**MY DSA DOCUMENT**

**BINARY SEARCH TREE**

A tree who is having at most 2 child ks called binary.

Child is named as left and right child



A Binary Tree node contains following parts.

1. Data
2. Pointer to left child
3. Pointer to right child

**Trees:** Unlike Arrays, Linked Lists, Stack and queues, which are linear data structures, trees are hierarchical data structures.

**Tree Vocabulary:**The topmost node is called root of the tree. The elements that are directly under an element are called its children. The element directly above something is called its parent. For example, ‘a’ is a child of ‘f’, and ‘f’ is the parent of ‘a’. Finally, elements with no children are called leaves.

tree

----

j <-- root

/ \

f k

/ \ \

a h z <-- leaves

**Why Trees?**  
**1.** One reason to use trees might be because you want to store information that naturally forms a hierarchy. For example, the file system on a computer:

file system

-----------

/ <-- root

/ \

... home

/ \

ugrad course

/ / | \

... cs101 cs112 cs113

**2.** Trees (with some ordering e.g., BST) provide moderate access/search (quicker than Linked List and slower than arrays).  
**3.** Trees provide moderate insertion/deletion (quicker than Arrays and slower than Unordered Linked Lists).  
**4.** Like Linked Lists and unlike Arrays, Trees don’t have an upper limit on number of nodes as nodes are linked using pointers.

**Main applications of trees include:**  
**1.** Manipulate hierarchical data.  
**2.** Make information easy to search (see tree traversal).  
**3.** Manipulate sorted lists of data.  
**4.** As a workflow for compositing digital images for visual effects.  
**5.**Router algorithms  
**6.**Form of a multi-stage decision-making (see business chess).

**Binary Tree Representation in C:**A tree is represented by a pointer to the topmost node in tree. If the tree is empty, then value of root is NULL.  
A Tree node contains following parts.  
1. Data  
2. Pointer to left child  
3. Pointer to right child

In C, we can represent a tree node using structures. Below is an example of a tree node with an integer data.

struct node

{

  int data;

  struct node \*left;

  struct node \*right;

};

**First Simple Tree in C**  
Let us create a simple tree with 4 nodes in C. The created tree would be as following.

tree

----

1 <-- root

/ \

2 3

/

4

CODE:

<https://www.geeksforgeeks.org/binary-tree-set-1-introduction/>

# Binary Tree | Set 2 (Properties)

Last Updated: 27-10-2020

We have discussed [Introduction to Binary Tree in set 1](http://quiz.geeksforgeeks.org/binary-tree-set-1-introduction/). In this post, the properties of a binary tree are discussed.

***1) The maximum number of nodes at level ‘l’ of a binary tree is 2l***.   
Here level is the number of nodes on the path from the root to the node (including root and node). Level of the root is 0.   
This can be proved by induction.   
For root, l = 0, number of nodes = 20 = 1   
Assume that the maximum number of nodes on level ‘l’ is 2l   
Since in Binary tree every node has at most 2 children, next level would have twice nodes, i.e. 2 \* 2l

***2) The*** ***Maximum number of nodes in a binary tree of height ‘h’ is 2h – 1***.   
Here the height of a tree is the maximum number of nodes on the root to leaf path. Height of a tree with a single node is considered as 1.   
This result can be derived from point 2 above. A tree has maximum nodes if all levels have maximum nodes. So maximum number of nodes in a binary tree of height h is 1 + 2 + 4 + .. + 2h-1. This is a simple geometric series with h terms and sum of this series is 2h – 1.   
In some books, the height of the root is considered as 0. In this convention, the above formula becomes 2h+1 – 1

***3) In a Binary Tree with N nodes, minimum possible height or***the ***minimum number of levels is? Log2(N+1)***

***4) A Binary Tree with L leaves has at least? Log2L? + 1   levels***

***5) In Binary tree where every node has 0 or 2 children, the*** ***number of leaf nodes is always one more than nodes with two children***.

