ECAP770

Advance Data Structures

Ashwani Kumar

Assistant Professor

Learning Outcomes

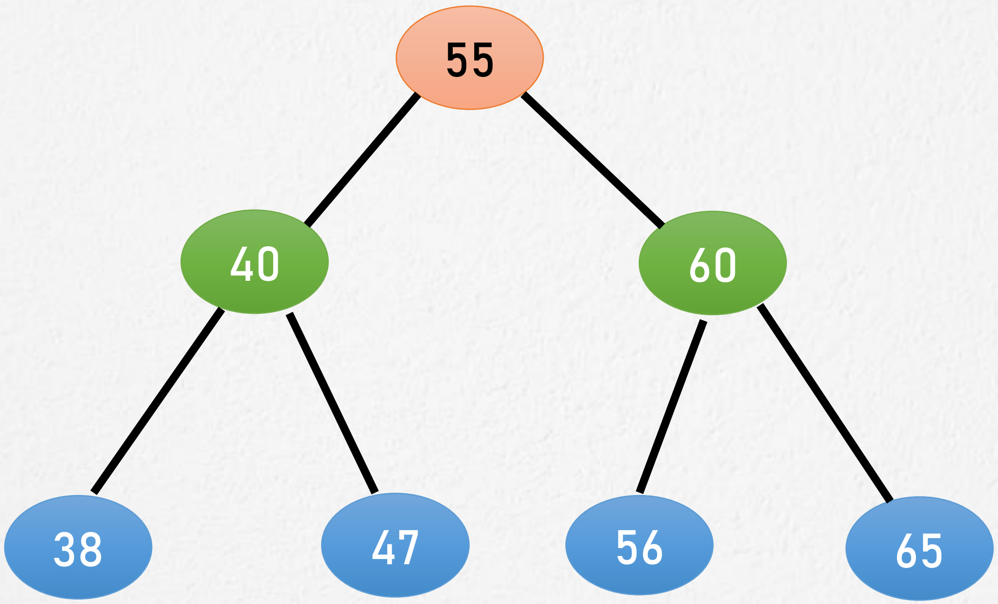
After this lecture, you will be able to

- Understand binary search tree operations
  - Search
  - Insertion
  - Deletion

Binary search tree

- Binary search tree is a non-linear data structure in which one node is connected to n number of nodes. It is a node-based data structure.
- In a binary search tree, the value of all the nodes in the left sub-tree is less than the value of the root.
- Similarly, value of all the nodes in the right sub-tree is greater than or equal to the value of the root.

Binary search tree



Binary Search Tree time complexities

- Search Operation - O(n)
- Insertion Operation - O(1)
- Deletion Operation - O(n)

Search operation

- Searching in BST to find or locate some specific element or node within a data structure.
- It is easy process in the binary search tree due to the fact that, elements in BST are stored in a particular order.

