BINARY SEARCH TREE (BST)

OR

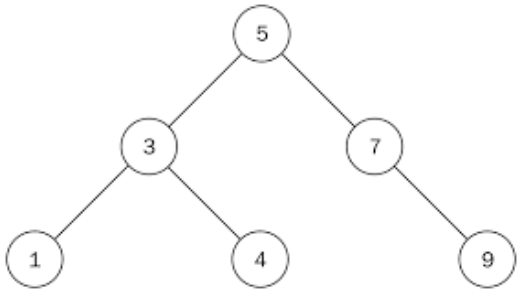
ORDERED BINARY TREE

OR

SORTED BINARY TREE

BINARY SEARCH TREES

- A **binary search tree (BST)** is a binary tree in which all nodes in the left sub-tree have a value less than that of the root node and all nodes in the right sub-tree have a value either equal to or greater than the root node.
- The same rule is applicable to every sub-tree in the tree.



- Due to its efficiency in searching elements, BSTs are widely used in dictionary problems where the code always inserts and searches the elements that are indexed by some key value.

BINARY SEARCH TREES

Various operation can be performed on BST

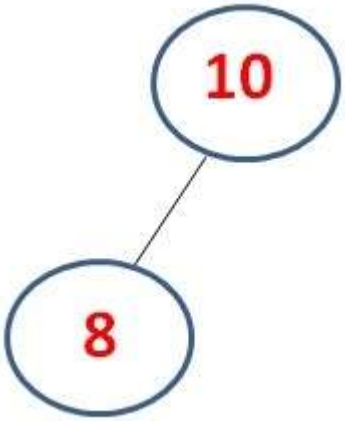
- Insertion
- Deletion
- Search
- Find min
- Find max

Algorithm to Insert a Value in a BST

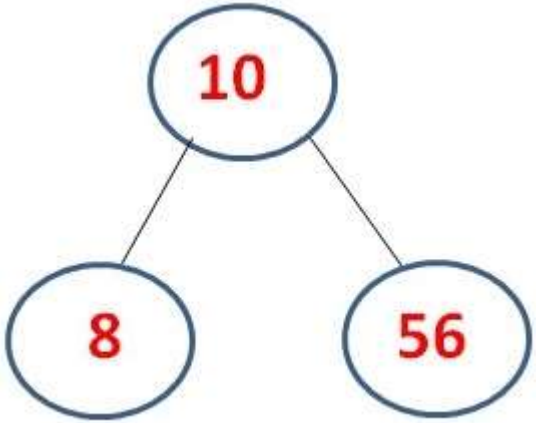
Insert 10, 8, 56, 95, 2, 64, 5, 99, 25, 100