# Types of Binary Tree

**Full Binary Tree**

18

/ \

15 30

/ \ / \

40 50 100 40

18

/ \

15 20

/ \

40 50

/ \

30 50

18

/ \

40 30

/ \

100 40

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

The following are examples of Complete Binary Trees 

18

/ \

15 30

/ \ / \

40 50 100 40

18

/ \

15 30

/ \ / \

40 50 100 40

/ \ /

8 7 9

**Perfect Binary Tree** A Binary tree is a Perfect Binary Tree in which all the internal nodes have two children and all leaf nodes are at the same level.

18

/ \

15 30

/ \ / \

40 50 100 40

18

/ \

15 30

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

10

/

20

\

30

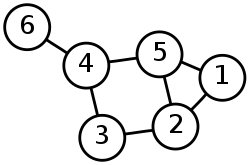
\

40

# Handshaking lemma

From Wikipedia, the free encyclopedia

[Jump to navigation](https://en.wikipedia.org/wiki/Handshaking_lemma#mw-head)[Jump to search](https://en.wikipedia.org/wiki/Handshaking_lemma#searchInput)

[](https://en.wikipedia.org/wiki/File:6n-graf.svg)

In this graph, an even number of vertices (the four vertices numbered 2, 4, 5, and 6) have odd degrees. The sum of degrees of all six vertices is 2 + 3 + 2 + 3 + 3 + 1 = 14, twice the number of edges.

In [graph theory](https://en.wikipedia.org/wiki/Graph_theory), a branch of mathematics, the **handshaking lemma** is the statement that every finite [undirected graph](https://en.wikipedia.org/wiki/Undirected_graph) has an even number of vertices with odd [degree](https://en.wikipedia.org/wiki/Degree_(graph_theory)) (the number of edges touching the vertex). In more colloquial terms, in a party of people some of whom shake hands, an even number of people must have shaken an odd number of other people's hands.

The handshaking lemma is a consequence of the **degree sum formula** (also sometimes called the **handshaking lemma**),

{\displaystyle \sum \_{v\in V}\deg v=2|E|}

**How many different Unlabeled Binary Trees can be there with n nodes?**

For n = 1, there is only one tree

o

For n = 2, there are two trees

o o

/ \

o o

For n = 3, there are five trees

o o o o o

/ \ / \ / \

o o o o o o

/ \ \ /

o o o o

For example, let T(n) be count for n nodes.

T(0) = 1 [There is only 1 empty tree]

T(1) = 1

T(2) = 2

T(3) = T(0)\*T(2) + T(1)\*T(1) + T(2)\*T(0) = 1\*2 + 1\*1 + 2\*1 = 5

T(4) = T(0)\*T(3) + T(1)\*T(2) + T(2)\*T(1) + T(3)\*T(0)

= 1\*5 + 1\*2 + 2\*1 + 5\*1

= 14

The above pattern basically represents [n’th Catalan Numbers](https://www.geeksforgeeks.org/program-nth-catalan-number/). First few catalan numbers are 1 1 2 5 14 42 132 429 1430 4862,…



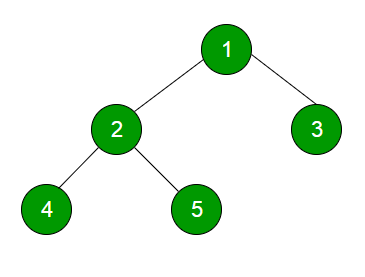
Here,  
T(i-1) represents number of nodes on the left-sub-tree  
T(n−i-1) represents number of nodes on the right-sub-tree

n’th Catalan Number can also be evaluated using direct formula.

T(n) = (2n)! / (n+1)!n!

