Data Structures and Algorithms

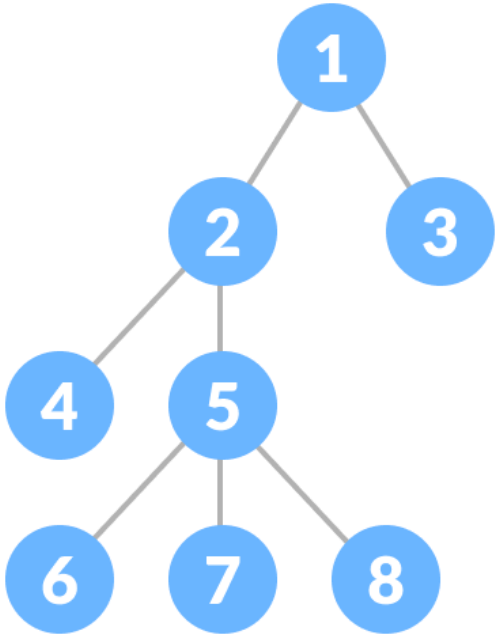
Lecture 25: Trees - Binary trees

Trees

A tree is a nonlinear hierarchical data structure that consists of nodes connected by edges.

The data in a tree are not stored in a sequential manner i.e., they are not stored linearly.

Instead, they are arranged on multiple levels or we can say it is a hierarchical structure. For this reason, the tree is considered to be a non-linear data structure.



Why Tree Data Structure?

Other data structures such as arrays, linked list, stack, and queue are linear data structures that store data sequentially. In order to perform any operation in a linear data structure, the time complexity increases with the increase in the data size. But, it is not acceptable in today's computational world.

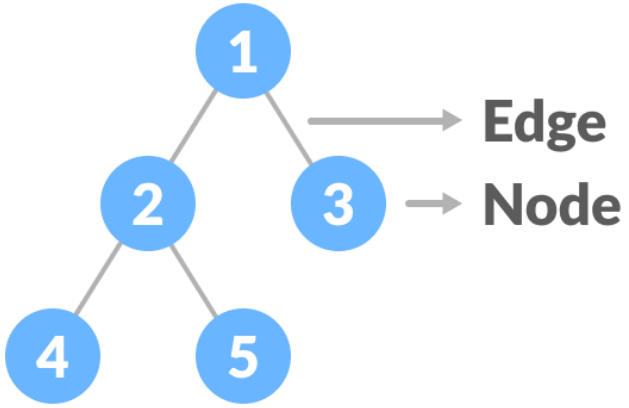
Different tree data structures allow quicker and easier access to the data as it is a non-linear data structure.

Tree Terminologies

Node - A node is an entity that contains a key or value and pointers to its child nodes.

- The last nodes of each path are called leaf nodes or external nodes that do not contain a link/pointer to child nodes.
- The node having at least a child node is called an internal node.

Edge - It is the link between any two nodes.



