Tree

Manoj Wairiya

Department of Computer Science
MNNIT ALLAHABAD

February 26, 2018

Outline

Trees

- A tree is recursively defined as a set of one or more nodes where one node is designated as the root of the tree and all the remaining nodes can be partitioned into non-empty sets each of which is a sub-tree of the root.
- Types of Trees
  - General Trees
  - Forests
  - Binary Trees
  - Expression Trees
  - Tournament Trees

General Trees

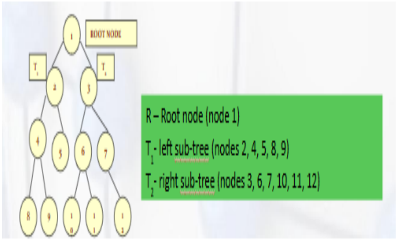
- General trees are data structures that store elements hierarchically.
- The top node of a tree is the root node and each node, except the root, has a parent.
- A node in a general tree (except the leaf nodes) may have zero or more sub-trees.
- General trees which have 3 sub-trees per node are called ternary trees.
- However, the number of sub-trees for any node may be variable. For example, a node can have 1 sub-tree, whereas some other node can have 3 sub-trees.

Forests

- A forest is a disjoint union of trees. A set of disjoint trees (or forest) is obtained by deleting the root and the edges connecting the root node to nodes at level 1.
- Every node of a tree is the root of some sub-tree. Therefore, all the sub-trees immediately below a node form a forest.
- A forest can also be defined as an ordered set of zero or more general trees.
- While a general tree must have a root, a forest on the other hand may be empty because by definition it is a set, and sets can be empty.
- We can convert a forest into a tree by adding a single node as the root node of the tree.

Binary Trees

- A binary tree is a data structure which is defined as a collection of elements called nodes.
- In a binary tree, the topmost element is called the root node, and each node has 0, 1, or at the most 2 children.
- . Every node contains a data element, a "left" pointer which points to the left child, and a "right" pointer which points to the right child.
- The root element is pointed by a "root" pointer.
- If root = NULL, then it means the tree is empty.


ROOT NODE
T1
T2
1
2
3
4
5
6
7
8
9
10
11
12
R – Root node (node 1)
T1 - left sub-tree (nodes 2, 4, 5, 8, 9)
T2 - right sub-tree (nodes 3, 6, 7, 10, 11, 12)

Binary Trees -Key Terms

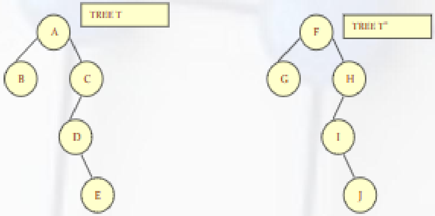
- **Parent:** If N is any node in T that has left successor S1 and right successor S2, then N is called the parent of S1 and S2. Correspondingly, S1 and S2 are called the left child and the right child of N. Every node other than the root node has a parent.
- **Sibling:** S1 and S2 are said to be siblings. In other words, all nodes that are at the same level and share the same parent are called siblings (brothers).
- **Level number:** Every node in the binary tree is assigned a level number. The root node is defined to be at level 0. The left and right child of the root node have a level number 1. Similarly, every node is at one level higher than its parents.
- **Leaf node:** A node that has no children.
- **Degree:** Degree of a node is equal to the number of children that a node has.

Binary Trees - Key Terms

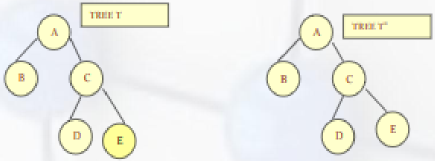
- **In-degree** of a node is the number of edges arriving at that node.
- **Out-degree** of a node is the number of edges leaving that node.
- **Edge:** It is the line connecting a node N to any of its successors
- **Path:** A sequence of consecutive edges is called a path.
- **Depth:** The depth of a node N is given as the length of the path from the root to the node N. The depth of the root node is zero.
- **Height:** It is the total number of nodes on the path from the root node to the deepest node in the tree. A tree with only a root node has a height of 1.
- A binary tree of height h has at least h nodes and at most 2h 1 nodes. This is because every level will have at least one node and can have at most 2 nodes.
- The height of a binary tree with n nodes is at least log2(n+1) and at most n.

Binary Trees - Key Terms

Similar binary trees: Given two binary trees T and T' are said to be similar if both these trees have the same structure.

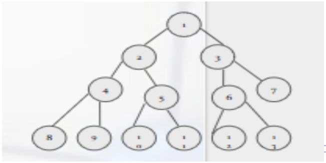


Copies of binary trees: Two binary trees T and T' are said to be *copies* if they have similar structure and same content at the corresponding nodes.



