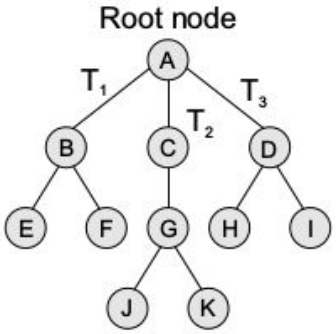
Tree

Tree is a data structure which is recursively defined as a collection of nodes where the starting node is called as root. Each node including root consists of a data value and a variable number of pointers referencing child nodes also known as successors.

There can be zero or more child nodes but the parent must be one.

Below is a tree with root node A. All the remaining nodes of the tree can be divided into non-empty sets, also known as sub-trees of the root (e.g., T1, T2 and T3).



Leaf node: A node which has no children. For example, nodes E, F, J, K, H and I.

Nodes other than the leaf nodes are said to be internal nodes. E.g., A, B, C etc.

Path is a sequence of nodes. Eg. Path from A to K is given as: A, C, G and K.

Length of a path = number of nodes in the path - 1. Eg. len(path from A to K) = 3

Depth of a node = length of the path from the root to the node
E.g.: depth(I) = 2

Height of a tree: Maximum depth of any node within the tree.
E.g., height of the given tree is 3

Height of a tree which has only one (root) node will be zero.

If a path exists from a node a to b then: (i) a is ancestor of b and (ii) b is descendent of a.
A node will be both an ancestor and a descendant of itself. However, we can add the adjective strict to exclude equality: a is a strict descendent of b if a is a descendant of b but a ≠ b.

Root node is an ancestor of all nodes.

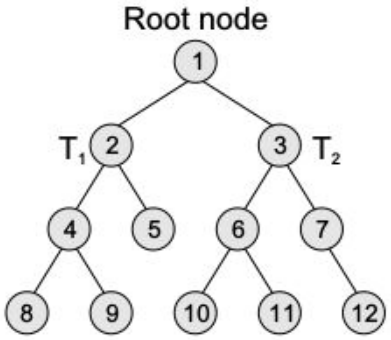
In the above figure, descendents of C are C, G, J and K. Ancestors of C are C and A. In some books, definitions of ancestors and descendants exclude this equality.

Nodes with the same parent are known as siblings. E.g., B, C and D are siblings.

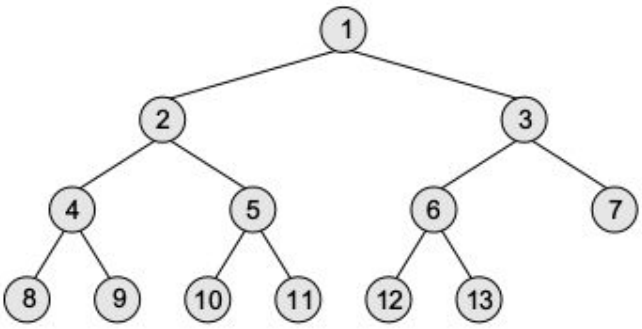
Binary Tree

Arbitrary number of children in general trees is often unnecessary and many real-life trees are restricted to two branches.
There are also issues with general trees including some implementation issues also.

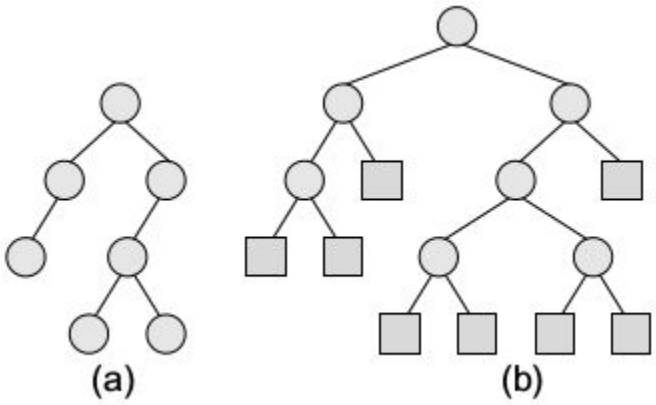
A binary tree is a restriction where each node has at most two children. This restriction allows us to label the children as left and right subtrees. In the below figure, T1 and T2 are left and right subtrees of node 1.



Complete binary tree: It is a binary tree in which all levels except the last one are completely filled and all nodes appear as far left as possible.
Eg., In the below figure, a complete binary is depicted.