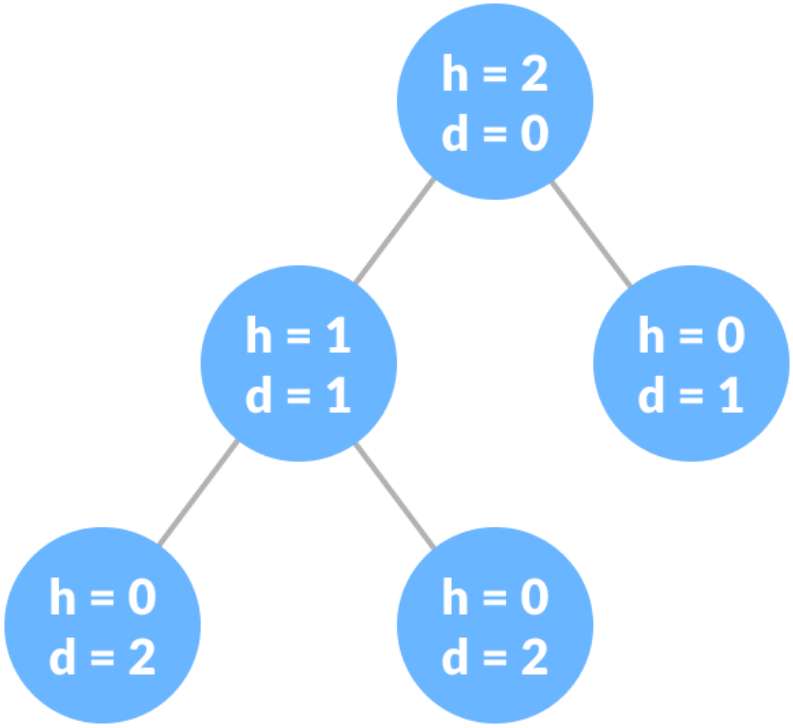
Tree Terminologies

Root - It is the topmost node of a tree.

Height of a Node - The height of a node is the number of edges from the node to the deepest leaf (ie. the longest path from the node to a leaf node).

Depth of a Node - The depth of a node is the number of edges from the root to the node.

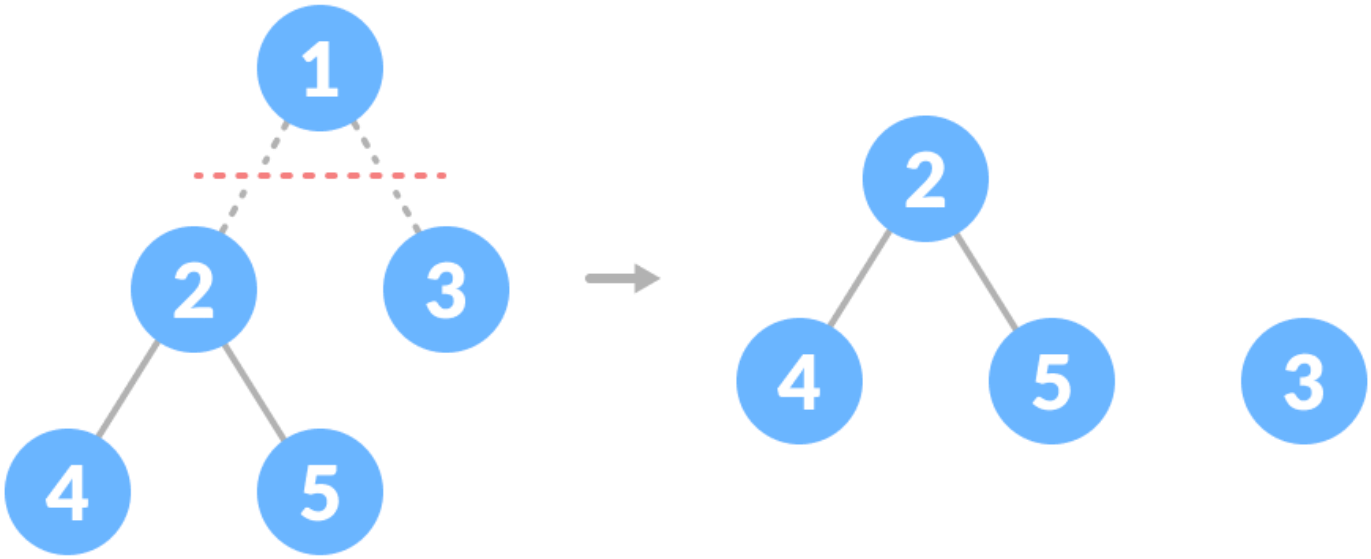
Height of a Tree - The height of a Tree is the height of the root node or the depth of the deepest node.



Tree Terminologies

Degree of a Node - The degree of a node is the total number of branches of that node.

Forest - A collection of disjoint trees is called a forest.



Tree Terminologies

Types of Tree:

- Binary Tree
- Binary Search Tree
- AVL Tree
- B-Tree

Operations of Tree DS

- **Create** – create a tree in the data structure.
- **Insert** − Inserts data in a tree.
- **Search** − Searches specific data in a tree to check whether it is present or not.
- **Traversal**:

Preorder Traversal – perform Traveling a tree in a pre-order manner in the data structure.