**Level order binary tree traversal or BFS**

It is BFS for the tree

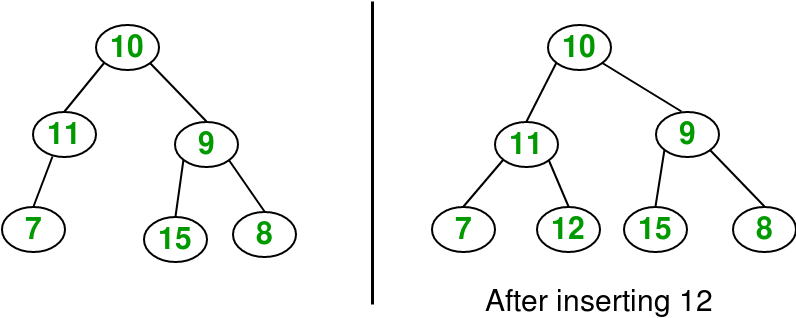


Code:

<https://www.geeksforgeeks.org/level-order-tree-traversal/>

**Time Complexity:** O(n) where n is number of nodes in the binary tree   
**Space Complexity:** O(n) where n is number of nodes in the binary tree

**Insertion in BST in level order**



**CODE:**

[**https://www.geeksforgeeks.org/insertion-in-a-binary-tree-in-level-order/**](https://www.geeksforgeeks.org/insertion-in-a-binary-tree-in-level-order/)

**Deletion in BT**

Given a binary tree, delete a node from it by making sure that tree shrinks from the bottom (i.e. the deleted node is replaced by bottom most and rightmost node). This different from [BST deletion](https://www.geeksforgeeks.org/binary-search-tree-set-2-delete/). Here we do not have any order among elements, so we replace with last element.  
Examples :

Delete 10 in below tree

10

/ \

20 30

Output :