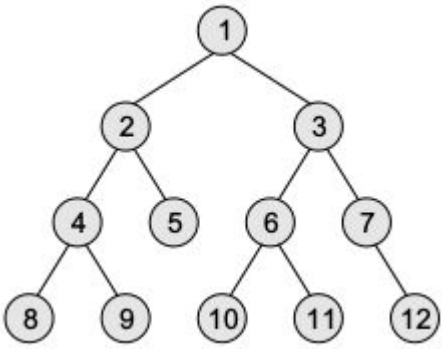


Extended binary tree: A binary tree where each node has either two children or no children. We can convert a binary into an extended binary tree. To do this, we need to replace every empty subtree with a new node called an external node. In the below figure(b), external nodes are represented using rectangles.



Representation of binary trees:

(i) Linked representation: Similar to linked list, we can use struct node to declare a tree node structure with three elements i.e., node data, left pointer and right pointer. In memory, a tree can be looked like as follows. Our root pointer will point to the first node of the tree. If it is null then the tree is said to be empty.

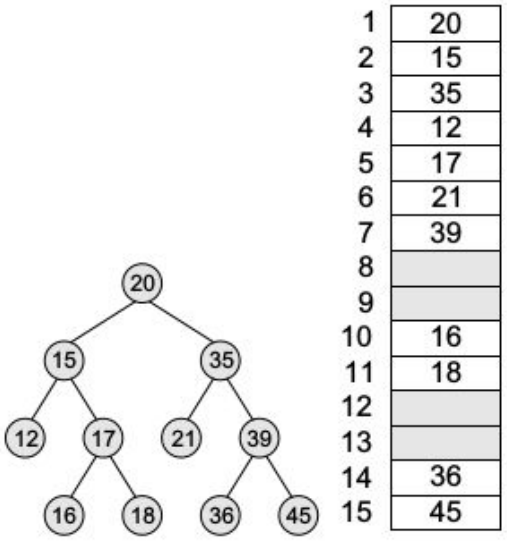


ROOT 3

AVAIL 15

	LEFT	DATA	RIGHT
1	–1	8	–1
2	–1	10	–1
3	5	1	8
4			
5	9	2	14
6			
7			
8	20	3	11
9	1	4	12
10			
11	–1	7	18
12	–1	9	–1
13			
14	–1	5	–1
15			
16	–1	11	–1
17			
18	–1	12	–1
19			
20	2	6	16

(ii) Sequential (array) representation:

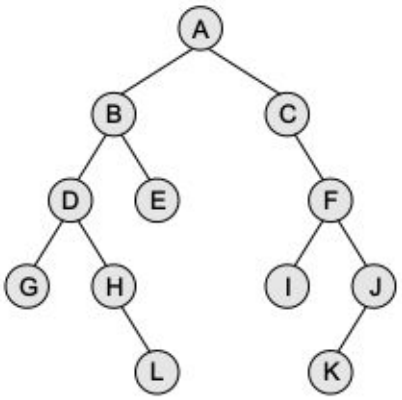


Simplest technique and requires a single or 1-dimensional array. But, it has a disadvantage as it requires a lot of space. An example is as shown in the figure above. In the given representation, the root node is present at the first location (say, 1 in this example) of the array. Left child of a node located at k will be stored at array[2k] and the right child of the same node will be stored at array[2k+1]. For example array[2] will contain the left child of root and array[2+1] = array[3] will store the right child of the root.
If the height of the tree is h then the maximum number of elements that can be stored in the array will be 2^h-1. If the tree is not complete and some elements are not present in the tree (e.g., children of node 12) then space will be reserved for these elements.

Traversing a Binary Tree:
(i) Pre-order (ii) In-order (iii) Post-order

(i)Pre-order: Also known as depth-first traversal. Each node recursively performs the following operations.

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



Pre-order traversal of the above tree is A, B, D, G, H, L, E, C, F, I, J, and K

(ii) In-order: At each node, following operations are performed recursively.

1: Traverse the left subtree

2: Visit the root node
3: Traverse the right subtree

In-order traversal of the above tree is G, D, H, L, B, E, A, C, I, F, K, and J.

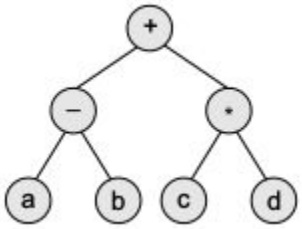
(iii) Post-order: Following operations are recursively performed at every node of the tree.
1: Traverse the left subtree
2: Traverse the right subtree
3: Visit the root node

Post-order traversal of the above tree will be G, L, H, D, E, B, I, K, J, F, C, and A

(iii) Level-order: Also known as breadth-first traversal. All nodes are traversed level-by-level. Before going to the next level, all nodes of the present level must be traversed.
E.g.: Level-order traversal of the above tree is A, B, C, D, E, F, G, H, I, J, L and K.