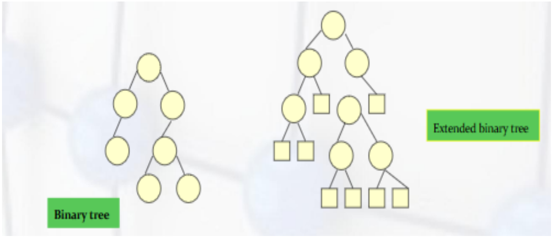
Complete Binary Trees

- A complete binary tree is a binary tree which satisfies two properties.
- First, in a complete binary tree every level, except possibly the last, is completely filled.
- Second, all nodes appear as far left as possible.
- In a complete binary tree T_n, there are exactly n nodes and level r of T can have at most 2^r nodes.
- The formula to find the parent, left child and right child can be given as:
- If K is a parent node, then its left child can be calculated as 2 * K and its right child can be calculated as 2 * K + 1. For example, the children of node 4 are 8 (2*4) and 9 (2* 4 + 1)
- Similarly, the parent of node K can be calculated as $\|K/2\|$.



Extended Binary Trees

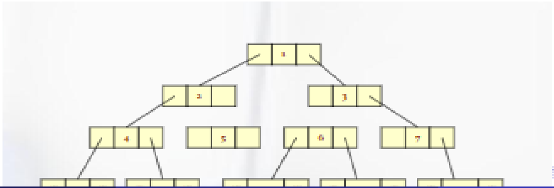
- A binary tree T is said to be an extended binary tree (or a 2-tree) if each node in the tree has either no child or exactly two children.
- In an extended binary tree nodes that have two children are called internal nodes and nodes that have no child or zero children are called external nodes. In the figure internal nodes are represented using a circle and external nodes are represented using squares.
- To convert a binary tree into an extended tree, every empty sub-tree is replaced by a new node. The original nodes in the tree are the internal nodes and the new nodes added are called the external nodes.



Linked Representation of Binary Trees

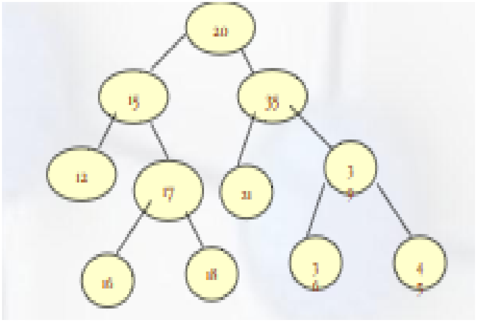
- In computers memory, a binary tree can be maintained either using a linked representation or using sequential representation.
- In linked representation of binary tree, every node will have three parts: the data element, a pointer to the left node and a pointer to the right node. So in C, the binary tree is built with a node type given as below.

```
struct node
{ int data;
struct node* left;
struct node* right;
};
```



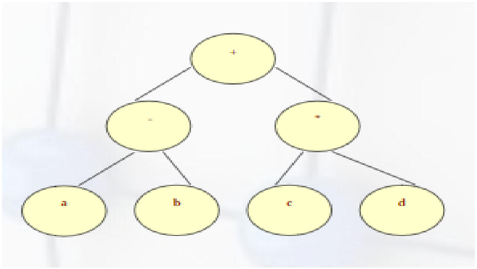
Sequential Representation of Binary Trees

- Sequential representation of trees is done using a single or one dimensional array. Though, it is the simplest technique for memory representation, it is very inefficient as it requires a lot of memory space.
- A sequential binary tree follows the rules given below:
- One dimensional array called TREE is used.
- The root of the tree will be stored in the first location. That is, TREE[1] will store the data of the root element.
- The children of a node K will be stored in location (2*K) and (2*K+1).
- The maximum size of the array TREE is given as (2h-1), where h is the height of the tree.
- An empty tree or sub-tree is specified using NULL. If TREE[1] = NULL, then the tree is empty.


20
15
12
17
16
18

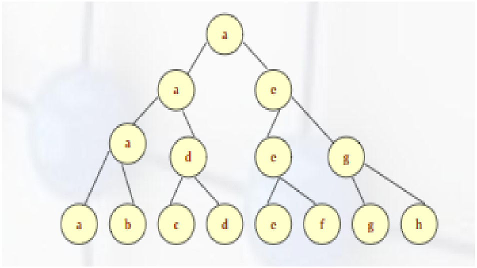
Expression Trees

- Binary trees are widely used to store algebraic expressions. For example, consider the algebraic expression Exp given as: Exp = (a - b) + (c * d)
- This expression can be represented using a binary tree as shown in figure



Tournament Trees

- In a tournament tree (also called a selection tree), each external node represents a player and each internal node represents the winner of the match played between the players represented by its children nodes.
- These tournament trees are also called winner trees because they are being used to record the winner at each level.
- We can also have a loser tree that records the loser at each level.