Step 1 : Insert 10

Step 2 : Insert 8



Step 3 : Insert 56

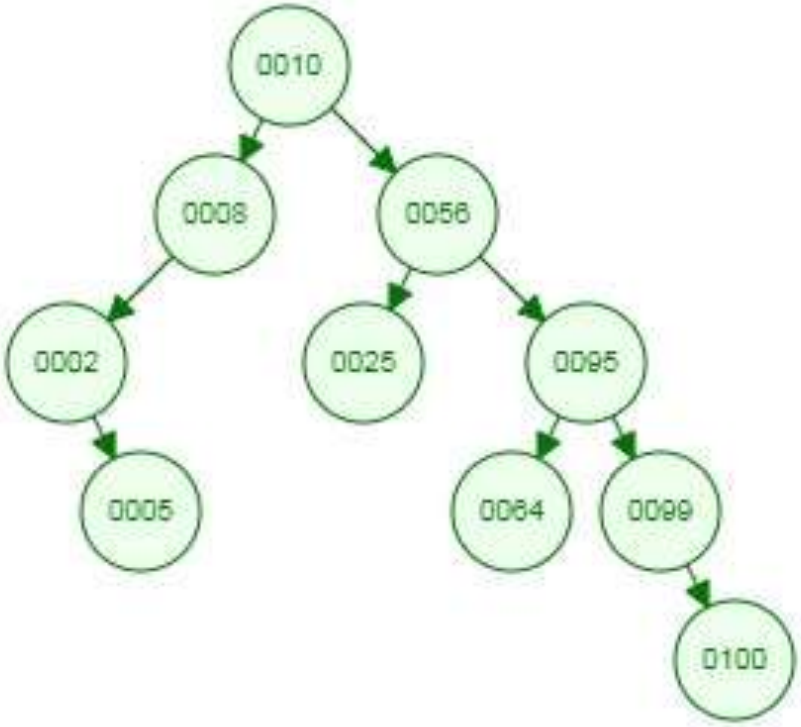


Algorithm to Insert a Value in a BST

VISUALIZATION TOOL DEMO

https://www.cs.usfca.edu/~galles/visualization/Algorithms.html

Insert 10, 8, 56, 95, 2, 64, 5, 99, 25, 100



Algorithm to Insert a Value in a BST

INSERT (TREE, VAL)

```
Step 1: IF TREE = NULL, then
          Allocate memory for TREE
          SET TREE->DATA = VAL
          SET TREE->LEFT = TREE ->RIGHT = NULL
        ELSE
          IF VAL < TREE->DATA
            Insert(TREE->LEFT, VAL)
          ELSE
            Insert(TREE->RIGHT, VAL)
            [END OF IF]
        [END OF IF]
Step 2: End
```

ALGORITHM TO DELETE FROM A BST

- The delete function deletes a node from the binary search tree
- utmost care should be taken that the properties of the binary search tree are not violated and nodes are not lost in the process
- We will take up three cases

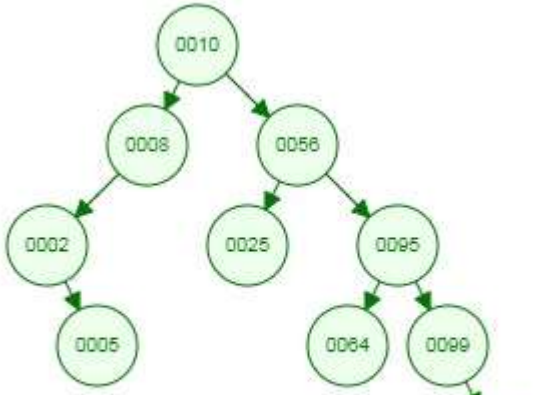
Case 1: Deleting a Node that has No Children

Case 2: Deleting a Node with One Child

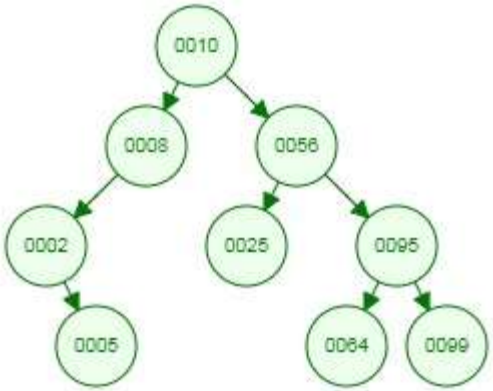
Case 3: Deleting a Node with Two Children

