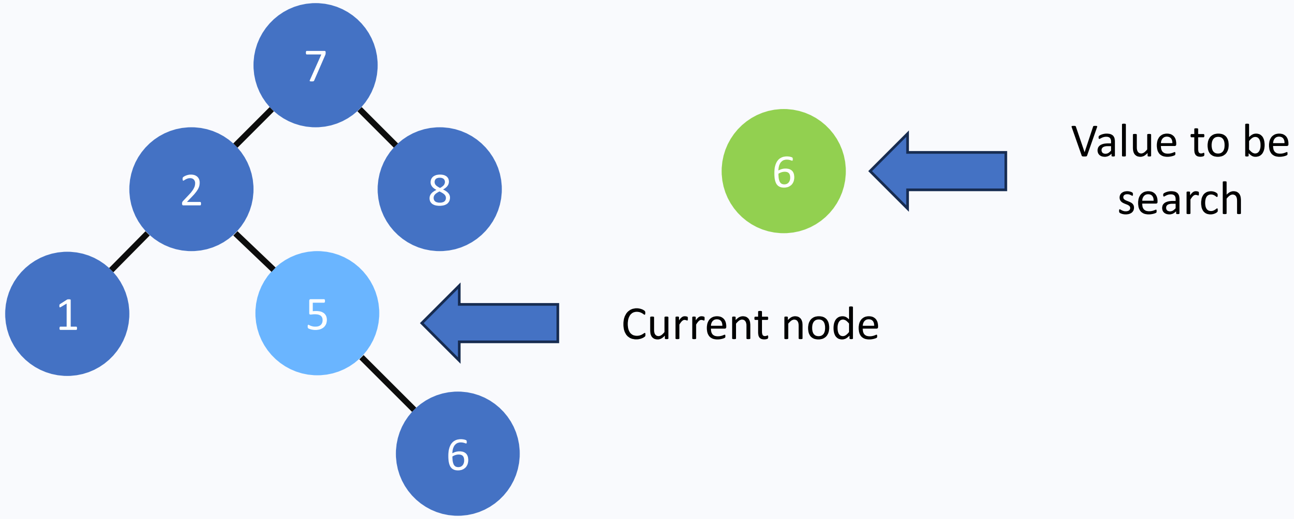
Value to be search

Go left

EXPLAIN

Searching in BST

First we **compare** the value with root, if root greater than value go **left**, if lower go **right**

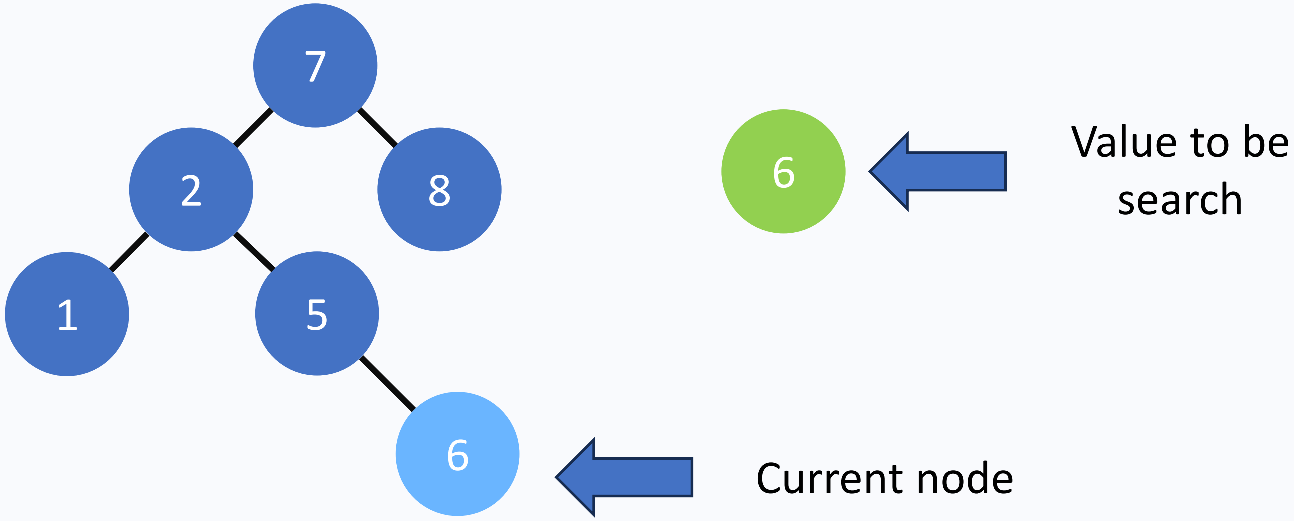


Go right

EXPLAIN

Searching in BST

First we **compare** the value with root, if root greater than value go **left**, if lower go **right**