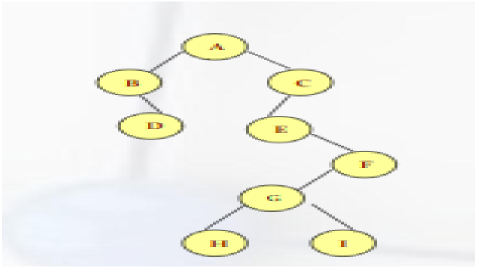
Traversing a Binary Tree

- Traversing a binary tree is the process of visiting each node in the tree exactly once in a systematic way.
- There are three different algorithms for tree traversals, which differ in the order in which the nodes are visited.
- These algorithms are:
  - Pre-order algorithm
  - In-order algorithm
  - Post-order algorithm

Preorder Algorithm

- To traverse a non-empty binary tree in preorder, the following operations are performed recursively at each node. The algorithm starts with the root node of the tree and continues by:
  - Visiting the root node
  - Traversing the left subtree
  - Traversing the right subtree

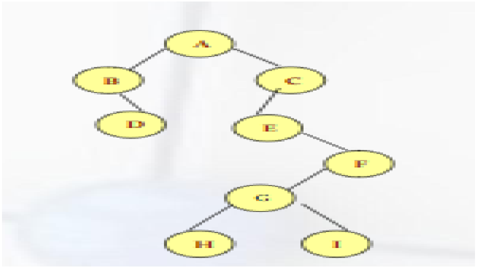
A, B, D, C, E, F, G, H and I



Inorder Algorithm

- To traverse a non-empty binary tree in in-order, the following operations are performed recursively at each node.
- The algorithm starts with the root node of the tree and continues by,
    - Traversing the left subtree
    - Visiting the root node
    - Traversing the right subtree

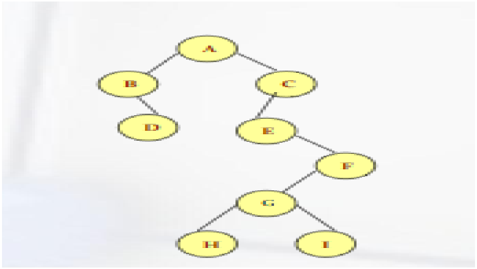
B, D, A, E, H, G, I, F and C



Postorder Algorithm

- To traverse a non-empty binary tree in post-order, the following operations are performed recursively at each node.
- The algorithm starts with the root node of the tree and continues by,
  - Traversing the left subtree
  - Traversing the right subtree
  - Visiting the root node

D, B, H, I, G, F, E, C and A



Applications of Trees

- Trees are used to store simple as well as complex data. Here simple means an int value, char value and complex data (structure).
- Trees are often used for implementing other types of data structures like hash tables, sets, and maps.
- A self-balancing tree, Red-black tree is used in kernel scheduling to preempt massively multi-processor computer operating system use.
- Another variation of tree, B-trees are used to store tree structures on disc. They are used to index a large number of records.
- B-trees are also used for secondary indexes in databases, where the index facilitates a select operation to answer some range criteria.
- Trees are used for compiler construction.
- Trees are also used in database design.
- Trees are used in file system directories.
- Trees are also widely used for information storage and retrieval in symbol tables.

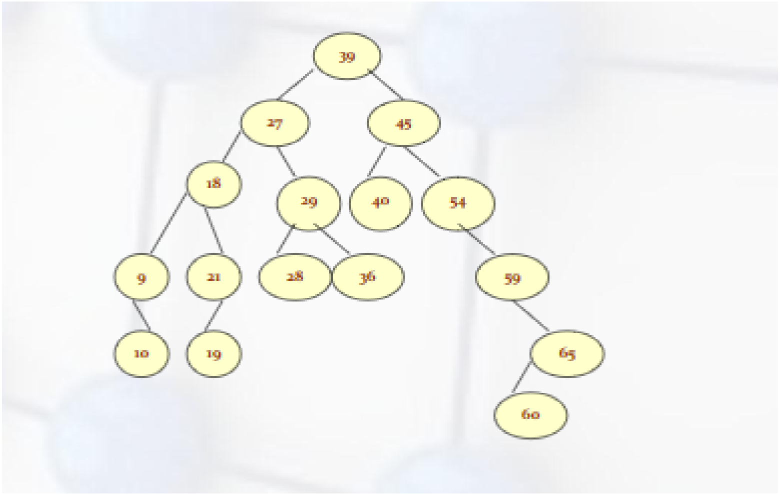
Efficient Binary Trees

- Binary Search Trees
- Threaded Binary Trees
- AVL Trees
- Red Black Trees
- Splay Trees

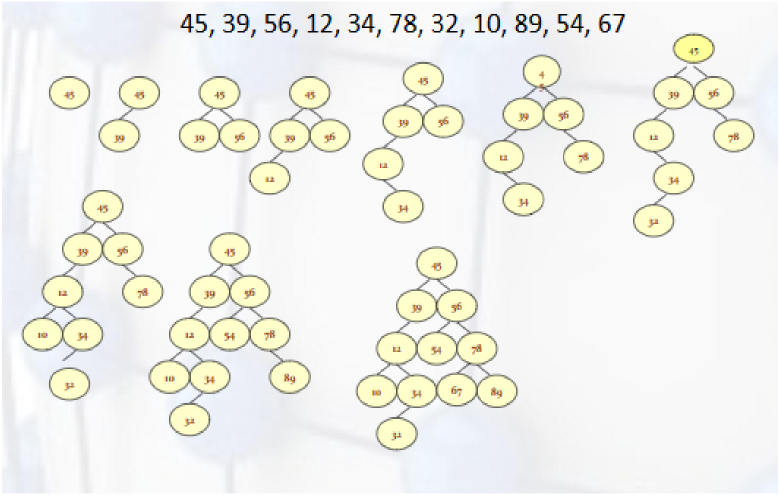
Binary Search Trees

- A binary search tree (BST), also known as an ordered binary tree, is a variant of binary tree in which the nodes are arranged in order.
- In a BST, all nodes in the left sub-tree have a value less than that of the root node.
- Correspondingly, 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 Tree