In order Traversal – perform Traveling a tree in an in-order manner.

Post-order Traversal –perform Traveling a tree in a post-order manner.

Operations of Tree DS

- **Tree Traversal**
- In order to perform any operation on a tree, you need to reach to the specific node. The tree traversal algorithm helps in visiting a required node in the tree.

Application of Tree Data Structure

- **File System:** This allows for efficient navigation and organization of files.
- **Data Compression:** Huffman coding is a popular technique for data compression that involves constructing a binary tree where the leaves represent characters and their frequency of occurrence. The resulting tree is used to encode the data in a way that minimizes the amount of storage required.
- **Compiler Design:** In compiler design, a syntax tree is used to represent the structure of a program.
- **Database Indexing:** B-trees and other tree structures are used in database indexing to efficiently search for and retrieve data.

Advantages of Tree Data Structure:

- Tree offer Efficient Searching Depending on the type of tree, with average search times of O(log n) for balanced trees like AVL.
- Trees provide a hierarchical representation of data, making it easy to organize and navigate large amounts of information.
- The recursive nature of trees makes them easy to traverse and manipulate using recursive algorithms.

Disadvantages of Tree Data Structure:

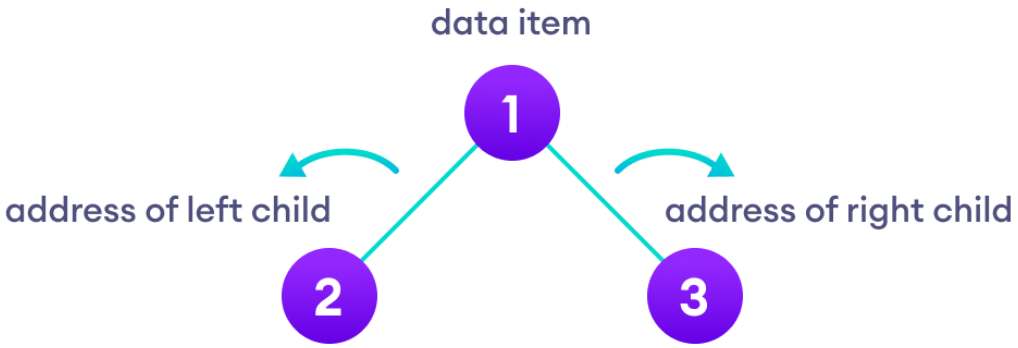
- Unbalanced Trees, meaning that the height of the tree is skewed towards one side, which can lead to inefficient search times.
- Trees demand more memory space requirements than some other data structures like arrays and linked lists, especially if the tree is very large.
- The implementation and manipulation of trees can be complex and require a good understanding of the algorithms.

Binary Tree

A binary tree is a tree data structure in which each parent node can have at most two children.

Each node of a binary tree consists of three items:

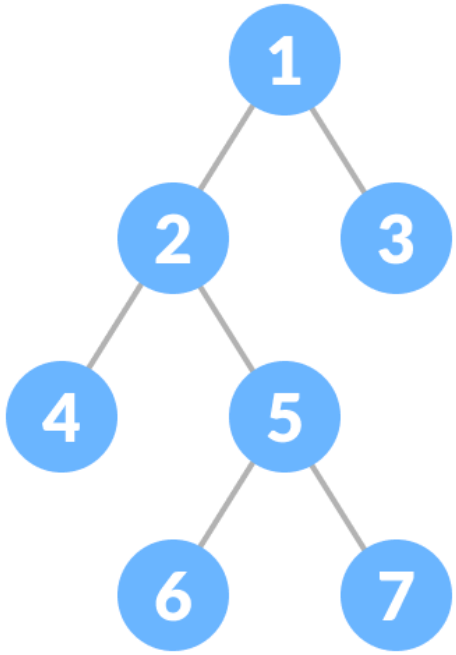
- data item
- address of left child
- address of right child