Value to be search

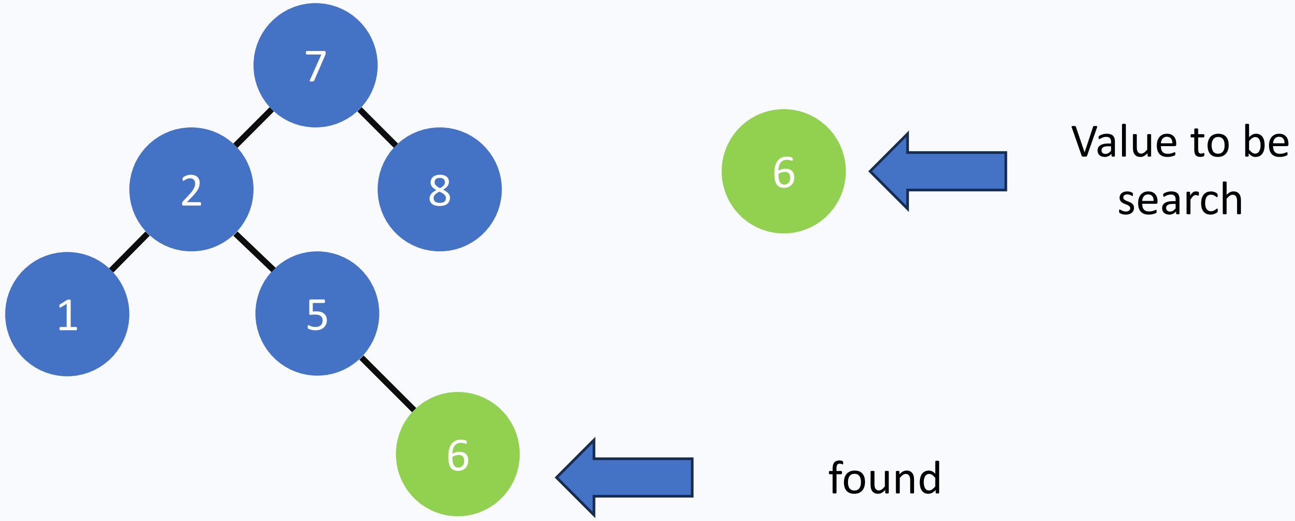
Current node

Go right

EXPLAIN

Searching in BST

First we **compare** the **value** with root, if root greater than value go **left**, if lower go **right**



Value to be search

found

DEMO

Searching in BST

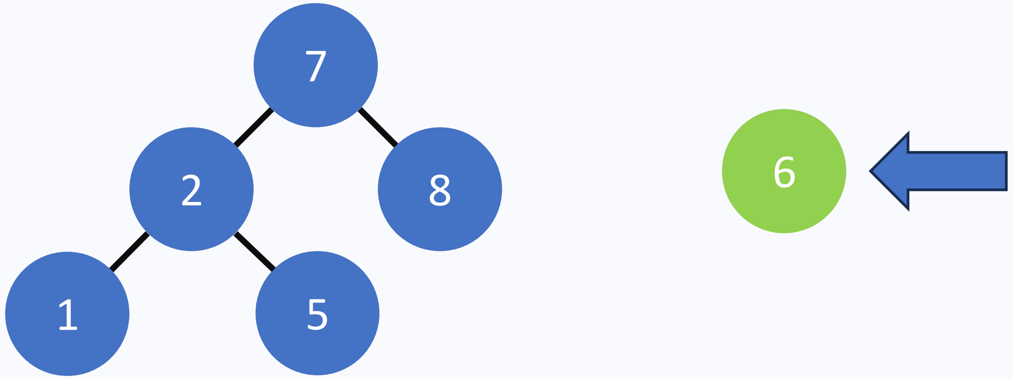
Searching in a BST involves recursively or iteratively traversing the tree by comparing the target value with the current node’s value.

```
function SEARCH(root, value):
    while root != null:
        if root.value == value:
            return true
        if target < root.value:
            root = root.left
        else:
            root = root.right
    return false
```

EXPLAIN

Insertion in BST

A **new node** is always **inserted** at the **leaf** by maintaining the property of the **BST**.