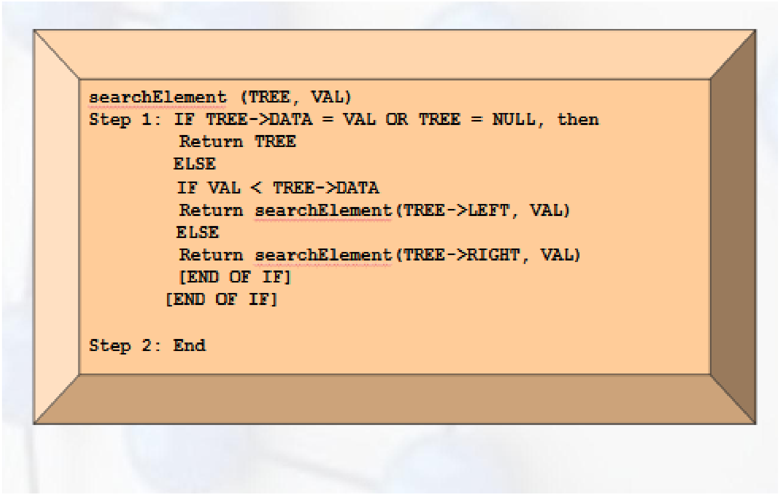
Creating a Binary Search from Given Values



Searching for a Value in a BST

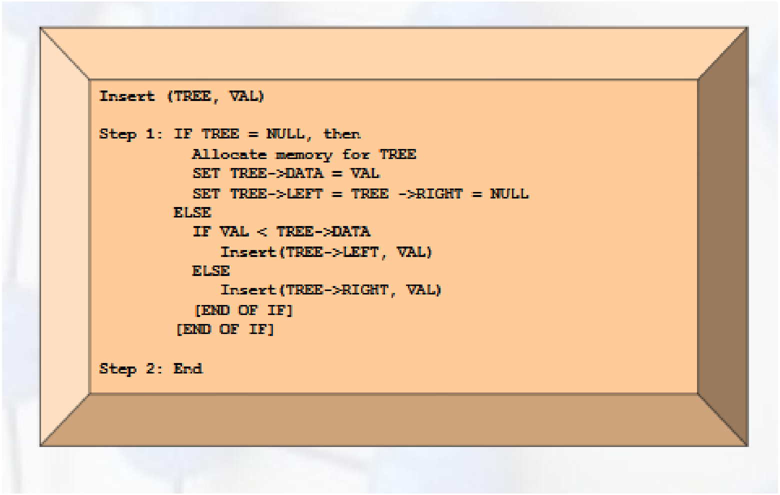
- The search function is used to find whether a given value is present in the tree or not.
- The function first checks if the BST is empty. If it is, then the value we are searching for is not present in the tree, and the search algorithm terminates by displaying an appropriate message.
- However, if there are nodes in the tree then the search function checks to see if the key value of the current node is equal to the value to be searched.
- If not, it checks if the value to be searched for is less than the value of the node, in which case it should be recursively called on the left child node.
- In case the value is greater than the value of the node, it should be recursively called on the right child node.

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
```

Algorithm to Insert a Value in a BST

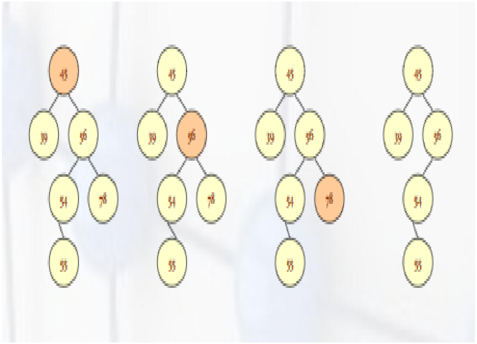


```
Insert (TREE, VAL)