Types of Binary Tree

1. Full Binary Tree:

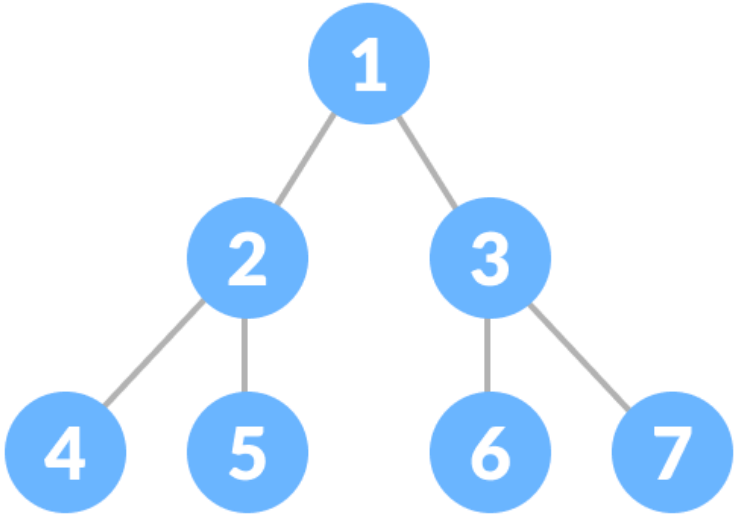
A full Binary tree is a special type of binary tree in which every parent node/internal node has either two or no children.



Types of Binary Tree

2. Perfect Binary Tree

A perfect binary tree is a type of binary tree in which every internal node has exactly two child nodes and all the leaf nodes are at the same level.

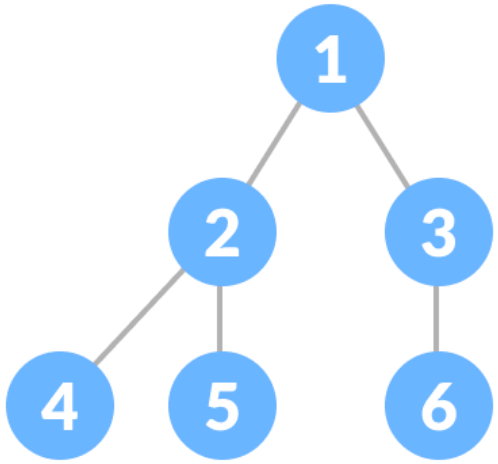


Types of Binary Tree

3. Complete Binary Tree

A complete binary tree is just like a full binary tree, but with two major differences

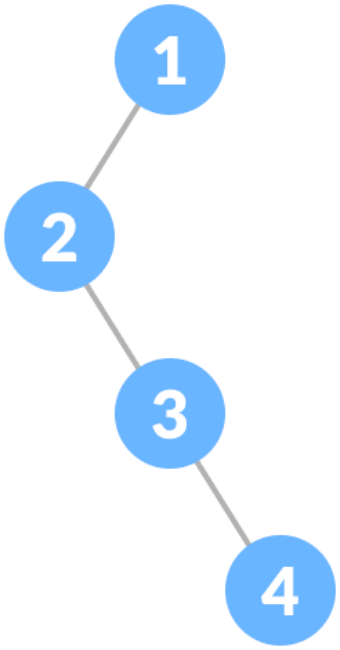
1. Every level must be completely filled
2. All the leaf elements must lean towards the left.
3. The last leaf element might not have a right sibling i.e. a complete binary tree doesn't have to be a full binary tree.