ALGORITHM TO DELETE FROM A BST

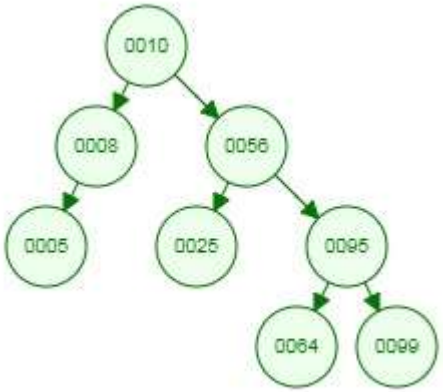
Delete 100, 2, 56



Step 1 : Delete 100

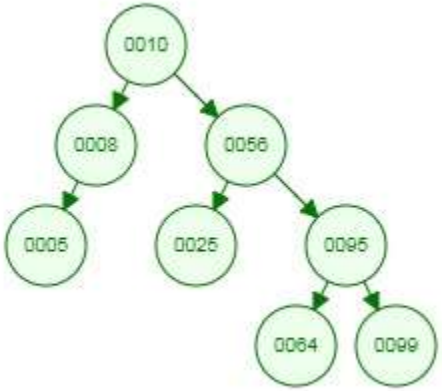


Step 2 : Delete 2

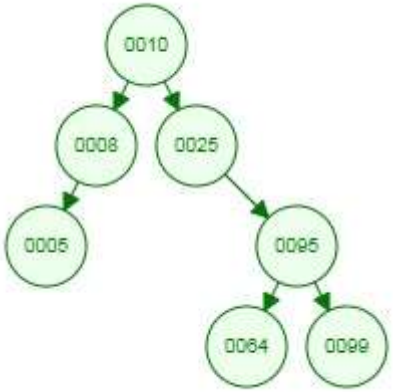


ALGORITHM TO DELETE FROM A BST

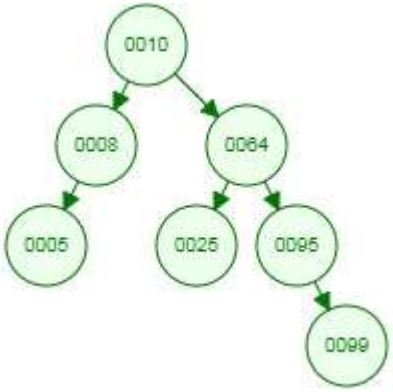
Delete 100, 2, 56



Step 3 : Delete 56



OR



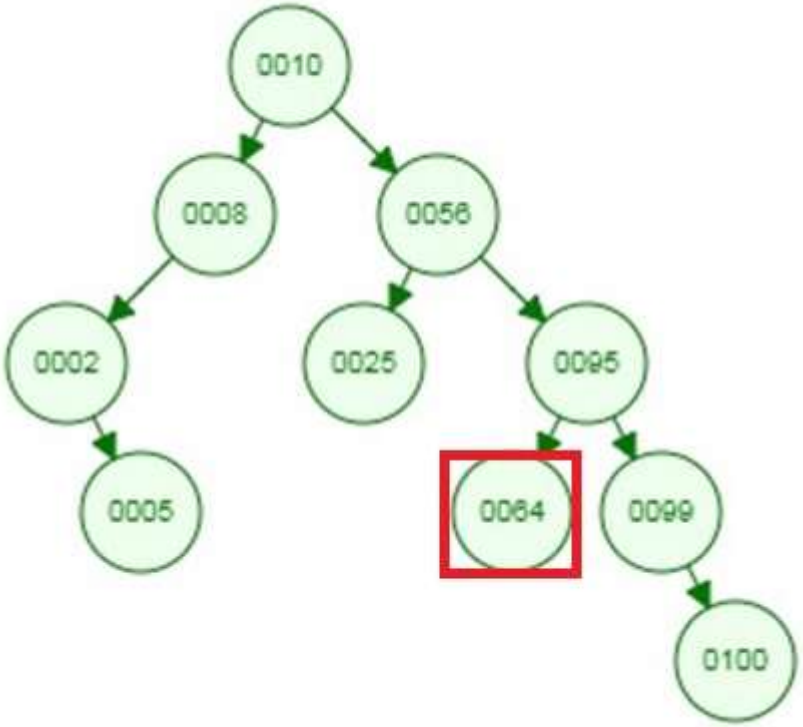
ALGORITHM TO DELETE FROM A BST

DELETE (TREE, VAL)

```
Step 1: IF TREE = NULL, then
          Write "VAL not found in the tree"
        ELSE IF VAL < TREE->DATA
            Delete(TREE->LEFT, VAL)
        ELSE IF VAL > TREE->DATA
          Delete(TREE->RIGHT, VAL)
        ELSE IF TREE->LEFT AND TREE->RIGHT
              SET TEMP = findLargestNode(TREE->LEFT)
              SET TREE->DATA = TEMP->DATA
              Delete(TREE->LEFT, TEMP->DATA)
        ELSE
             SET TEMP = TREE
             IF TREE->LEFT = NULL AND TREE ->RIGHT = NULL
                       SET TREE = NULL
             ELSE IF TREE->LEFT != NULL
                       SET TREE = TREE->LEFT
             ELSE
                       SET TREE = TREE->RIGHT
             [END OF IF]
             FREE TEMP
        [END OF IF]
Step 2: End
```

