# Binary Tree Data Structure

A tree whose elements have at most 2 children is called a binary tree. Since each element in a binary tree can have only 2 children, we typically name them the left and right child.



A Binary Tree node contains following parts.

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

Introduction :

# Binary Tree | Set 1 (Introduction)

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: A tree whose elements have at most 2 children is called a binary tree. Since each element in a binary tree can have only 2 children, we typically name them the left and right child.

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.

* C
* Python
* Java
* C#

filter\_none

brightness\_4

|  |
| --- |
| /\* Class containing left and right child of current     node and key value\*/  **class** Node  {  **int** key;      Node left, right;    **public** Node(**int** item)      {          key = item;          left = right = **null**;      }  } |

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

* C
* Python
* Java
* C#

filter\_none

edit

play\_arrow

brightness\_4

|  |
| --- |
| /\* Class containing left and right child of current     node and key value\*/  **class** Node  {  **int** key;      Node left, right;    **public** Node(**int** item)      {          key = item;          left = right = **null**;      }  }    // A Java program to introduce Binary Tree  **class** BinaryTree  {      // Root of Binary Tree      Node root;        // Constructors      BinaryTree(**int** key)      {          root = **new** Node(key);      }        BinaryTree()      {          root = **null**;      }    **public** **static** **void** main(String[] args)      {          BinaryTree tree = **new** BinaryTree();            /\*create root\*/          tree.root = **new** Node(1);            /\* following is the tree after above statement                  1              /   \            null  null     \*/            tree.root.left = **new** Node(2);          tree.root.right = **new** Node(3);            /\* 2 and 3 become left and right children of 1                 1               /   \              2      3            /    \    /  \          null null null null  \*/              tree.root.left.left = **new** Node(4);          /\* 4 becomes left child of 2                      1                  /       \                 2          3               /   \       /  \              4    null  null  null             /   \            null null           \*/      }  } |

Summary: Tree is a hierarchical data structure. Main uses of trees include maintaining hierarchical data, providing moderate access and insert/delete operations. Binary trees are special cases of tree where every node has at most two children.

1) The maximum number of nodes at level ‘l’ of a binary tree is 2l.  
Here level is number of nodes on path from root to the node (including root and node). Level of root is 0.  
This can be proved by induction.  
For root, l = 0, number of nodes = 20 = 1  
Assume that 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

Real life Usage?

2) Maximum number of nodes in a binary tree of height ‘h’ is 2h – 1.  
Here height of a tree is maximum number of nodes on root to leaf path. Height of a tree with 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, height of the root is considered as 0. In this convention, the above formula becomes 2h+1 – 1

Real life usage ?