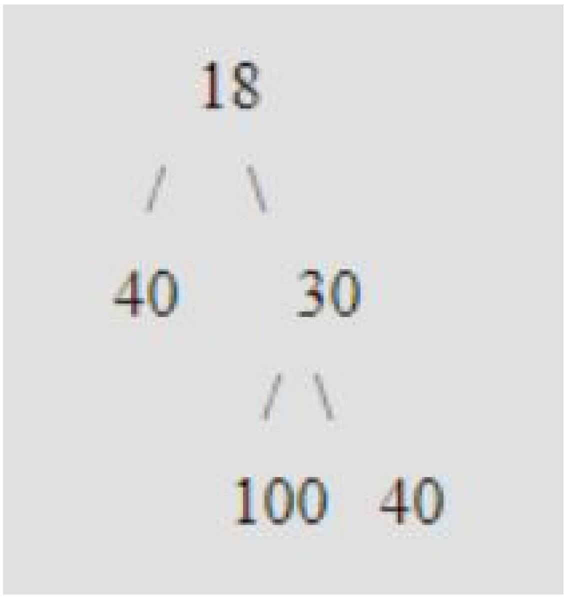
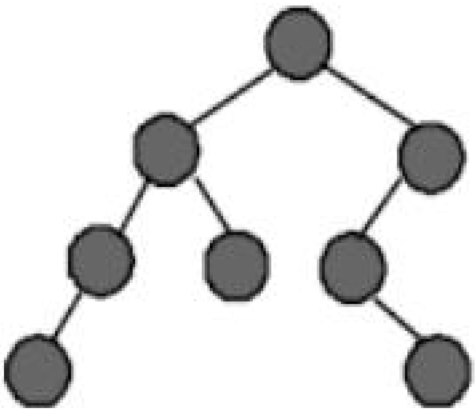
Binary Tree

- If every node in a tree can have at most children, it is called binary tree. Its subtree on its left is called left subtree while on the other is the right subtree.
- A Binary Tree can contain only left or right subtree or no nodes at all, in which case it's a leaf.

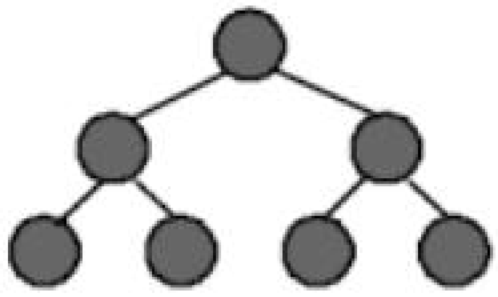
Types of Binary Tree

1. **Full Binary Tree** A Binary Tree Is Full If Every Node Has 0 Or 2 Children. Following Are Examples of a Full Binary Tree. We Can Also Say A Full Binary Tree Is A Binary Tree in Which All Nodes Except Leaves Have Two Children.





2. **Complete Binary Tree:** A Binary Tree is complete Binary Tree if all levels are completely filled except possibly the last level and the last level has all keys as left as possible



3. **Degenerate (or pathological) tree** A Tree where every internal node has one child. Such trees are performance-wise same as linked list.

Binary Tree — Applications

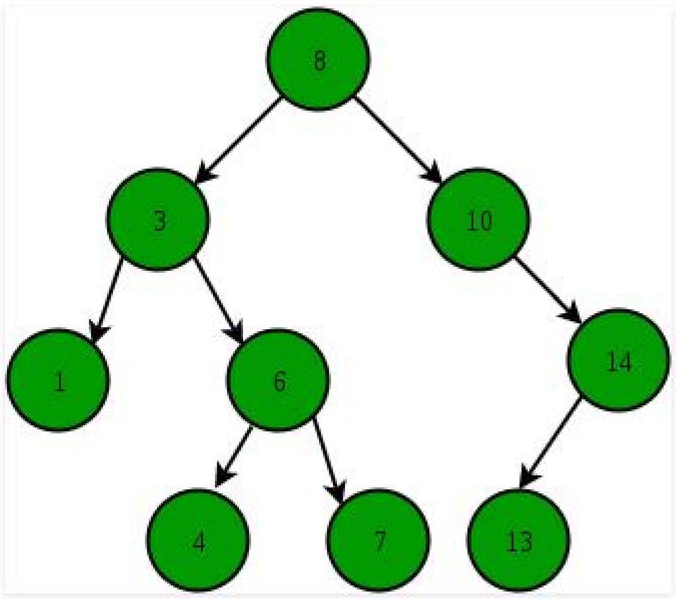
The advantages of using tree are that it is easy to implement and has a good performance for searching.

There are two applications of binary tree:

- Binary Expression Tree
- Binary Search Tree

Binary Search Tree

- Binary Search Trees are trees such that any node has a key which is no more than the key in its right child node and no less than the key in its left child node.



BST Operation — Inserting a Node

Start from the root node, if the node to insert is less than the root, we go to left child, and otherwise we go to the right child of the root. We continue this process (each node is a root for some sub tree) until we find a null pointer (or leaf node) where we cannot go any further. We then insert the node as a left or right child of the leaf node based on node is less or greater than the leaf node. We note that a new node is always inserted as a leaf node.