Steps for search operation

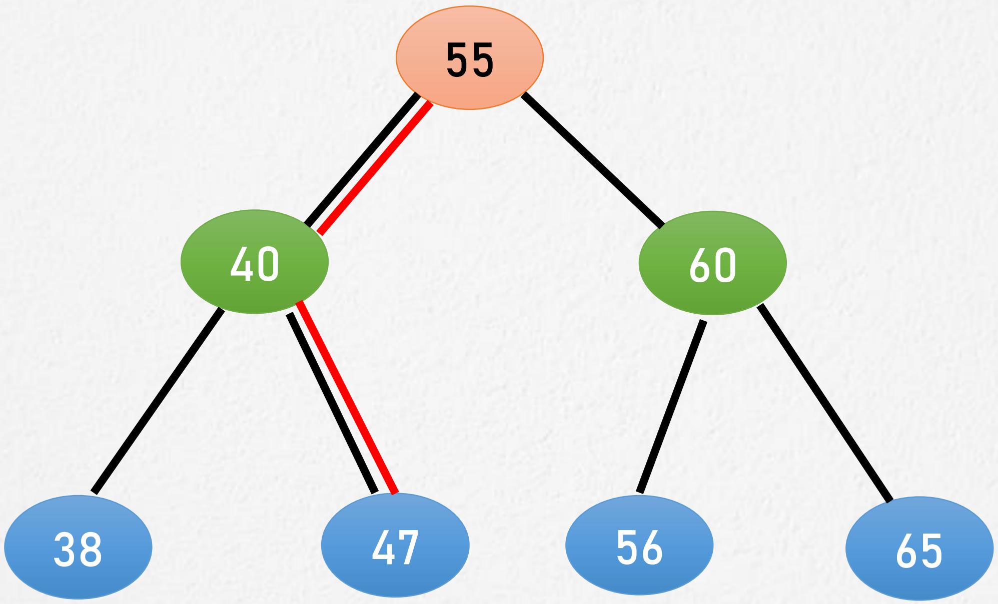
- Compare the element with the root of the tree.
- If the item is matched then return the location of the node.
- Otherwise check if item is less than the element present on root, if so then move to the left sub-tree.
- If not, then move to the right sub-tree.
- Repeat this procedure recursively until match found.
- If element is not found then return NULL.

Algorithm: Search operation

```
Step 1: IF ROOT -> DATA = ITEM OR ROOT = NULL
     Return ROOT
    ELSE
    IF ROOT < ROOT -> DATA
    Return search(ROOT -> LEFT, ITEM)
   ELSE
    Return search(ROOT -> RIGHT,ITEM)
   [END OF IF]
   [END OF IF]
Step 2: END
```

Search operation

Search item = 47



Insert operation

- Insert operation is performed to add a new element in a binary search tree at appropriate location.
- During insert operation user must follow the property of binary search tree at each value.

Steps for insert operation

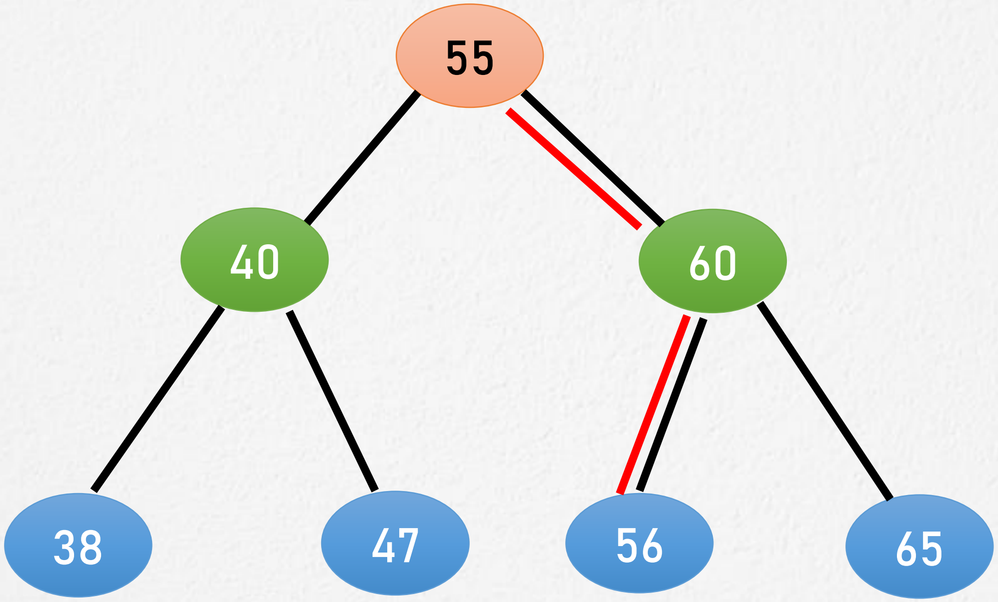
- Allocate the memory for tree.
- Set the data part to the value and set the left and right pointer of tree, point to NULL.
- If the item to be inserted, will be the first element of the tree, then the left and right of this node will point to NULL.
- Else, check if the item is less than the root element of the tree, if this is true, then recursively perform this operation with the left of the root.
- If this is false, then perform this operation recursively with the right sub-tree of the root.