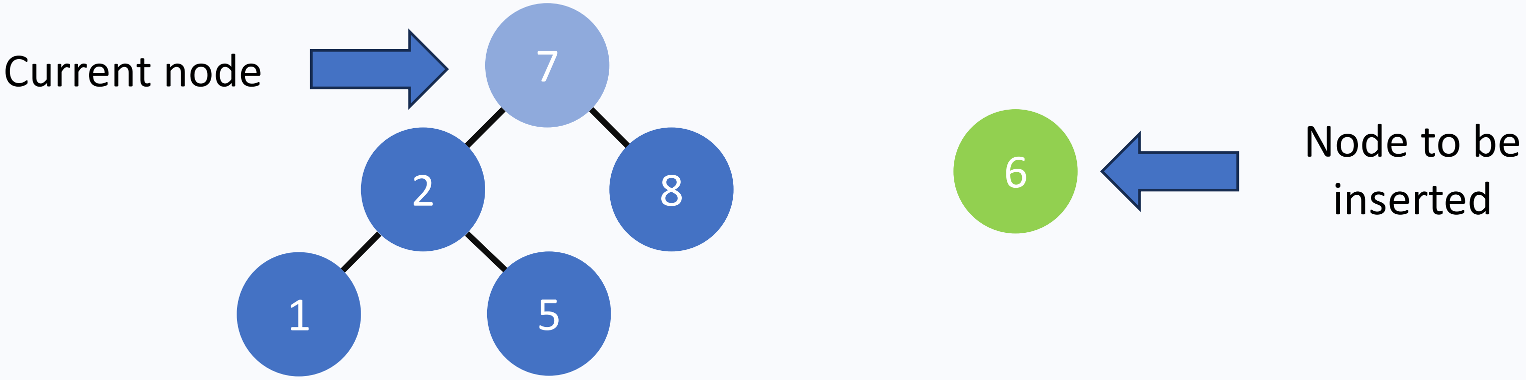


Node to be inserted

EXPLAIN

Insertion in BST

A **new node** is always **inserted** at the **leaf** by maintaining the property of the **BST**.

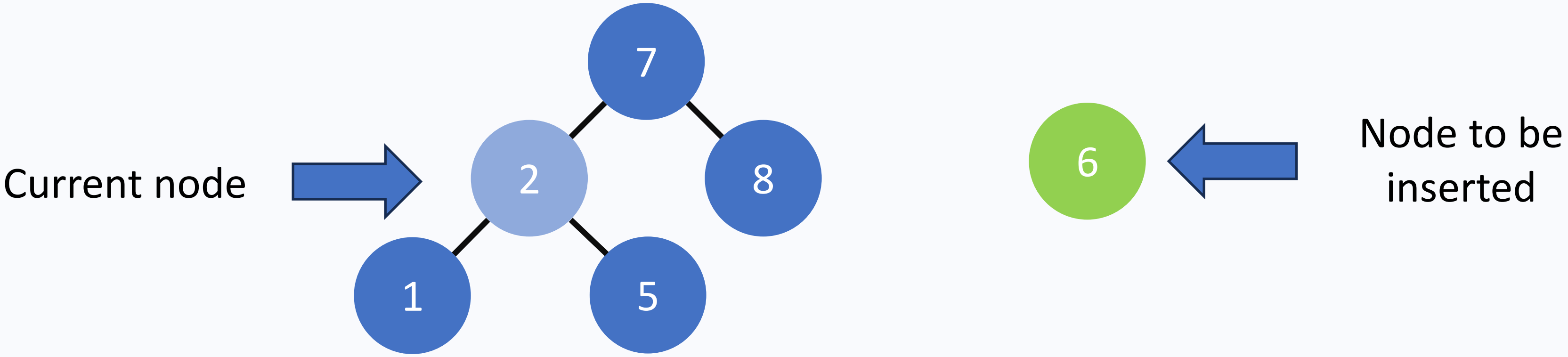


Start with root

EXPLAIN

Insertion in BST

A **new node** is always **inserted** at the **leaf** by maintaining the property of the **BST**.