Insert(N, T) = N if T is empty

= insert(N, T.left) if N < T

= insert(N, T.right) if N > T

BST Operation – Searching for a Node

Searching for a node is similar to inserting a node. We start from root, and then go left or right until we find (or not find the node). A recursive definition of search is as follows. If the node is equal to root, then we return true. If the root is null, then we return false. Otherwise we recursively solve the problem for T.left or T.right, depending on N < T or N > T. A recursive definition is as follows.
Search should return a true or false, depending on the node is found or not.

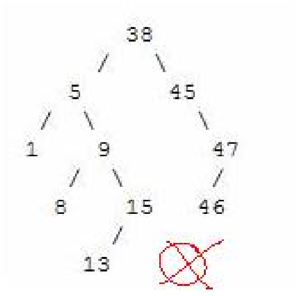
```
Search(N, T) =  false   if T is empty
=  true    if T = N
= search(N, T.left) if N < T
= search(N, T.right) if N > T
```

BST Operation – Deleting a Node

- A BST is a connected structure. That is, all nodes in a tree are connected to some other node. For example, each node has a parent, unless node is the root. Therefore, deleting a node could affect all sub trees of that node. For example, deleting node 5 from the tree could result in losing sub trees that are rooted at 1 and 9.
- Hence, we need to be careful about deleting nodes from a tree. The best way to deal with deletion seems to be considering special cases. What if the node to delete is a leaf node? What if the node is a node with just one child? What if the node is an internal node (with two children? The latter case is the hardest to resolve. But we will find a way to handle this situation as well.

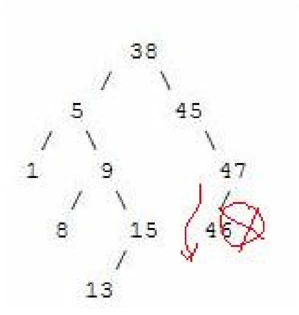
BST Operation — Deleting a Node

- **Case 1 : The node to delete is a leaf node.** This is a very easy case. Just delete the node. We are done.



Case 2 : The node to be deleted is a node with one child

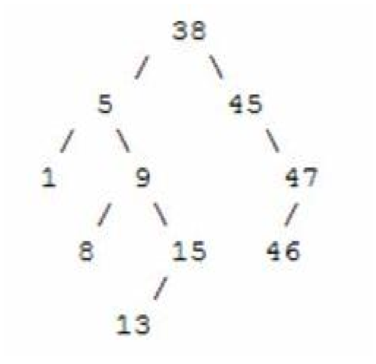
This is also not too bad. If the node to be deleted is a left child of the parent, then we connect the left pointer of the parent (of the deleted node) to the single child. Otherwise if the node to be deleted is a right child of the parent, then we connect the right pointer of the parent (of the deleted node) to single child.



Case 3: The node to deleted is a node with two children

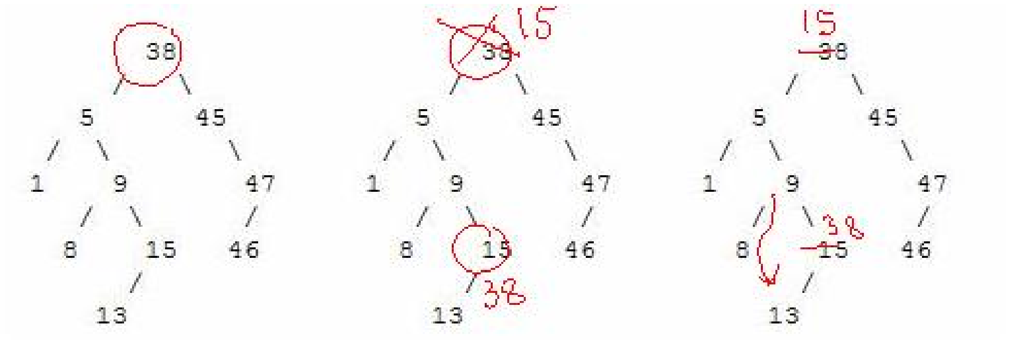
1. Find a replacement node (from leaf node or nodes with one child) for the node to be deleted. Then we swap leaf node or node with one child with the node to be deleted (swap the data) and delete the leaf node or node with one child (case 1 or case
2. Next problem is finding a replacement leaf node for the node to be deleted. We can easily find this as follows. If the node to be deleted is N, then find the largest node in the left sub tree of N or the smallest node in the right sub tree of N. These are two

candidates that can replace the node to be deleted without losing the order property. For example, consider the following tree and suppose we need to delete the root 38.



Case 3: The node to be deleted is a node with two children

3. Then we find the largest node in the left sub tree (15) or smallest node in the right sub tree (45) and replace the root with that node and then delete that node. The following set of images demonstrates this process.

