Lecture Notes on Nov/14

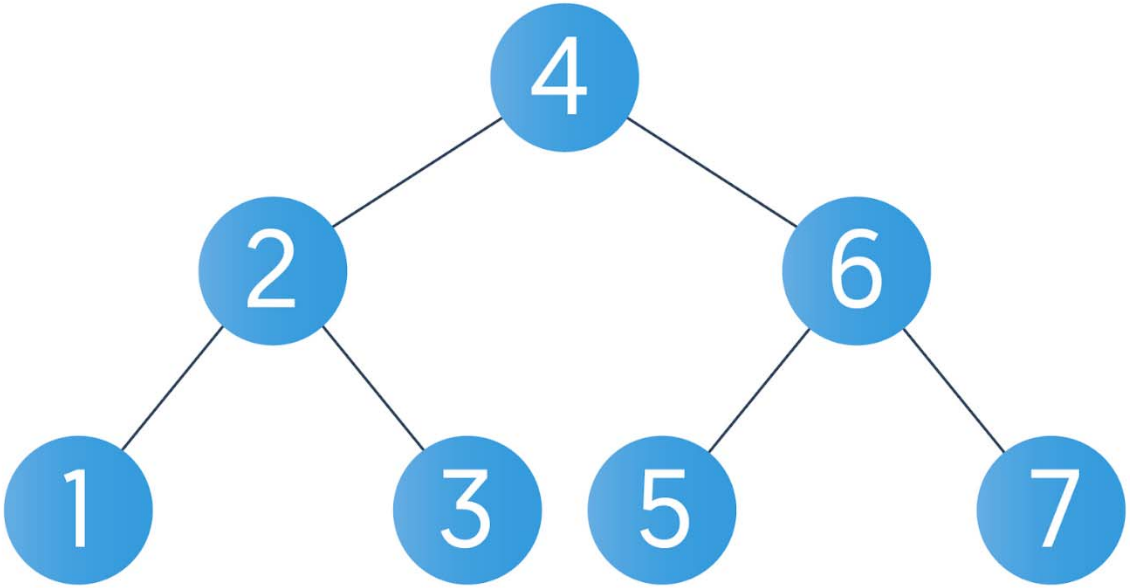
Binary Search Tree

ECE217 Data Structure and Algorithms

Instructor: Dr. Shayan (Sean) Taheri

Binary Search Tree (BST)

- **BST Smallest Node**
- **BST Height**
- **BST Number of Nodes**
- **BST Mirror Image**
- **BST Deletion**
- **Balanced BST**



In order traversal- 1 2 3 4 5 6 7

Pre order traversal- 4 2 1 3 6 5 7

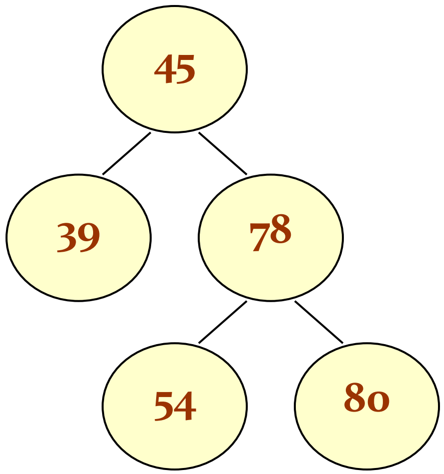
Post order traversal- 1 3 2 5 7 6 4

Level order traversal- 1 4 2 6 1 3 5 7

[BST Traversal]

Finding the Smallest Node in a BST

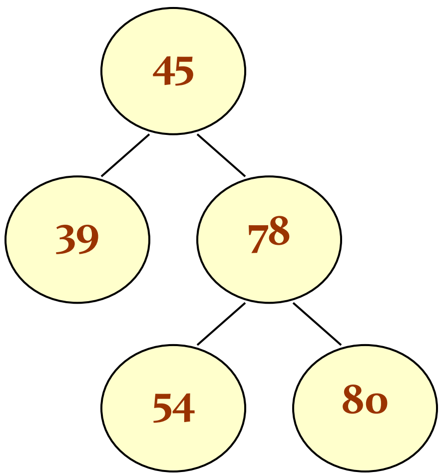
- The basic property of a BST states that the smaller value will occur in the left sub-tree.
- If the left sub-tree is NULL, then the value of root node will be smallest as compared with nodes in the right sub-tree.
- So, to find the node with the smallest value, we will find the value of the leftmost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

Determining the Height of a BST

- In order to determine the height of a BST, we will calculate the height of the left and the right sub-trees.
- Next, whichever height is greater, 1 is added to it.
- **Height of Tree = Height of Tallest Sub-Tree + 1**.
- In figure, since height of right sub-tree is greater than the height of the left sub-tree, then: **Height of Tree = height (right sub-tree) + 1 = 3**.