Applications of Binary Tree:

(i) Expression Tree: We can use binary trees in order to represent/store algebraic expressions.
Eg. Given expression (a – b) + (c * d) can be represented using the following binary tree.



We can achieve prefix expression of any input infix expression by first converting the input infix expression into a binary tree and then by following its pre-order traversal.
E.g.: Pre-order traversal of the above tree is + – a b * c d (i.e., equivalent prefix expression).

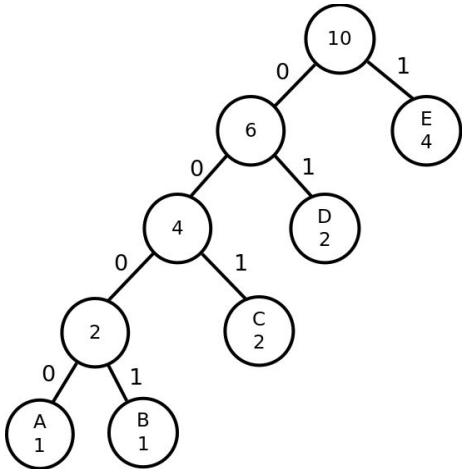
Similarly, post-order traversal can be used for extracting postfix notation from an expression tree. E.g.: Post-order traversal of the above tree is a b - c d * + (i.e., equivalent postfix expression).

(ii) Huffman Encoding: A lossless data compression technique developed by David A. Huffman. It is a variable length encoding and has significant advantages over the fixed-length encoding techniques. The main idea behind this technique is that it encodes the most common characters using shorter strings of bits than those used for less common source characters.
We can use priority queue and binary tree in order to implement huffman encoding.

Algorithm
1: Create a leaf node for each character. Add the character and its weight/frequency of occurrence to the priority queue. We consider a character with lowest weight as the highest priority.
2: Repeat Steps 3 to 5 while the total number of nodes in the queue is greater than 1.
3: Remove two nodes that have the lowest weight/highest priority in the queue.
4: Create a new internal node by merging these two nodes as children and with weight equal to the sum of the two nodes' weights.

Step 5: Add the newly created node to the queue.

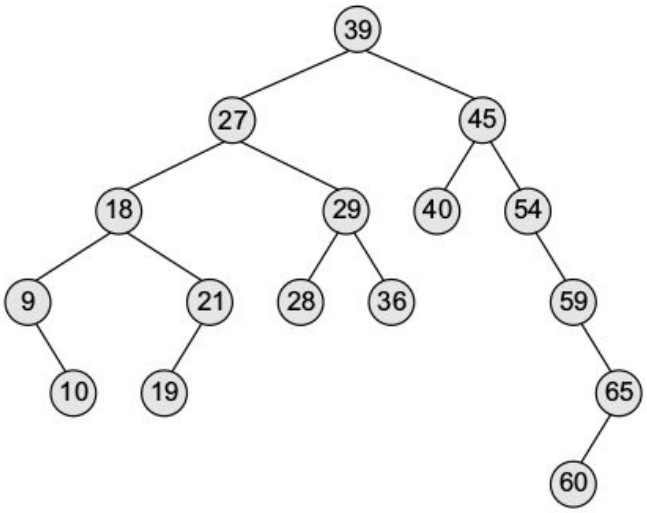
Example-
Character: A B C D E
Frequency: 1 1 2 2 4



Output: Symbols used for A is 0000,
B is 0001, C is 001, D is 01, and E is 1.

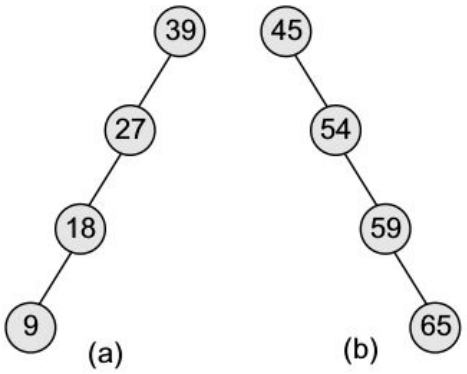
Binary Search Tree

A binary search tree which is also known as an ordered binary tree, is a variant of binary trees in which nodes are arranged in a particular order. All nodes in the left subtree have a value less than that of the root node and all the nodes in the right subtree have a value either equal or greater than that of the root node. This rule is applicable to every subtree of the tree.



Above figure shows an example of the binary search tree in which root node is 39 and all the nodes in its left subtree (i.e., from 9 to 36) are less than 39 and all the node in its right subtree (i.e., from 40 to 65) are greater than or equal to 39.