Types of Binary Tree

4. Degenerate or Pathological Tree:

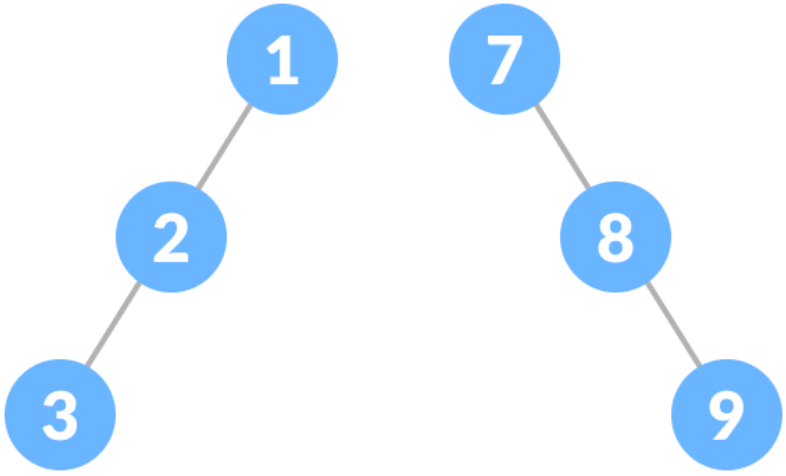
A degenerate or pathological tree is the tree having a single child either left or right.



Types of Binary Tree

5. Skewed Binary Tree:

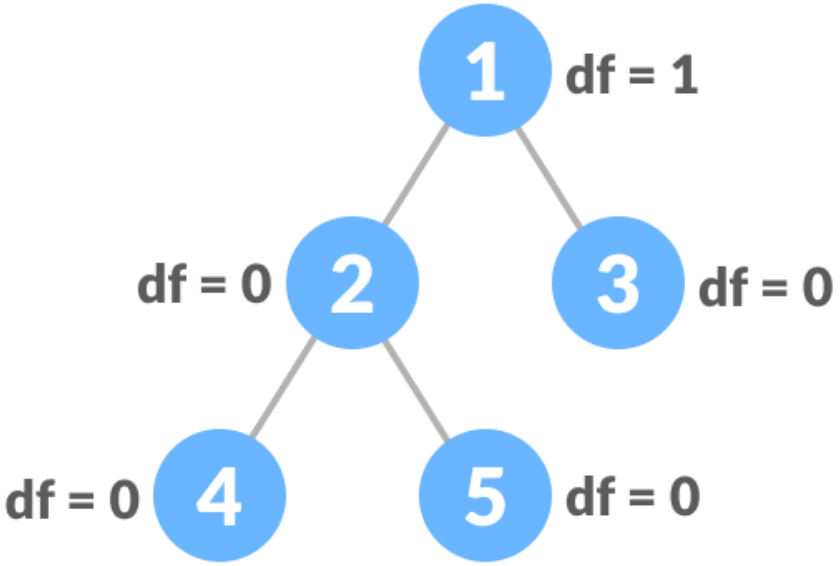
A skewed binary tree is a pathological/degenerate tree in which the tree is either dominated by the left nodes or the right nodes. Thus, there are two types of skewed binary tree: left-skewed binary tree and right-skewed binary tree.



Types of Binary Tree

6. Balanced Binary Tree:

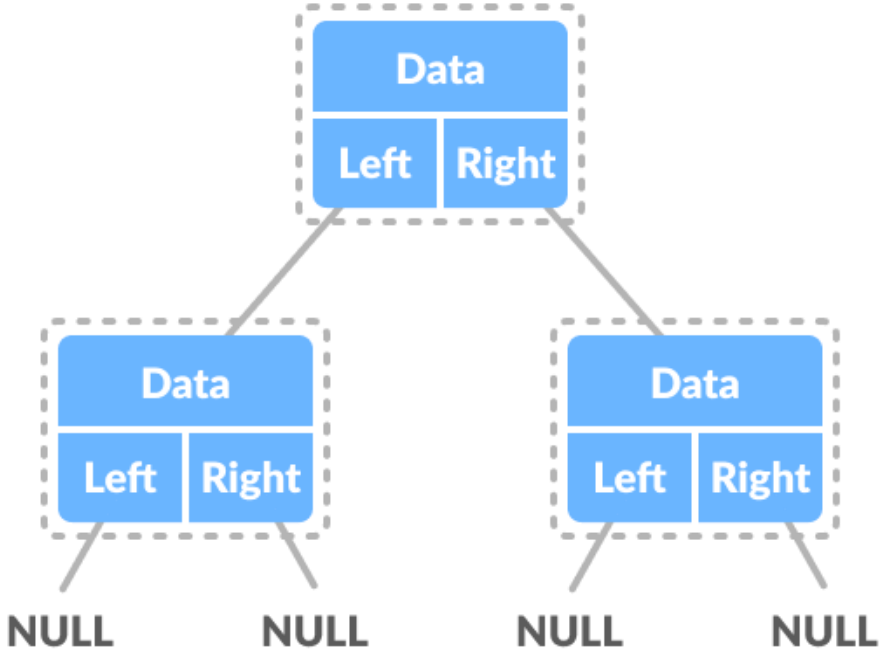
It is a type of binary tree in which the difference between the height of the left and the right subtree for each node is either 0 or 1.