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
```

Deleting a Value from a BST

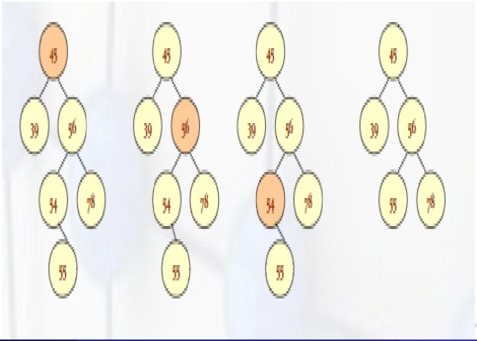
- The delete function deletes a node from the binary search tree.
- However, care should be taken that the properties of the BSTs do not get violated and nodes are not lost in the process.
- The deletion of a node involves any of the three cases.

Case 1: Deleting a node that has no children. For example, deleting node 78 in the tree below.



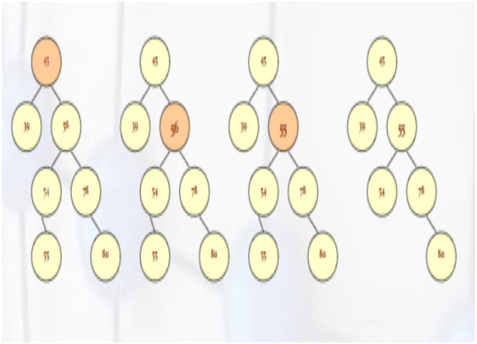
Deleting a Value from a BST

- Case 2: Deleting a node with one child (either left or right).
- To handle the deletion, the nodes child is set to be the child of the nodes parent.
- Now, if the node was the left child of its parent, the nodes child becomes the left child of the nodes parent.
- Correspondingly, if the node was the right child of its parent, the nodes child becomes the right child of the nodes parent.

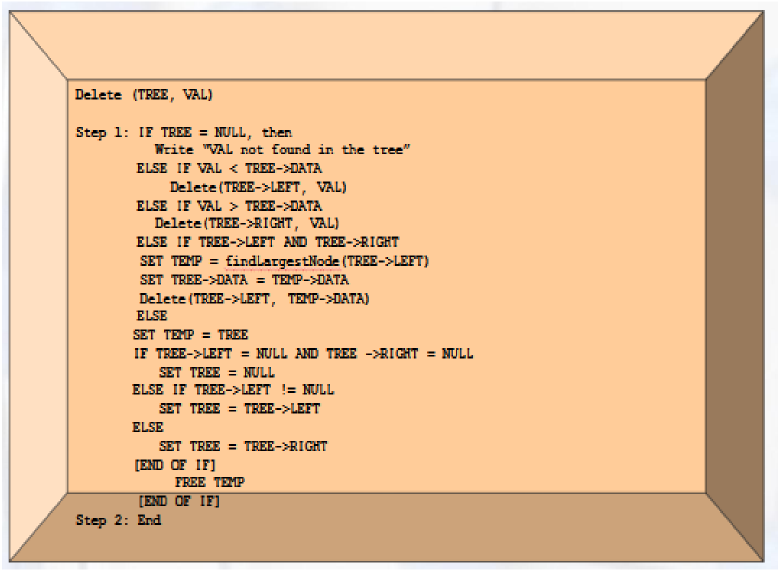


Deleting a Value from a BST

- Case 3: Deleting a node with two children.
- To handle this case of deletion, replace the nodes value with its in-order predecessor (largest value in the left sub-tree) or in-order successor (smallest value in the right sub-tree).
- The in-order predecessor or the successor can then be deleted using any of the above cases.



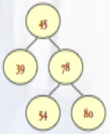
Algorithm to Delete from a BST

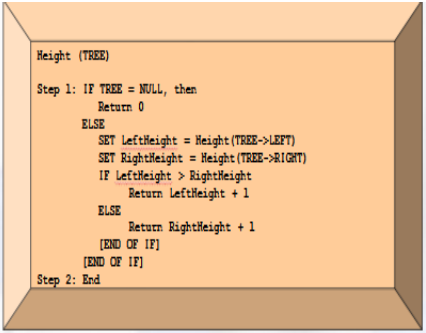


```
Delete (TREE, VAL)

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
```

Determining the Height of a BST

- In order to determine the height of a BST, we will calculate the height of the left and right sub-trees. Whichever height is greater, 1 is added to it.
- Since height of right sub-tree is greater than the height of the left sub-tree, the height of the tree = height (right sub-tree) + 1= 2 + 1 = 3

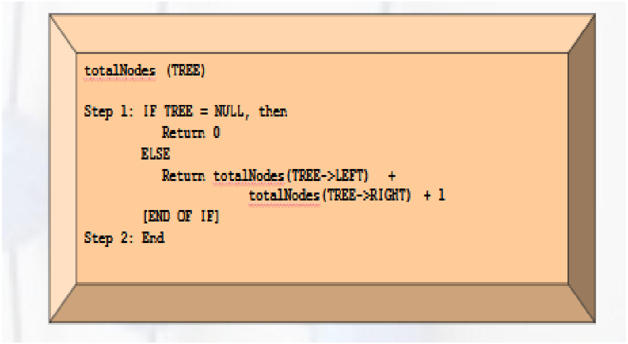




```
Height (TREE)

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 count 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