30

/

20

Delete 20 in below tree

10

/ \

20 30

\

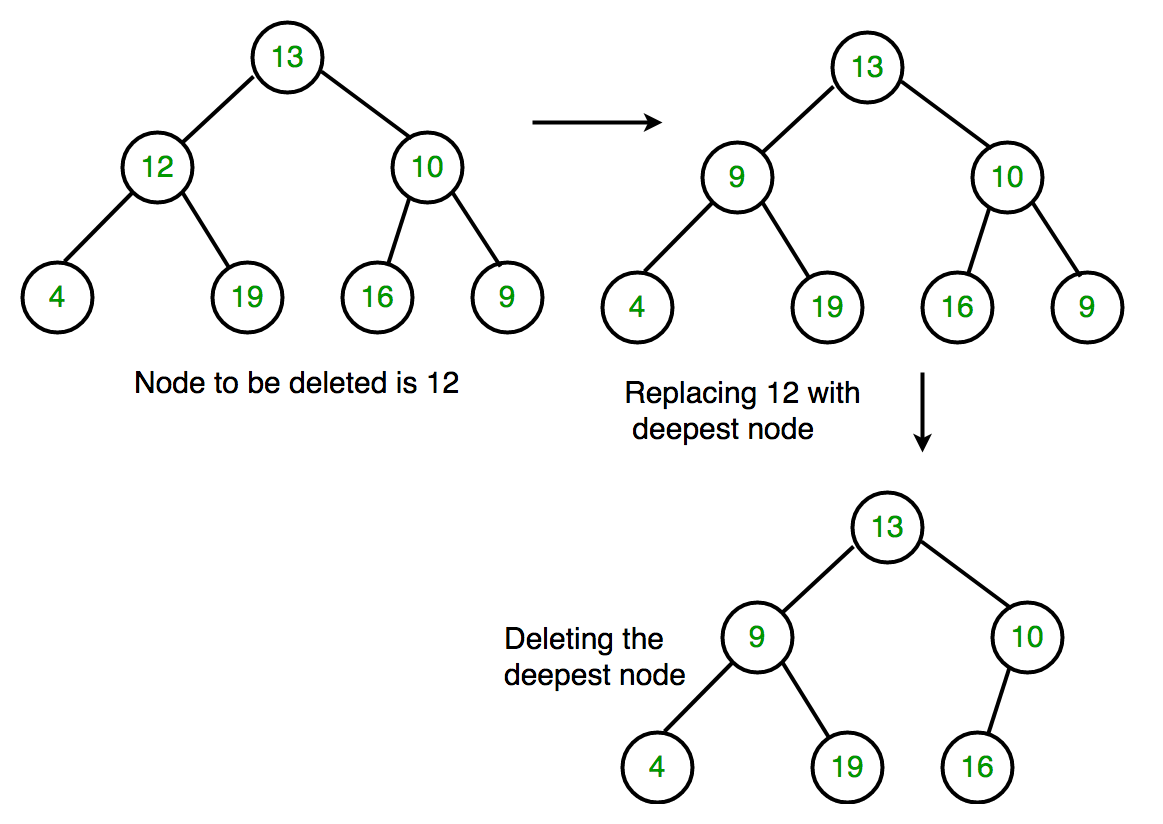
40

Output :

10

/ \

40 30



**CODE:**

[**https://www.geeksforgeeks.org/deletion-binary-tree/**](https://www.geeksforgeeks.org/deletion-binary-tree/)

**BFS VS DFS FOR BT**

A tree can be travelled in 2 ways:

* Breath first search-

\*Level order traversal

* Depth first search-

\*Inorder Traversal(Left-Root-Right)

\*Preorder Traversal(Root-Left-Right)

\*Postorder Traversal(Left-Right-Root)



BFS-Level order-1 2 3 4 5

DFS

* Inorder(L-C-R)-4 2 5 1 3
* Preorder-(C-L-R)-1 2 4 5 3
* Postorder(L-R-C)-4 5 2 3 1

All four traversals require **O(n)** time as they visit every node exactly once.

There is difference in terms of extra space required.

1. Extra Space required for Level Order Traversal is O(w) where w is maximum width of Binary Tree. In level order traversal, queue one by one stores nodes of different level.
2. Extra Space required for Depth First Traversals is O(h) where h is maximum height of Binary Tree. In Depth First Traversals, stack (or function call stack) stores all ancestors of a node.
3. Extra Space can be one factor (Explained above)
4. Depth First Traversals are typically recursive and recursive code requires function call overheads.
5. The most important points is, BFS starts visiting nodes from root while DFS starts visiting nodes from leaves. So if our problem is to search something that is more likely to closer to root, we would prefer BFS. And if the target node is close to a leaf, we would prefer DFS.

**Representation:**

**Array implementation**

` A(0)

/ \

B(1) C(2)

/ \ \

D(3) E(4) F(6)

OR,

A(1)

/ \

B(2) C(3)

/ \ \

D(4) E(5) F(7)

For first case(0—n-1),   
if (say)father=p;   
then left\_son=(2\*p)+1;   
and right\_son=(2\*p)+2;  
For second case(1—n),   
if (say)father=p;   
then left\_son=(2\*p);   
and right\_son=(2\*p)+1;   
where father, left\_son and right\_son are the values of indices of the array.

**Code:**

[**https://www.geeksforgeeks.org/binary-tree-array-implementation/**](https://www.geeksforgeeks.org/binary-tree-array-implementation/)

# AVL tree

a [self-balancing binary search tree](https://en.wikipedia.org/wiki/Self-balancing_binary_search_tree). It was the first such [data structure](https://en.wikipedia.org/wiki/Data_structure) to be invented.[[2]](https://en.wikipedia.org/wiki/AVL_tree#cite_note-2) In an AVL tree, the [heights](https://en.wikipedia.org/wiki/Tree_height) of the two [child](https://en.wikipedia.org/wiki/Child_node) subtrees of any node differ by at most one; if at any time they differ by more than one, rebalancing is done to restore this property. Lookup, insertion, and deletion all take [O](https://en.wikipedia.org/wiki/Big_O_notation)(log *n*) time in both the average and worst cases

**Applications**

Unlike Array and Linked List, which are linear data structures, tree is hierarchical (or non-linear) data structure.