Binary Tree Representation

A node of a binary tree is represented by a structure containing a data part and two pointers to other structures of the same type.

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



Application of Tree Data Structure

- File Systems
- Search Engines
- Searching and Sorting Algorithms
- Expression Trees
- Huffman Coding
- Game AI

Application of Tree Data Structure

Field	Application
File Systems	Organizing and storing files
Search Engines	Indexing and ranking web pages
Sorting	Sorting algorithms, such as binary insertion sort and quicksort
Math	Representing math expressions for evaluating and simplifying
Data Compression	Encoding characters using Huffman coding
Decision Trees	Modeling decisions and consequences in machine learning
String Retrieval	Storing and retrieving strings using trie data structures
Game AI	Modeling possible moves and outcomes in game artificial intelligence
Network Routing	Routing data through computer networks
Arithmetic Coding	Encoding characters with variable-length codes using a binary tree
Priority Queues	Efficient access to the item with the highest priority
Image Processing	Representing image regions or shapes
Cryptography	Generating and managing encryption and decryption keys

Binary Tree- Representation in Memory

Array Representation:

- A small and almost complete binary tree can be easily stored in a linear array. Small tree is preferably stored in linear array because searching process in a linear array is expensive. Complete means that if most of the nodes have two child nodes.
- To store binary tree in a linear array, you need to consider the positional indexes of the nodes. This indexing must be considered starting with 1 from the root node going from left to right as you go down from one level to other.

