Binary Tree

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

A binary tree is a structure comprising nodes, where each node has the following 3 components:

1. Data element: Stores any kind of data in the node.
2. Left pointer: Points to the tree on the left side of node.
3. Right pointer: Points to the tree on the right side of the node.

As the name suggests, the data element stores any kind of data in the node.
The left and right pointers point to binary trees on the left and right side of the node respectively.
If a tree is empty, it is represented by a null pointer.

Commonly-used terminologies

- **Root:** Top node in a tree
- **Child:** Nodes that are next to each other and connected downwards
- **Parent:** Converse notion of child
- **Siblings:** Nodes with the same parent
- **Descendant:** Node reachable by repeated proceeding from parent to child
- **Ancestor:** Node reachable by repeated proceeding from child to parent.
- **Leaf:** Node with no children
- **Internal node:** Node with at least one child
- **External node:** Node with no children

Binary Tree properties

1. The maximum number of nodes at level 'l' of a binary tree is 2^{l-1} (root at l = 1)
2. Maximum number of nodes in a binary tree of height 'h' is $2^h – 1$ (root at h = 1)
3. In a Binary Tree with N nodes, minimum possible height or minimum number of levels is $[Log_2(N+1)]$
4. A Binary Tree with L leaves has at least $[Log_2L] + 1$ levels

```
L   =  2^(l-1)  [From 1]
l =   ⌈Log2(L)⌉ + 1
where l is the minimum number of levels.
```

5. In Binary tree where every node has 0 or 2 children, number of leaf nodes is always one more than nodes with two children (proof: Handshaking Lemma and Tree)

```
L = T + 1
Where L = Number of leaf nodes
      T = Number of internal nodes with two children
```

Types of Binary Trees

- **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.
- **Perfect Binary Tree:** A Binary tree is Perfect Binary Tree in which all internal nodes have two children and all leaves are at the same level. it satisfy $N = 2^h – 1$
- **Balanced Binary Tree:** A binary tree is balanced if the height of the tree is O(Log n) where n is the number of nodes.
  For Example, AVL tree maintains O(Log n) height by making sure that the difference between heights of left and right sub-trees is 1.
  Red-Black trees maintain O(Log n) height by making sure that the number of Black nodes on every root to leaf paths are same and there are no adjacent red nodes.
  Balanced Binary Search trees are performance wise good as they provide O(log n) time for search, insert and delete.
- **A Degenerate (or Pathological) Tree:** A Tree where every internal node has one child. Such trees are performance-wise same as linked list.

When to use 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.
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 or references.

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 composting digital images for visual effects.
5. Router algorithms
6. Forming of a multi-stage decision-making (see business chess).