Current node

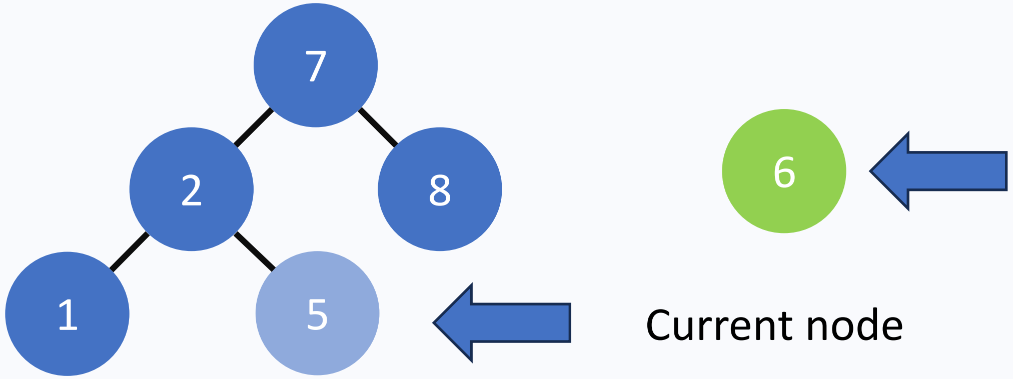
Node to be inserted

Go to left child, because 7 > 6

EXPLAIN

Insertion in BST

A **new node** is always **inserted** at the **leaf** by maintaining the property of the **BST**.



Node to be inserted

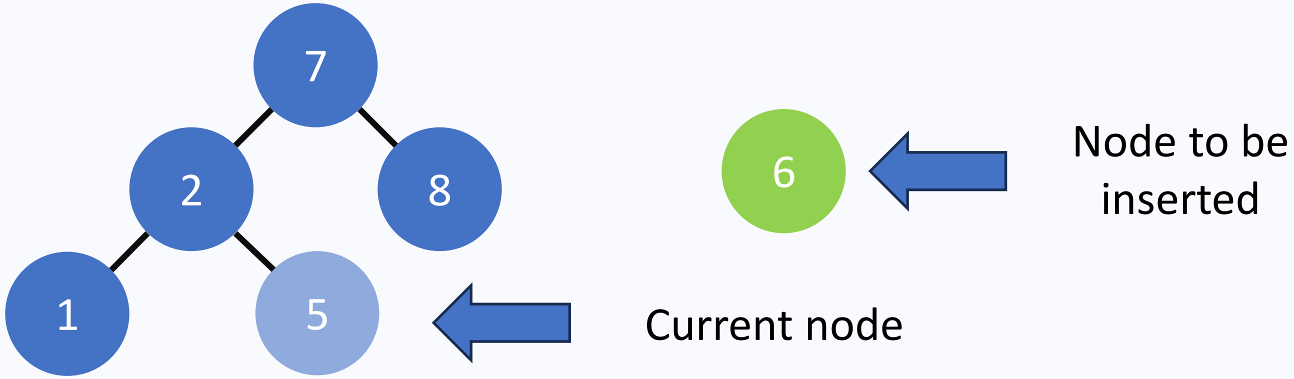
Current node

Go to right child, because 2 < 6

EXPLAIN

Insertion in BST

A **new node** is always **inserted** at the **leaf** by maintaining the property of the **BST**.



