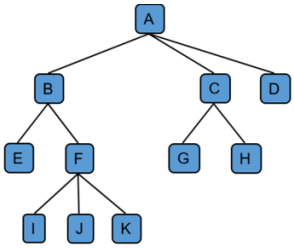
TREE DATA STRUCTURES AND ALGORITHMS

Tree- Introduction

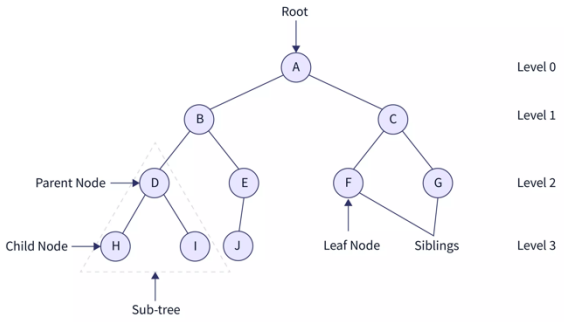


Tree Data Structure is a hierarchical structure that is used to represent and organize data in a way that is easy to navigate and search. It is a collection of nodes that are co nnected by edges and has a hierarchical relationship between the nodes.

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.

Tree Terminology

Root – the starting node, thought of as the top of the (upside-down) tree. In the example, A is the root.



Parent – every node in a tree, except the root, has exactly one parent. The parent is the node just above. A parent can have many children (siblings). In the example, A is the parent of B, C, and D. B, C, and D are siblings.

Child – a node right below, one step away from the parent. In the example, B is a child of A and K is a child of F

Internal node – a node with at least one child. The internal nodes of the example are A. B. C. and F.

External node, also called a leaf – a node without any children. The leaves of the example are E, I, J, K, G, H, and D.

Level or depth of a node – The root is at level 0. The level of other nodes is determined by how many steps a node is away from the root. For example, node E is at level 2, node J is at level 3, and node C is at level 1.

Height of a tree – the maximum depth of any node in the tree. In our example, the height is 3.

Ancestor – The ancestors of a node include that node, the parent of that node, the grandparent of that node, and so on up to the root. The ancestors of a node contain every node from that node to the root, inclusive. In the example, A is an ancestor of every node in the tree, including A itself.

Proper Ancestor – An ancestor that is not the node itself. For example, A and B are ancestors of B, but only A is a proper ancestor of B.

Descendant – The descendants of a node include that node, the children of that node, the grandchildren of that node, and so on until the bottom of the tree is reached. The descendants of a node is every node from that node to the leaves that may be reached, inclusive. In the example, every node in the tree is a descendant of A, including A itself.

Proper Descendant – A descendant that is not the node itself. For example, F, I, J, and K are descendants of F, but only I, J, and K are proper descendants of F.

Path - The nodes and connections between two nodes, in a downward direction only. For example, there is no path from C to D, nor is there a path from C to A. There is a path from B to J and that path is B-F-J.

Binary Tree – A tree where each node has two link fields (a left child and a right child). Each link field may point to another node or to null so in a binary tree each node has zero, one or two children. The example tree is not a binary tree since A and F have three children.

Subtree – a tree consisting of a node and its descendants. For example, the nodes C, G and H with their connections form a subtree. The node J alone is a subtree. The node B alone is not a subtree. The nodes B, E, F, I, J, and K with their connections form a subtree. A tree may be defined recursively as *a tree is empty or a tree has a root plus the root's subtrees, one subtree per child of the root*.

Visit – Doing work at a particular node of a tree. The work may be printing the node, manipulating the information in the node, using the information somehow, etc.

Traversal – A way of moving through a tree. We always need to start at the root, and progress from the root by following the links.

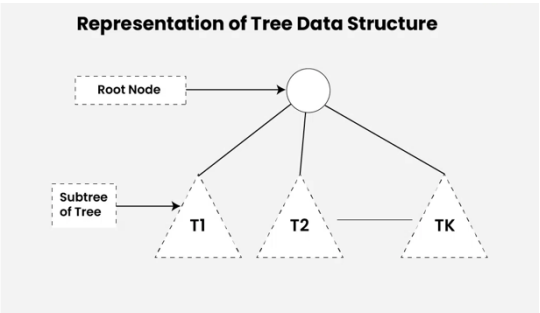
Properties of Tree Data Structure:

- Number of Edges: An edge can be defined as the connection between two nodes. If a tree has N nodes then it will have (N-1) edges. There is only one path from each node to any other node of the tree.
- Depth of a Node: The depth of a node is defined as the length of the path from the root to that node. Each edge adds 1 unit of length to the path. So, it can also be defined as the number of edges in the path from the root of the tree to the node.
- Height of a Node: The height of a node can be defined as the length of the longest path from the node to a leaf node of the tree.
- Height of the Tree: The height of a tree is the length of the longest path from the root of the tree to a leaf node of the tree.
- Degree of a Node: The total count of subtrees attached to that node is called the degree of the node. The degree of a leaf node must be 0. The degree of a tree is the maximum degree of a node among all the nodes in the tree.

Representation of Tree Data Structure

A tree consists of a root node, and zero or more subtrees T1, T2, … , Tk such that there is an edge from the root node of the tree to the root node of each subtree. Subtree of a node X consists of all the nodes which have node X as the ancestor node.

A tree can be represented using a collection of nodes. Each of the nodes can be represented with the help of class.



Types of Tree Data Structures

Tree data structure can be classified into three types based upon the number of children each node of the tree can have. The types are:

- Binary Tree: In a Binary Tree, each node can have a maximum of two children linked to it. Some common types of binary trees include full binary trees, complete binary trees, balanced binary trees, and degenerate or pathological binary trees. Examples of Binary Tree are Binary Search Tree and Binary Heap. On the basis of completion of levels, the binary tree can be divided into following types: Complete Binary Tree, Perfect Binary Tree, Balanced Binary Tree.