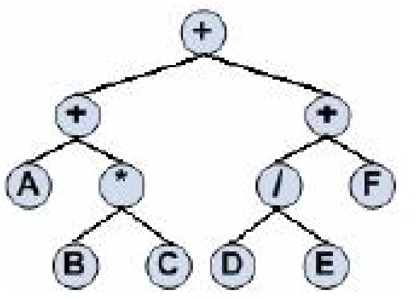


Binary Expression Tree

- An Expression Tree is a binary tree used in representing arithmetic expression; specifically, the combinations of operators, operands and order of evaluation.
- For simplicity, let us assume that the only operations possible are :

plus (+), minus (–), times (*) and divide (/).

- The following diagram shows an example of an expression tree for the mathematical expression:
- (A + B * C) + (D / E +F)



- The leaves of an expression tree will always contain operands. The root of all subtrees (except subtrees with one node), will always represent the operators in the expression.
- In the sample expression tree, the leaf nodes {A, B, C, D, F} represent the set of operands in the expression. The set of root nodes { +, *, +/, +} represent all the operators in the expression.
- In any given mathematical expression, there are always at most two (2) operands to one (1) operator. Since all operators are root nodes and all operands are leaf nodes, this means that an expression tree is in fact a binary tree.

Binary Tree Traversal

- Traversing a tree means to visit all the nodes of the tree in order. This process is not commonly used as finding, inserting, deleting nodes because traversal is not particularly fast but another operation you can perform in binary tree. However, traversing is useful in some circumstances and simpler than deleting of nodes.

There are three different methods in traversing a tree.

1. Inorder

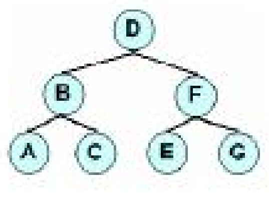
2. Preorder
3. Postorder

✓ Among the three methods, Inorder is the commonly used for binary search trees.
✓ The other two methods are useful when writing programs that parse or analyze algebraic expressions.
✓ An infix notation when traverse using InOrder traversal generates the correct inorder sequence.
✓ While Preorder method generates the prefix notation, and the Postorder method generates the postfix notation of the binary tree.

When a tree is empty, it is “traversed” by doing nothing. The following is a recursive definition of the steps needed for each of the tree types of traversal.

InOrder Traversal Steps

1. If the node has a left subtree:
   - Traverse the left subtree in preorder (recursive call). Once completed, proceed to step 2.
   - Otherwise, proceed to step 2.
2. Read the root node. If the node has a right subtree:
   - Traverse the right subtree in preorder (recursive call).



For simpler algorithm:

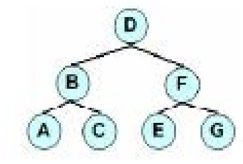
1. Traverse the left subtree
2. Visit the root node
3. Traverse the right subtree

The nodes of the tree will be visited in the order.

A B C D E F G

Preorder Traversal

1. Read the root node.
2. If the node has a left subtree:
   - Traverse the left subtree in preorder (recursive call).
   - Once completed, proceed to step 3.
   - Otherwise, proceed to step 3.
3. If the node has a right subtree:
   - Traverse the right subtree in preorder (recursive call**).**



***For simpler algorithm*:**

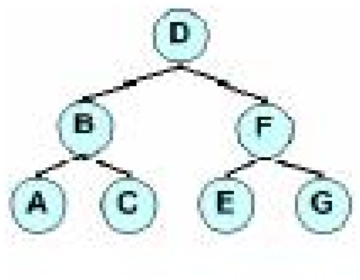
1. Start at the root node.
2. Traverse the left subtree.
3. Traverse the right subtree.

The nodes of the tree will be visited in order.

D B A C F E G

Postorder Traversal

1. If the node has a left subtree:
   - Traverse the left subtree in preorder, once completed proceed to step 2.
   - Otherwise, proceed to step2.
2. If the node has a right subtree:
   - Traverse the right subtree in preorder, once completed proceed to step3.
   - Otherwise, proceed to step3
3. Read the root



For simple algorithm:

1. Traverse the left subtree.
2. Traverse the right subtree.
3. Visit the root node.

The nodes of the tree will be visited in the order:

A C B E G F D

Evaluation/Generalization

- Tree consist of nodes (circles) connected by edges (lines).
- The root is the topmost node in a tree; it has no parent.
- In a binary tree, a node has at most two children.

- In a binary search tree, all nodes that are left descendants of node A have a key values less than A; all the nodes that are A’s right descendant have key values greater than or equal to A.
- Nodes represent the data-objects being stored in the tree.
- Edges are most commonly represented in a program by references to a node’s children and sometimes to its parent.
- Traversing a tree means visiting all its nodes in some order.
- The simplest traversals are inorder, preorder and postorder.
- Insertion involves finding the place to insert the new node and then changing a child field in its new parent to refer to it.
- An inorder traversal visits node in order of ascending keys.
- Preorder and postorder traversals are useful for parsing algebraic expression.

In binary expression trees, operators are the internal nodes while operands are the leaf nodes.