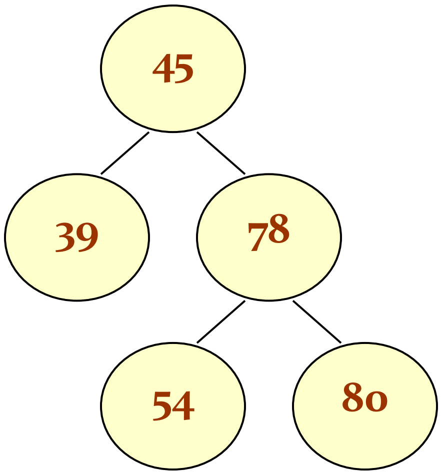

```
Height (TREE)

Step 1: IF TREE = NULL, then
              Return 0
        ELSE
         SET LeftHeight = Height(TREE->LEFT)
           SET RightHeight = Height(TREE->RIGHT)
          IF LeftHeight > RightHeight
             Return LeftHeight + 1
          ELSE
             Return RightHeight + 1
            [END OF IF]
        [END OF IF]
Step 2: End
```

Determining the Number of Nodes

- To calculate the total number of elements/nodes in a BST, we will add one to the number of nodes in the left sub-tree and the right sub-tree.
- **Number of nodes =**

  totalNodes(left sub-tree) + total Nodes(right sub-tree) + 1.

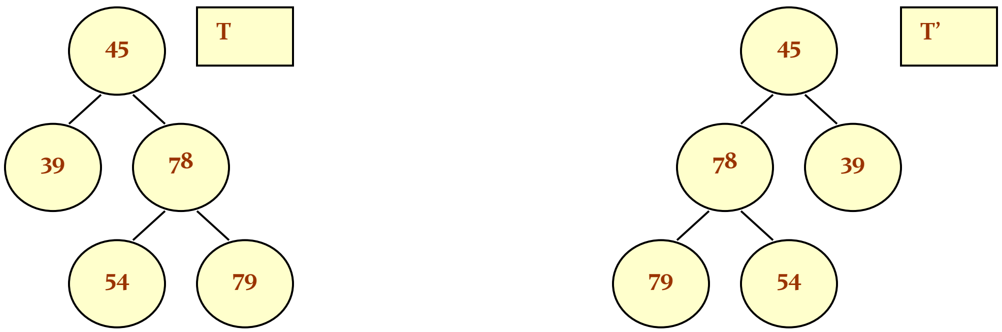



```
totalNodes (TREE)

Step 1: IF TREE = NULL, then
                Return 0
        ELSE
                Return totalNodes(TREE->LEFT)  +
                           totalNodes(TREE->RIGHT) + 1
        [END OF IF]
Step 2: End
```

Finding the Mirror Image of a BST

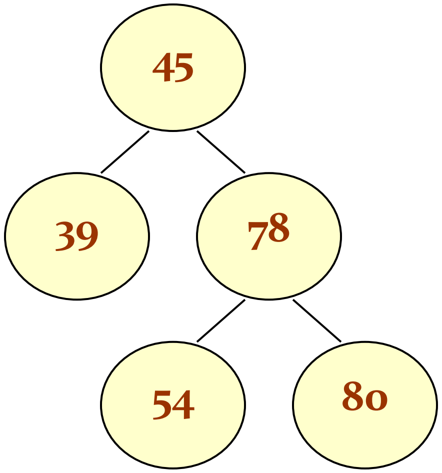
- Mirror image of a binary search tree is obtained by interchanging the left sub-tree with the right sub-tree at every node of the tree.
- For example, given the tree T, the mirror image of T can be obtained as T'.




```
MirrorImage (TREE)
Step 1: IF TREE != NULL , then
               MirrorImage(TREE->LEFT)
               MirrorImage(TREE->RIGHT)
               SET TEMP = TREE->LEFT
               SET TREE->LEFT = TREE->RIGHT
               SET TREE_>RIGHT = TEMP
        [END OF IF]
Step 2: End
```