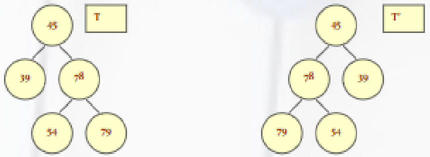


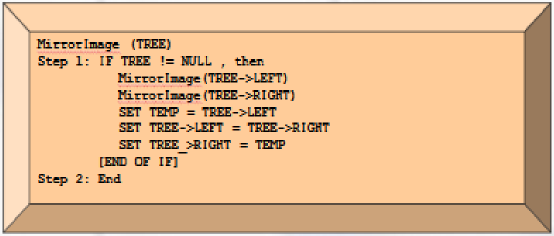
```
totalNodes (TREE)

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.

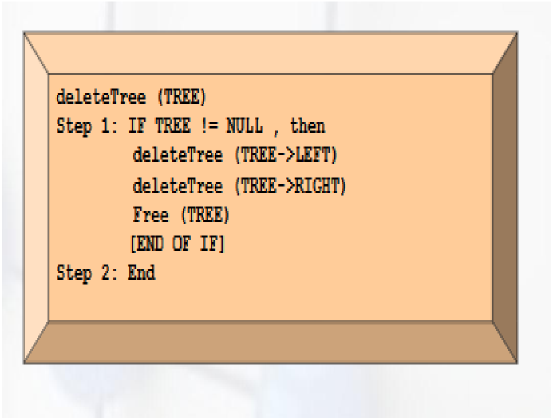