Algorithm: Insert operation

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

Insert operation

Insert item = 56



Delete operation

- Delete operation is performed to delete the specified node from a binary search tree.
- Deleting a node from Binary search tree includes following three cases.

Case 1: Deleting a Leaf node (A node with no children)

Case 2: Deleting a node with one child

Case 3: Deleting a node with two children

Deleting a leaf node

- Step 1 - Find the node to be deleted using search operation
- Step 2 - Delete the node using free function (If it is a leaf) and terminate the function.

Deleting a node with one child

- Step 1 - Find the node to be deleted using search operation
- Step 2 - If it has only one child then create a link between its parent node and child node.
- Step 3 - Delete the node using free function and terminate the function.

Deleting a node with two children

- Step 1 - Find the node to be deleted using search operation
- Step 2 - If it has two children, then find the largest node in its left subtree (OR) the smallest node in its right subtree.
- Step 3 - Swap both deleting node and node which is found in the above step.
- Step 4 - Then check whether deleting node came to case 1 or case 2 or else goto step 2
- Step 5 - If it comes to case 1, then delete using case 1 logic.
- Step 6- If it comes to case 2, then delete using case 2 logic.
- Step 7 - Repeat the same process until the node is deleted from the tree.

Algorithm

Step 1: IF TREE = NULL

Write "item not found in the tree" ELSE IF ITEM < TREE -> DATA

Delete(TREE->LEFT, ITEM)

ELSE IF ITEM > TREE -> DATA

Delete(TREE -> RIGHT, ITEM)

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]

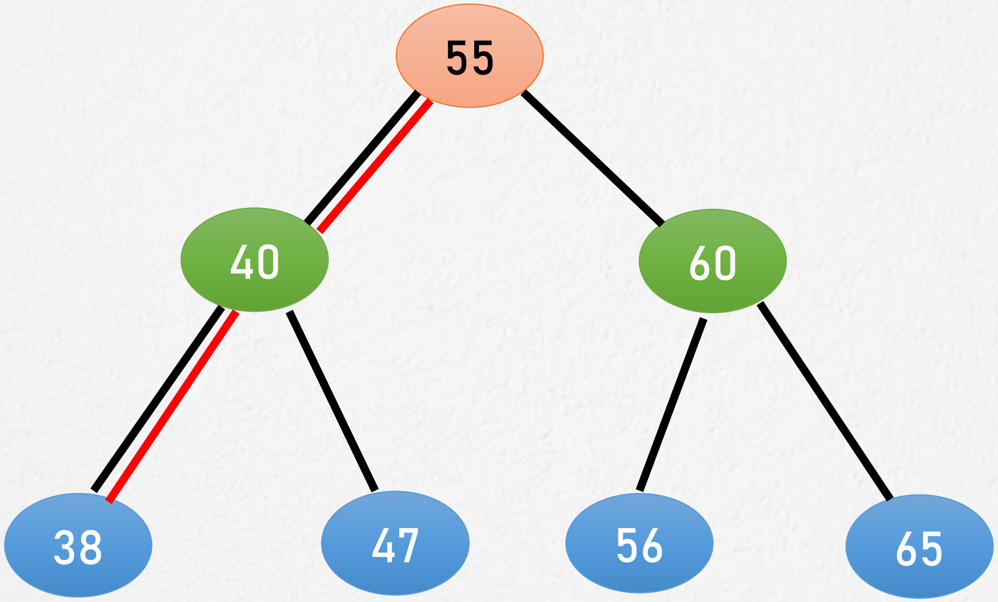
FREE TEMP

[END OF IF]

Step 2: END

Delete operation

Delete item = 38



That’s all for now…