Deleting a BST

- To delete/remove the entire binary search tree from the memory, we will first delete the elements/nodes in the left sub-tree and then delete the right sub-tree.




```
deleteTree (TREE)
Step 1: IF TREE != NULL , then
                deleteTree (TREE->LEFT)
                deleteTree (TREE->RIGHT)
                Free (TREE)
           [END OF IF]
Step 2: End
```

Balanced BST

- Having a balanced BST means the height of left and right subtrees are equal or there are not much differences for them.
- In a **full balanced binary tree** of n nodes:
  - The search in it can be done in **log(n) time**, **O(log n)**.
  - Depth of recursion is **O(log n)**.
  - Time complexity is **O(log n)**.
  - Space complexity **O(log n)**.
- A BST is not fully balanced in general!

BST Operations in C++ Language

```cpp
// Binary Search Tree (BST) Operations in C++ Language
// Instructor: Dr. Shayan (Sean) Taheri

#include <iostream>
using namespace std;

struct node {
  int key;
  struct node *left, *right;
};

// Create a Tree Node
struct node *newNode(int item) {
  struct node *temp = (struct node *)malloc(sizeof(struct node));
  temp->key = item;
  temp->left = temp->right = NULL;
  return temp;
}

// Inorder Traversal Operation
void inorder(struct node *root) {
  if (root != NULL) {
    // Traverse left side of root
    inorder(root->left);

    // Traverse root
    cout << root->key << " -> ";

    // Traverse right side of root
    inorder(root->right);
  }
}
```

BST Operations in C++ Language (Cont.)

```
// Insertion Operation
struct node *insert(struct node *node, int key) {
  // Return a new node if the tree is empty
  if (node == NULL) return newNode(key);

  // Traverse to the correct location and insert the node
  if (key < node->key)
    node->left = insert(node->left, key);
  else
    node->right = insert(node->right, key);

  return node;
}

// Operation of Finding the Inorder Successor
struct node *minValueNode(struct node *node) {
  struct node *current = node;

  // Find the leftmost leaf (or child)
  while (current && current->left != NULL)
    current = current->left;

  return current;
}
```

BST Operations in C++ Language (Cont.)

```
// Deletion Operation
struct node *deleteNode(struct node *root, int key) {
  // Return if the tree is empty
  if (root == NULL) return root;

  // Find the node to be deleted
  if (key < root->key)
    root->left = deleteNode(root->left, key);
  else if (key > root->key)
    root->right = deleteNode(root->right, key);
  else {
    // If the node is with only one child or no child
    if (root->left == NULL) {
      struct node *temp = root->right;
      free(root);
      return temp;
    } else if (root->right == NULL) {
      struct node *temp = root->left;
      free(root);
      return temp;
    }

    // If the node has two children
    struct node *temp = minValueNode(root->right);

    // Place the inorder successor in position of the node to be deleted
    root->key = temp->key;

    // Delete the inorder successor
    root->right = deleteNode(root->right, temp->key);
  }
  return root;
}
```

BST Operations in C++ Language (Cont.)

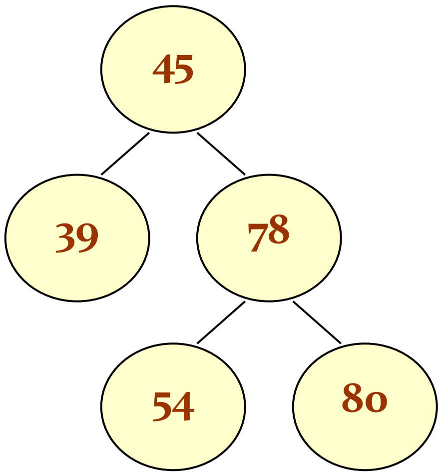
```cpp
// Driver Code
int main() {

  // Task 1: Create a Tree Node

  // Task 2: Execute BST Operations on the Created Tree

}
```

- **Example Tree using BST Operations**



Assignment

- Reading Assignment:
  - Data Structures Using C by Reema Thareja, Oxford University Press; 2nd Edition.
    - Chapter 9. Trees (Starting Page: 279).
    - Chapter 10. Efficient Binary Trees (Starting Page: 298).
- **Theoretical Assignment 2** Deadline: **November/21/2022**.

Questions?