Algorithm to search a value in a BST

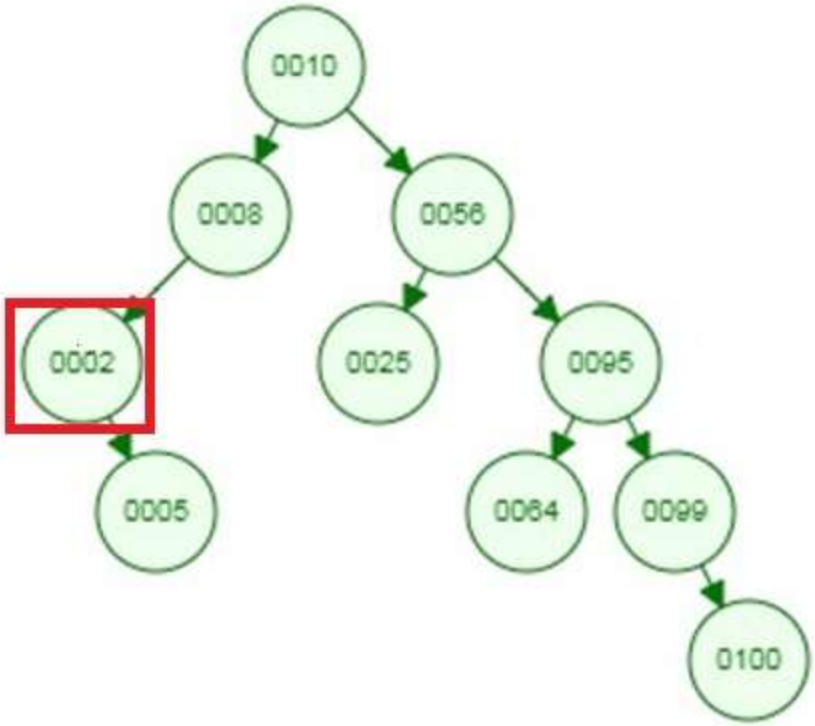
Search 64



Algorithm to Search a Value in a BST

```
SEARCHELEMENT (TREE, VAL)
Step 1: IF TREE->DATA = VAL OR TREE = NULL, then
                Return TREE
        ELSE
            IF VAL < TREE->DATA
                Return searchElement(TREE->LEFT, VAL)
            ELSE
                Return searchElement(TREE->RIGHT, VAL)
            [END OF IF]
        [END OF IF]
Step 2: End
```

Finding the smallest node in a BST

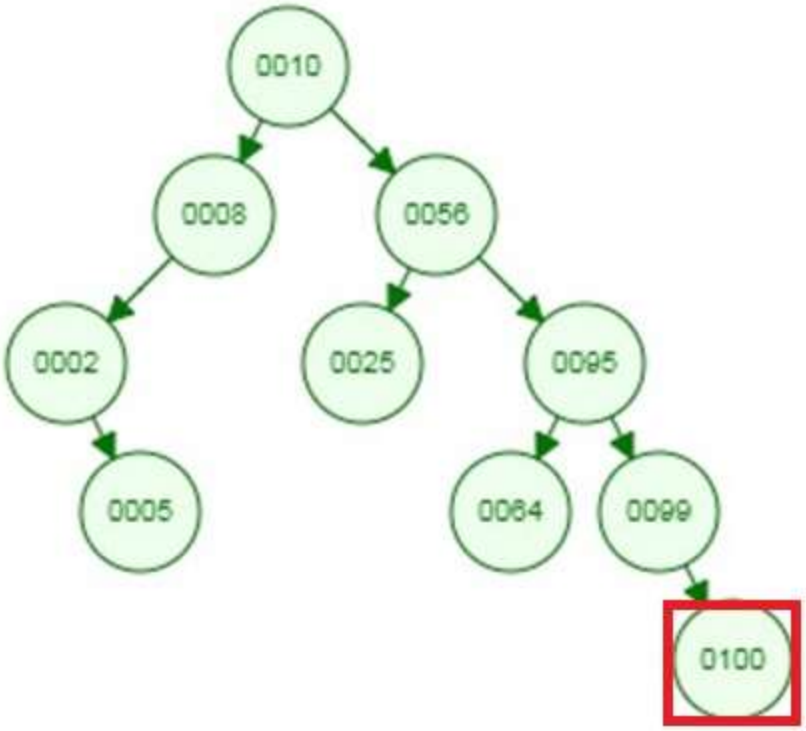


FINDING THE SMALLEST NODE IN A BST

- The basic property of a BST states that the smaller value will occur in the left sub-tree.
- If the left sub-tree is NULL, then the value of root node will be smallest as compared with nodes in the right sub-tree.
- So, to find the node with the smallest value, we will find the value of the leftmost node of the left sub-tree.
- However, if the left sub-tree is empty then we will find the value of the root node.

```
FINDSMALLESTELEMENT (TREE)
Step 1: IF TREE = NULL OR TREE->LEFT = NULL, then
                    Return TREE
          ELSE
                    Return findSmallestElement(TREE->LEFT)
          [END OF IF]
Step 2: End
```

FINDING THE LARGEST NODE IN A BST



FINDING THE LARGEST NODE IN A BST

- The basic property of a BST states that the larger value will occur in the right sub-tree.
- If the right sub-tree is NULL, then the value of root node will be largest as compared with nodes in the left sub-tree.
- So, to find the node with the largest value, we will find the value of the rightmost node of the right sub-tree.
- However, if the right sub-tree is empty then we will find the value of the root node.

```
FINDLARGESTELEMENT (TREE)
Step 1: IF TREE = NULL OR TREE->RIGHT = NULL, then
                  Return TREE
         ELSE
                  Return findLargestElement(TREE->RIGHT)
         [END OF IF]
Step 2: End
```