Binary search trees can be considered as efficient data structures especially when compared with sorted arrays and linked lists. For insertion, deletion and searching of an element, it gives O(h) time complexity where h is the height of the binary search tree. If we somehow managed the height of the tree then our time complexity can become O(log n). In the worst case as shown in figure below, time complexity can be O(n).



Searching a Node: This operation is a trivial operation as the data of the binary search tree is already sorted. All you need to perform the operations similar to binary search. In the first, you check the value of the root node. If your root is null or in other words your tree is empty then you will return directly. Otherwise, if the value of the root is equal to the key then you will return the location of the root. If the value of the root is greater then the key value then you will perform your search in the left subtree. In the remaining case, you will go for the right subtree. If you reach at some leaf node whose value is not equal to the key value then you can conclude that the element you are trying to search in the tree is not available in the tree.

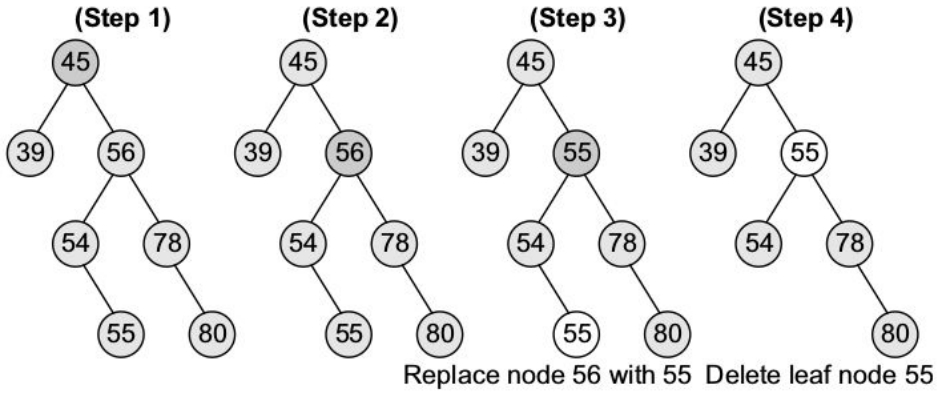
Inserting a node: This operation starts with searching a correct position for the node which you want to insert in the binary search tree. Following is the algorithm which can be used to insert a node.

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

Deleting a node: This operation contains the following three cases.

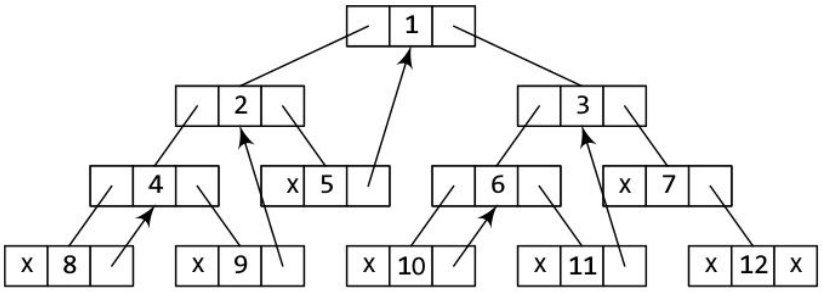
(a) When the node to be deleted has no children: It is trivial. We can simply delete such nodes.
(b) When the node to be deleted has one child: To handle this case, we replace the node which is to be deleted with its child.
(c) When the node to be deleted has two child nodes: In this case, we first replace the node's value with its in-order predecessor or its in-order successor. The in-order predecessor or the in-order successor can then be deleted by following any of the

above cases. Below figure shows an example of deletion of a node which has two children.



Threaded Binary Tree: In the linked representation of a tree, many nodes may contain a NULL pointer, can be either in left or right field or in both. This space can be efficiently utilized to store a pointer to in-order predecessor or in-order successor of the node. These special pointers are known as threads and binary trees containing these threads are called threaded binary trees.

Threading can be done in two ways i.e., one-way threading and two-way threading. In one-way threading, either left field or right field is used as thread. Whereas in two-way threading, both left and right fields contain the threads for in-order predecessor and successor of the node.



An example of a one-way threaded binary tree where the right field is used to store in-order successor of the node is presented in the above figure. Since, the rightmost leaf node (with value 12) has no in-order successor hence it contains null value in its right field. Advantage: Enables linear traversal of nodes of the binary tree which in normal situations can not be done without utilizing stacks. It also enables to locate the parent of the node without utilizing explicit parent pointer. In a two-way threaded binary tree, we can perform forward as well as backward traversal of nodes of the tree also.