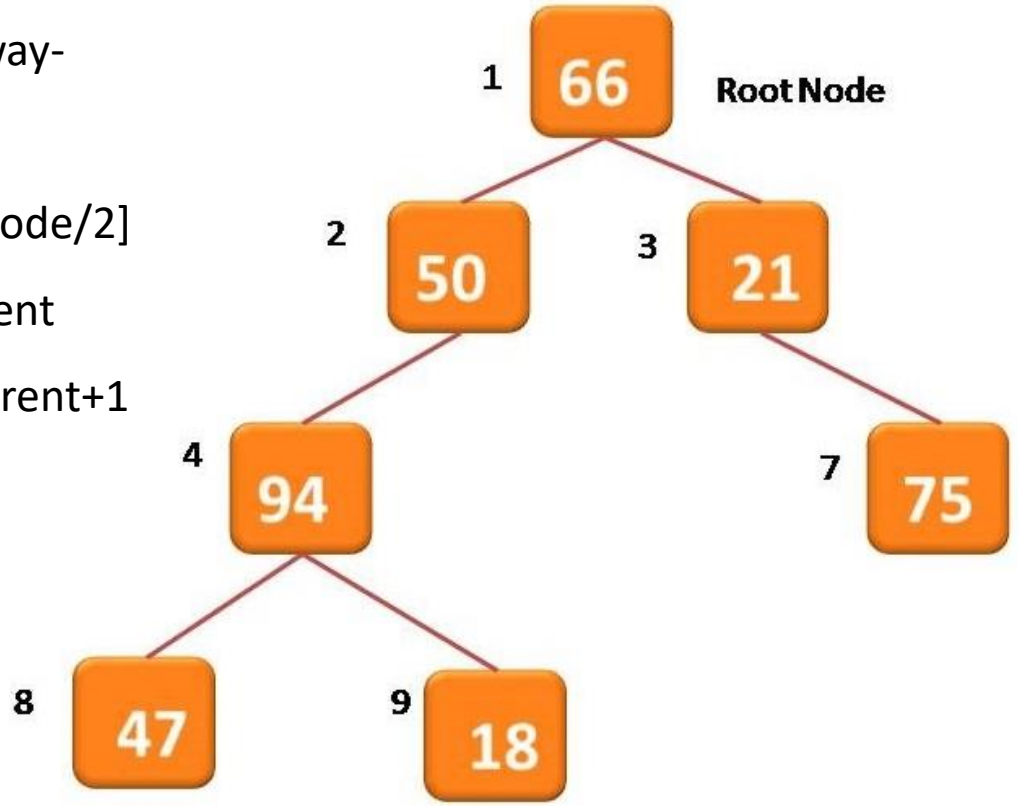
Binary Tree- Array Representation

Assigning of indexes is done in this way-

Index of parent= INT[index of child node/2]

Index of Left Child = 2 * Index of parent

Index of Right Child = 2 * Index of parent+1



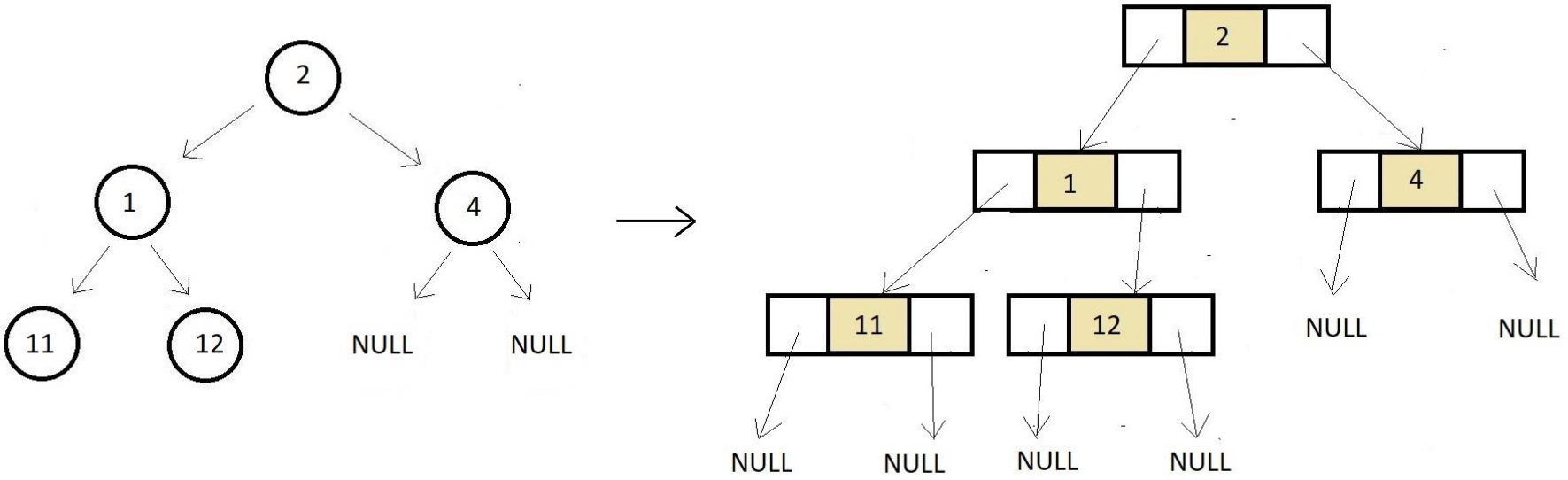
Binary Tree- Array Representation

These rules are used to store the tree of the above example in an array

1	2	3	4	5	6	7	8	9
66	50	21	94			75	47	18

If a binary tree contains less number of elements by is deep In structure, the memory underutilization is a major issue.

Binary Tree in a Linked Representation



Binary Tree in a Linked Representation

We use a double linked list to represent a binary tree. In a double linked list, every node consists of three fields. First field for storing left child address, second for storing actual data and third for storing right child address.

In this linked list representation, a node has the following structure...