Node to be inserted

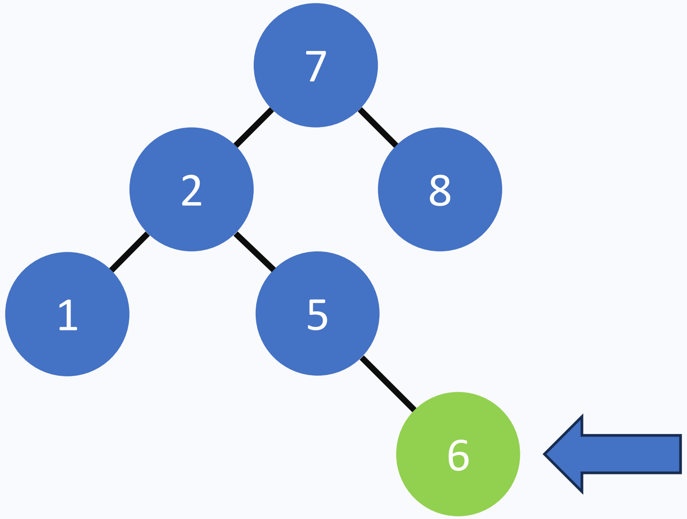
Current node

Go to right child, because 5 < 6
but 5 is leaf

EXPLAIN

Insertion in BST

A **new node** is always **inserted** at the **leaf** by maintaining the property of the **BST**.



Inserted to
BST

Insert 6 to the left child

DEMO

Insertion in BST

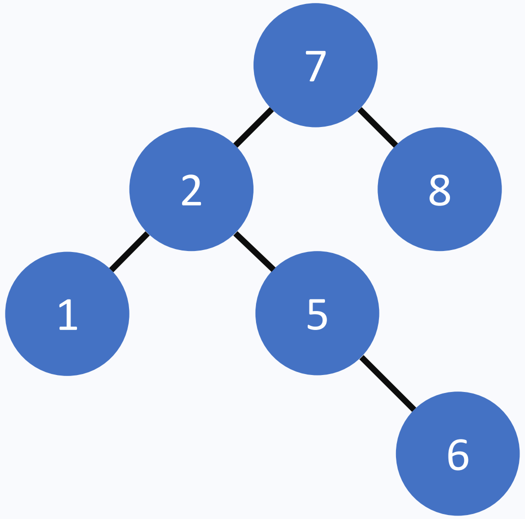
Insertion in a BST places a value in its correct position by traversing left or right based on comparisons, maintaining the tree's ordering property.

```
function INSERT(root, value):
    if root == null:
        root = CREATE_NEW_NODE(value)
        return root

    if value < root.value:
        root.left = INSERT(root.left, value)
    else:
        root.right = INSERT(root.right, value)

    return root
```

EXPLAIN

Deletion in BST

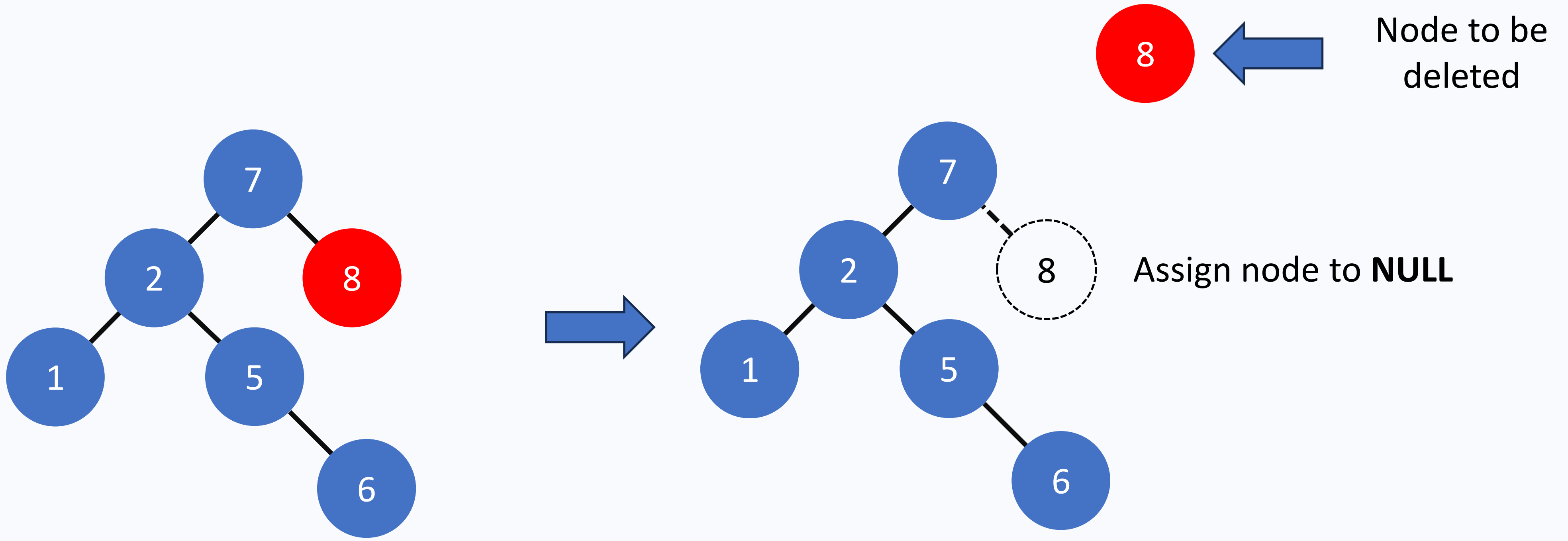
To delete a node in this **BST**, which can be broken down into **three** scenarios:



EXPLAIN

Deletion in BST

1st Scenario: Delete a **leaf** node



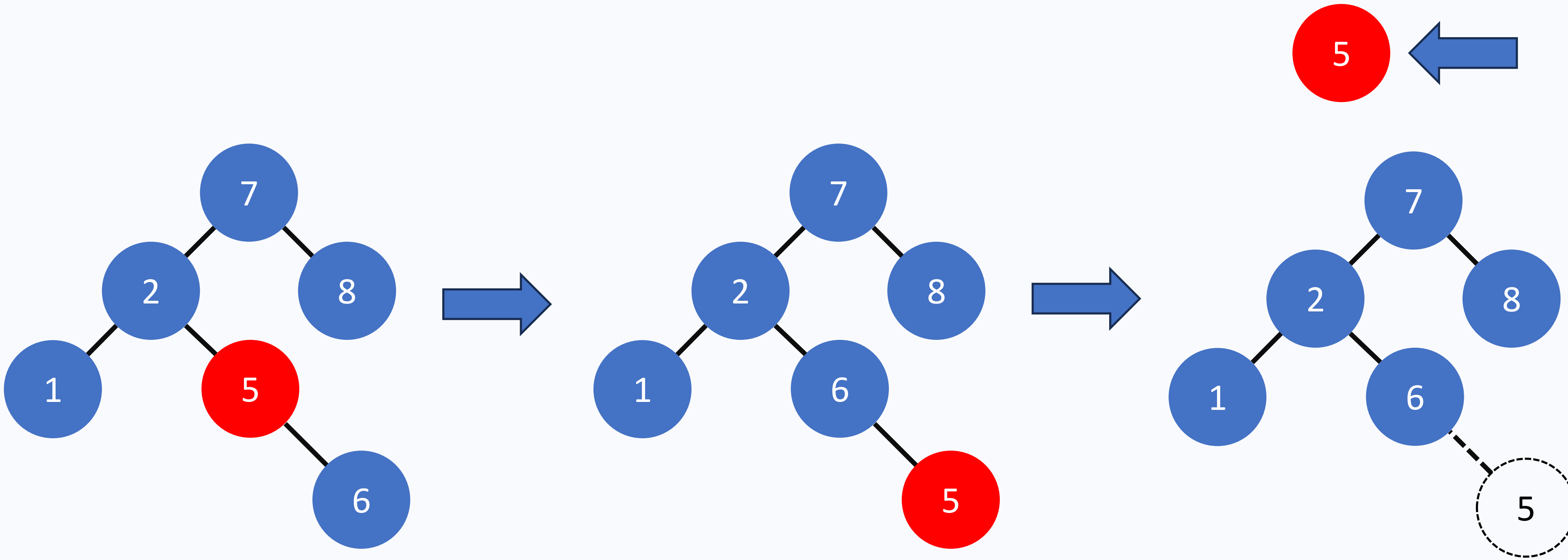
Node to be deleted

Assign node to **NULL**

EXPLAIN

Deletion in BST

2nd scenario: Delete a node with **single** child



Node to be deleted

To delete node **5**

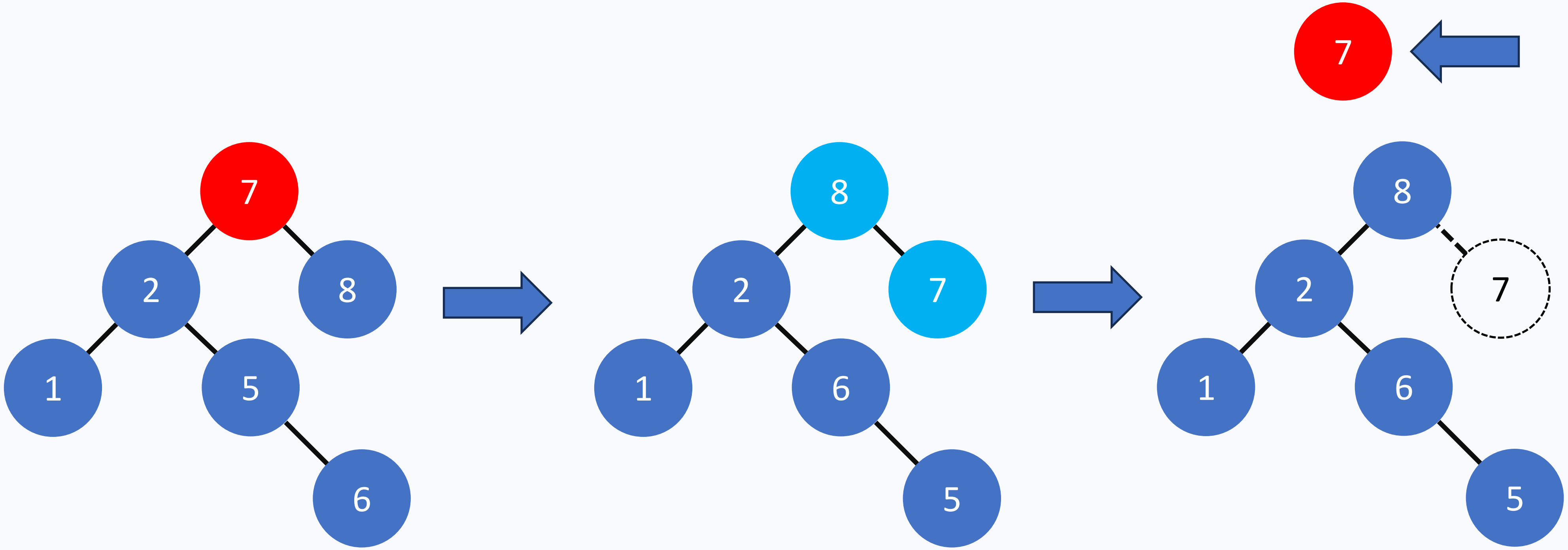
Replace **5** with **6**

Delete node **5**
Assign node to **NULL**

EXPLAIN

Deletion in BST

3rd scenario: Delete a node with **both** child



Node to be deleted

To delete node **7**

Replace **7** with **its in-order successor 8**

Delete node **7**
Assign node to **NULL**

DEMO

Deletion in BST

Deletion in a BST involves finding the node to remove, handling three cases (no children, one child, or two children), and restructuring the tree to maintain its properties

```
function DELETE(root, value):
    if root == null:
        return root

    if value < root.value:
        root.left = DELETE(root.left, value)
    else if value > root.value:
        root.right = DELETE(root.right, value)
    else:
        // Node with only one child or no child
        if root.left == null and root.right == null:
            return null
        else if root.left == null:
            return root.right
        else if root.right == null:
            return root.left

        // Node with two children: Get the inorder successor
        successor = MIN_VALUE(root.right)
        root.value = successor.value
        root.right = DELETE(root.right, successor.value)

    return root
```

3-2-1 Challenge

✓ List three things you **learned** today.
✓ List two **questions** you still have.
✓ List one aspect of the lesson or topic you **enjoyed**.