1. One reason to use trees might be because you want to store information that naturally forms a hierarchy. For example, the file system on a computer:

file system  
———–

/ <-- root

/ \

... home

/ \

ugrad course

/ / | \

... cs101 cs112 cs113

1. If we organize keys in form of a tree (with some ordering e.g., BST), we can search for a given key in moderate time (quicker than Linked List and slower than arrays). [Self-balancing search trees](http://en.wikipedia.org/wiki/Self-balancing_binary_search_tree)like [AVL](http://en.wikipedia.org/wiki/AVL_tree) and [Red-Black trees](http://en.wikipedia.org/wiki/Red-black_tree) guarantee an upper bound of O(Logn) for search.
2. We can insert/delete keys in moderate time (quicker than Arrays and slower than Unordered Linked Lists). [Self-balancing search trees](http://en.wikipedia.org/wiki/Self-balancing_binary_search_tree)like [AVL](http://en.wikipedia.org/wiki/AVL_tree) and [Red-Black trees](http://en.wikipedia.org/wiki/Red-black_tree) guarantee an upper bound of O(Logn) for insertion/deletion.
3. Like Linked Lists and unlike Arrays, Pointer implementation of trees don’t have an upper limit on number of nodes as nodes are linked using pointers.

**Other Applications :**

1. Store hierarchical data, like folder structure, organization structure, XML/HTML data.
2. [Binary Search Tree](http://www.geeksforgeeks.org/binary-search-tree-set-1-search-and-insertion/) is a tree that allows fast search, insert, delete on a sorted data. It also allows finding closest item
3. [Heap](https://www.geeksforgeeks.org/heap-data-structure/) is a tree data structure which is implemented using arrays and used to implement priority queues.
4. [B-Tree](https://www.geeksforgeeks.org/b-tree-set-1-introduction-2/) and[B+ Tree](https://www.geeksforgeeks.org/database-file-indexing-b-tree-introduction/) : They are used to implement indexing in databases.
5. [Syntax Tree](https://www.geeksforgeeks.org/compiler-design-syntax-directed-translation/): Used in Compilers.
6. [K-D Tree:](https://www.geeksforgeeks.org/k-dimensional-tree/)A space partitioning tree used to organize points in K dimensional space.
7. [Trie](http://www.geeksforgeeks.org/trie-insert-and-search/) : Used to implement dictionaries with prefix lookup.
8. [Suffix Tree](https://www.geeksforgeeks.org/pattern-searching-set-8-suffix-tree-introduction/) : For quick pattern searching in a fixed text.
9. [Spanning Trees](https://www.geeksforgeeks.org/applications-of-minimum-spanning-tree/) and shortest path trees are used in routers and bridges respectively in computer networks
10. As a workflow for compositing digital images for visual effects.

**Minimum Spanning Tree**

<https://youtu.be/0tBzHYoTfiY>

<https://www.youtube.com/watch?v=0tBzHYoTfiY&feature=youtu.be>

**Continous tree**

A tree is a Continuous tree if in each root to leaf path, the absolute difference between keys of two adjacent is 1. We are given a binary tree, we need to check if the tree is continuous or not.  
Examples: 

Input : 3

/ \

2 4

/ \ \

1 3 5

Output: "Yes"

// 3->2->1 every two adjacent node's absolute difference is 1

// 3->2->3 every two adjacent node's absolute difference is 1

// 3->4->5 every two adjacent node's absolute difference is 1

Input : 7

/ \

5 8

/ \ \

6 4 10

Output: "No"

// 7->5->6 here absolute difference of 7 and 5 is not 1.

// 7->5->4 here absolute difference of 7 and 5 is not 1.

// 7->8->10 here absolute difference of 8 and 10 is not 1.

[**https://www.geeksforgeeks.org/continuous-tree/**](https://www.geeksforgeeks.org/continuous-tree/)