```
MirrorImage (TREE)
Step 1: IF TREE != NULL , then
            MirrorImage(TREE->LEFT)
            MirrorImage(TREE->RIGHT)
            SET TEMP = TREE->LEFT
            SET TREE->LEFT = TREE->RIGHT
            SET TREE_>RIGHT = TEMP
         [END OF IF]
Step 2: End
```

Deleting a BST

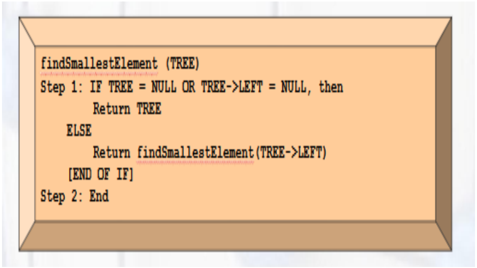
- To delete/remove the entire binary search tree from the memory, we will first delete the elements/nodes in the left sub-tree and then delete the right sub-tree.



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

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.



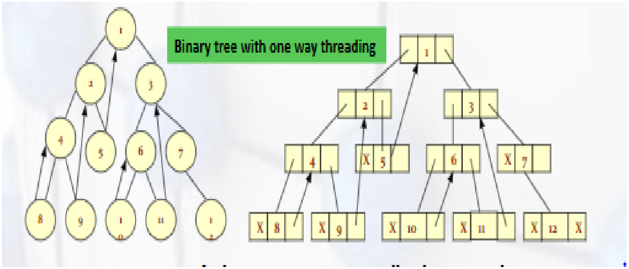
```
findSmallestElement (TREE)
Step 1: IF TREE = NULL OR TREE->LEFT = NULL, then
            Return TREE
    ELSE
            Return findSmallestElement(TREE->LEFT)
    [END OF IF]
Step 2: End
```

Threaded Binary Trees

A threaded binary tree is same as that of a binary tree but with a difference in storing NULL pointers.In the linked representation of a BST, a number of nodes contain a NULL pointer either in their left or right fields or in both. This space that is wasted in storing a NULL pointer can be efficiently used to store some other useful piece of information.For example, the NULL entries can be replaced to store a pointer to the in-order predecessor, or the in-order successor of the node. These special pointers are called threads and binary trees containing threads are called threaded trees. In the linked representation of a threaded binary tree, threads will be denoted using dotted lines.

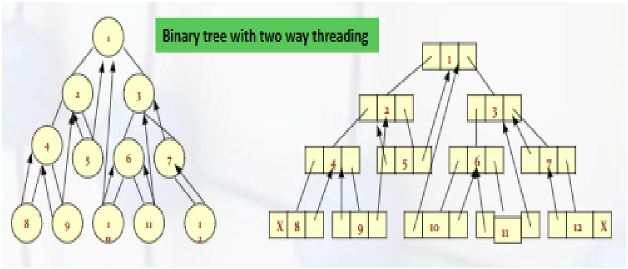
Threaded Binary Trees

- In one way threading, a thread will appear either in the right field or the left field of the node.
- If the thread appears in the left field, then it points to the in-order predecessor of the node. Such a one way threaded tree is called a left threaded binary tree.
- If the thread appears in the right field, then it will point to the in-order successor of the node. Such a one way threaded tree is called a right threaded binary tree.



Threaded Binary Trees

- In a two way threaded tree, also called a doubled threaded tree, threads will appear in both the left and right fields of the node.
- While the left field will point to the in-order predecessor of the node, the right field will point to its successor.
- A two way threaded binary tree is also called a fully threaded binary tree.



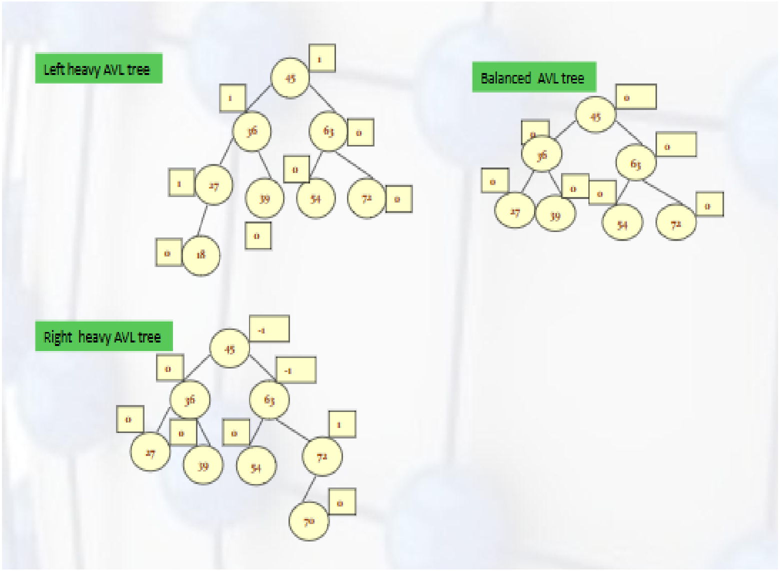
AVL Trees

- AVL tree is a self-balancing binary search tree in which the heights of the two sub-trees of a node may differ by at most one. Because of this property, AVL tree is also known as a height-balanced tree.
- The key advantage of using an AVL tree is that it takes **O(logn)** time to perform search, insertion and deletion operations in average case as well as worst case (because the height of the tree is limited to O(logn)).
- The structure of an AVL tree is same as that of a binary search tree but with a little difference. In its structure, it stores an additional variable called the *BalanceFactor*.

AVL Trees

- The balance factor of a node is calculated by subtracting the height of its right sub-tree from the height of its left sub-tree.
  Balance factor $=$ *Height (left sub-tree)* $-$ *Height (right sub-tree)*
- A binary search tree in which every node has a balance factor of -1, 0 or 1 is said to be height balanced. A node with any other balance factor is considered to be unbalanced and requires rebalancing.
- If the balance factor of a node is 1, then it means that the left sub-tree of the tree is one level higher than that of the right sub-tree. Such a tree is called *Left-heavy tree*.
- If the balance factor of a node is 0, then it means that the height of the left sub-tree is equal to the height of its right sub-tree.
- If the balance factor of a node is -1, then it means that the left sub-tree of the tree is one level lower than that of the right sub-tree. Such a tree is called *Right-heavy tree*.

AVL Trees


Left heavy AVL tree
45
1
1
36
63
0
1
27
39
0
54
72
0
0
18
0
Balanced AVL tree
45
0
0
36
63
0
0
27
39
0
0
54
72
0
Right heavy AVL tree
45
-1
0
36
63
-1
0
27
0
39
0
54
72
1
70
0

Searching for a Node in an AVL Tree

- Searching in an AVL tree is performed exactly the same way as it is performed in a binary search tree.
- Because of the height-balancing of the tree, the search operation takes **O(log n)** time to complete.