- Ternary Tree: A Ternary Tree is a tree data structure in which each node has at most three child nodes, usually distinguished as “left”, “mid” and “right”.

- N-ary Tree or Generic Tree: Generic trees are a collection of nodes where each node is a data structure that consists of records and a list of references to its children(duplicate references are not allowed). Unlike the linked list, each node stores the address of multiple nodes.

Special Types of Trees in Data Structure

Binary Search Tree: A Binary Search Tree is a node-based binary tree data structure that has the following properties:

- The left subtree of a node contains only nodes with keys lesser than the node’s key.
- The right subtree of a node contains only nodes with keys greater than the node’s key.
- The left and right subtree each must also be a binary search tree.

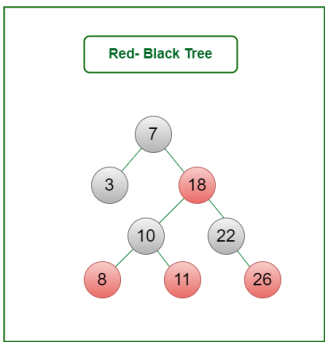
AVL Tree: AVL tree is a self-balancing Binary Search Tree (BST) where the difference between heights of left and right subtrees for any node cannot be more than one.

Red Black Tree: A red-black tree is a kind of self-balancing binary search tree where each node has an extra bit, and that bit is often interpreted as the color (red or black). These colors are used to ensure that the tree remains balanced during insertions and deletions.

Although the balance of the tree is not perfect, it is good enough to reduce the searching time and maintain it around O(log n) time, where n is the total number of elements in the tree.

Rules That Every Red-Black Tree Follows:

- Every node has a color either red or black.
- The root of the tree is always black.
- There are no two adjacent red nodes (A red node cannot have a red parent or red child).
- Every path from a node (including root) to any of its descendants’ NULL nodes has the same number of black nodes.
- All leaf (NULL) nodes are black nodes.



B-Tree: B-Tree is a self-balancing search tree. In most of the other self-balancing search trees (like AVL and Red-Black Trees), it is assumed that everything is in the main memory.

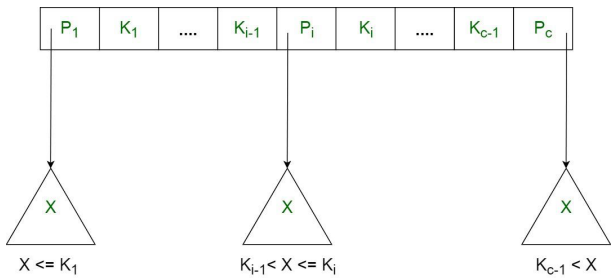
Properties of B-Tree:

- All leaves are at the same level.
- B-Tree is defined by the term minimum degree 't'. The value of 't' depends upon disk block size.
- Every node except the root must contain at least t-1 keys. The root may contain a minimum of 1 key.
- All nodes (including root) may contain at most (2*t – 1) keys.
- The number of children of a node is equal to the number of keys in it plus 1.
- All keys of a node are sorted in increasing order. The child between two keys k1 and k2 contains all keys in the range from k1 and k2.
- B-Tree grows and shrinks from the root which is unlike Binary Search Tree. Binary Search Trees grow downward and also shrink from downward.
- Like other balanced Binary Search Trees, the time complexity to search, insert and delete is O(log n).
- Insertion of a Node in B-Tree happens only at Leaf Node.

B+ Tree: B+ tree eliminates the drawback B-tree used for indexing by storing data pointers only at the leaf nodes of the tree. Thus, the structure of leaf nodes of a B+ tree is quite different from the structure of internal nodes of the B tree.

It may be noted here that, since data pointers are present only at the leaf nodes, the leaf nodes must necessarily store all the key values along with their corresponding data pointers to the disk file block, to access them. Moreover, the leaf nodes are linked to providing ordered access to the records. The leaf nodes, therefore form the first level of the index, with the internal nodes forming the other levels of a multilevel index. Some of the key values of the leaf nodes also appear in the internal nodes, to simply act as a medium to control the search of a record.

The Structure of the internal nodes of a B+ tree of order 'a' is as follows:



- Each internal node is of the form: <P1, K1, P2, K2, ….., Pc-1, Kc-1, Pc> where c <= a and each **Pi is a tree pointer (i.e points to another node of the tree)** and, each **Ki is a key-value** (see diagram-I for reference).
- Every internal node has : K1 < K2 < …. < Kc-1
- For each search field values 'X' in the sub-tree pointed at by Pi, the following condition holds: Ki-1 < X <= Ki, for 1 < i < c and, Ki-1 < X, for i = c (See diagram I for reference)
- Each internal node has at most '**a**' tree pointers.
- The root node has, at least two tree pointers, while the other internal nodes have at least \ceil(a/2) tree pointers each.
- If an internal node has 'c' pointers, c <= a, then it has 'c – 1' key values.

Segment Tree: In computer science, a Segment Tree, also known as a statistic tree, is a tree data structure used for storing information about intervals, or segments. It allows querying which of the stored segments contain a given point. It is, in principle, a static structure; that is, it's a structure that cannot be modified once it's built. A similar data structure is the interval tree.

A segment tree for a set I of n intervals uses O(n log n) storage and can be built in O(n log n) time. Segment trees support searching for all the intervals that contain a query point in time O(log n + k), k being the number of retrieved intervals or segments.