- Since the operation does not modify the structure of the tree, no special provisions need to be taken.

Inserting a Node in an AVL Tree

- Since an AVL tree is also a variant of binary search tree, insertion is also done in the same way as it is done in case of a binary search tree.
- Like in binary search tree, the new node is always inserted as the leaf node. But the step of insertion is usually followed by an additional step of rotation.
- Rotation is done to restore the balance of the tree. However, if insertion of the new node does not disturb the balance factor, that is, if the balance factor of every node is still -1, 0 or 1, then rotations are not needed.

Inserting a Node in an AVL Tree

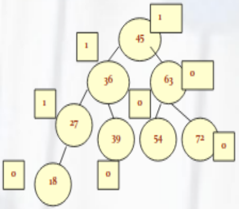
- During insertion, the new node is inserted as the leaf node, so it will always have balance factor equal to zero.
- The nodes whose balance factors will change are those which lie on the path between the root of the tree and the newly inserted node.
- The possible changes which may take place in any node on the path are as follows:
    - Initially the node was either left or right heavy and after insertion has become balanced. Initially the node was balanced and after insertion has become either left or right heavy.
    - Initially the node was heavy (either left or right) and the new node has been inserted in the heavy sub-tree thereby creating an unbalanced sub-tree. Such a node is said to be a critical node.

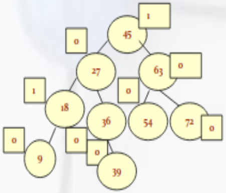
Rotations to Balance AVL Trees

- To perform rotation, our first work is to find the *critical node*. Critical node is the nearest ancestor node on the path from the root to the inserted node whose balance factor is neither -1, 0 nor 1.
- The second task is to determine which type of rotation has to be done.
- There are four types of rebalancing rotations and their application depends on the position of the inserted node with reference to the critical node.
  - *LL rotation*: the new node is inserted in the left sub-tree of the left sub-tree of the critical node
  - *RR rotation*: the new node is inserted in the right sub-tree of the right sub-tree of the critical node
  - *LR rotation*: the new node is inserted in the right sub-tree of the left sub-tree of the critical node
  - *RL rotation*: the new node is inserted in the left sub-tree of the right sub-tree of the critical node

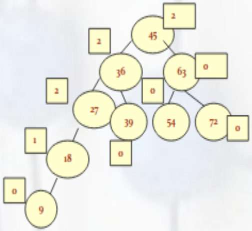
Rotations to Balance AVL Trees

Example: Consider the AVL tree given below and insert 9 into it.



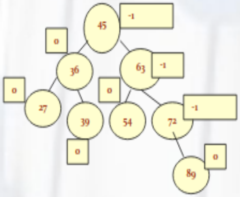


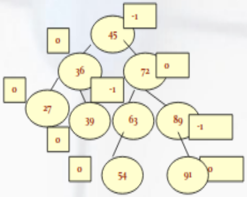
The tree is balanced using LL rotation



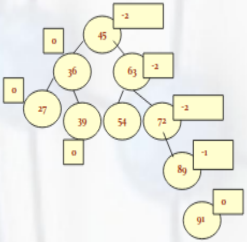
Rotations to Balance AVL Trees

Example: Consider the AVL tree given below and insert 91 into it.





The tree is balanced using RR rotation

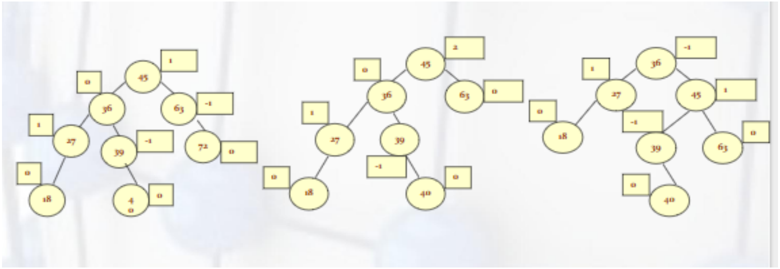


Deleting a Node from an AVL Tree

- Deletion of a node in an AVL tree is similar to that of binary search trees.
- But deletion may disturb the AVLness of the tree, so to re-balance the AVL tree we need to perform rotations.
- There are two classes of rotation that can be performed on an AVL tree after deleting a given node: R rotation and L rotation.
- If the node to be deleted is present in the left sub-tree of the critical node, then L rotation is applied else if node is in the right sub-tree, R rotation is performed.
- Further there are three categories of L and R rotations. The variations of L rotation are: L-1, L0 and L1 rotation. Correspondingly for R rotation, there are R0, R-1 and R1 rotations.

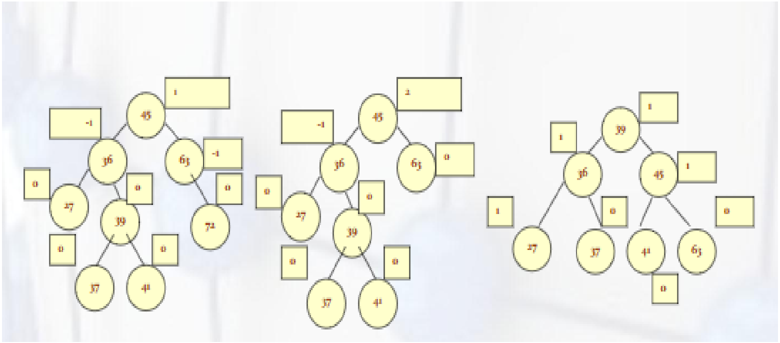
Deleting a Node from an AVL Tree

- R0 Rotation
- Let B be the root of the left or right sub-tree of A (critical node).
- R0 rotation is applied if the balance factor of B is 0. Consider the AVL tree given below and delete 72 from it.



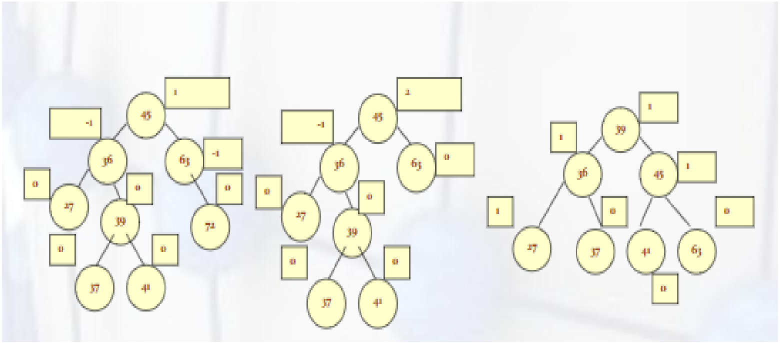
Deleting a Node from an AVL Tree

- R1 Rotation
- Let B be the root of the left or right sub-tree of the critical node.
- R1 rotation is applied if the balance factor of B is 1. Consider the AVL tree given below and delete 72 from it.



Deleting a Node from an AVL Tree

- R-1Rotation
- Let B be the root of the left or right sub-tree of the critical node.
- R-1 rotation is applied if the balance factor of B is -1. Consider the AVL tree given below and delete 72 from it.



1
-1
45
36
63
-1
0
27
39
0
0
72
0
0
37
41
2
-1
45
36
63
0
0
27
39
0
0
0
37
41
1
1
39
36
45
1
1
0
0
27
37
41
63
0