AEL-2

Tree Data Structure

Definition and Types

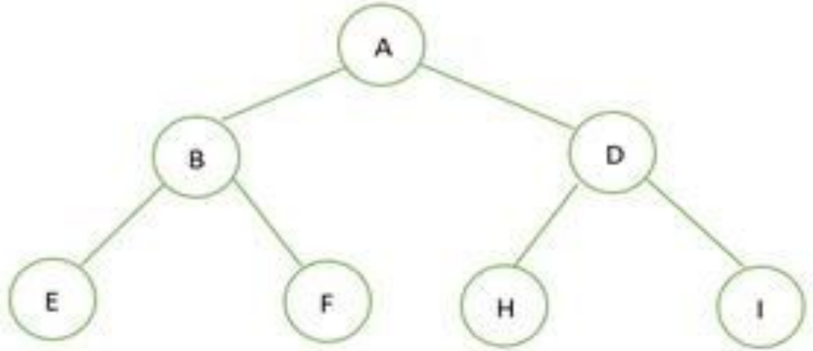
A **tree data structure** is a hierarchical structure that is used to represent and organize data in a way that is easy to navigate and search.

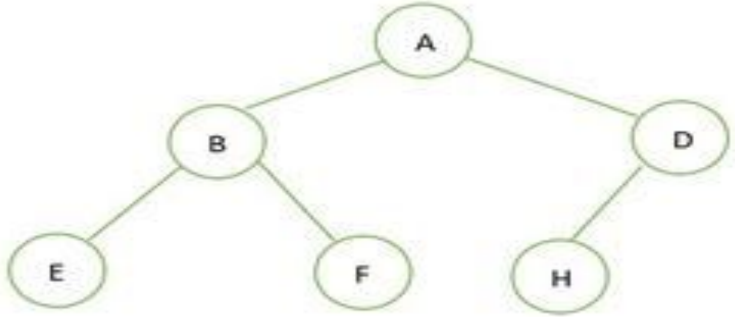
Tree can be of three types : Binary, Ternary and N-ary or Generic Tree.

But normally when we ask about types of Tree, we mean types of Binary Tree.

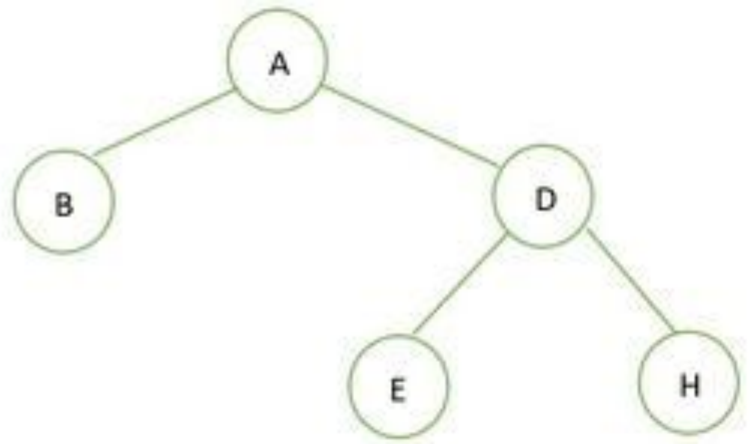
a) Full : A full binary tree is a binary tree with either zero or two child nodes for each node.
b) Complete : A complete binary tree is a special type of binary tree where all the levels of the tree are filled completely except the lowest level nodes which are filled from as left as possible.
c) Perfect : A binary tree of height ‘h’ having the maximum number of nodes is a perfect binary tree.

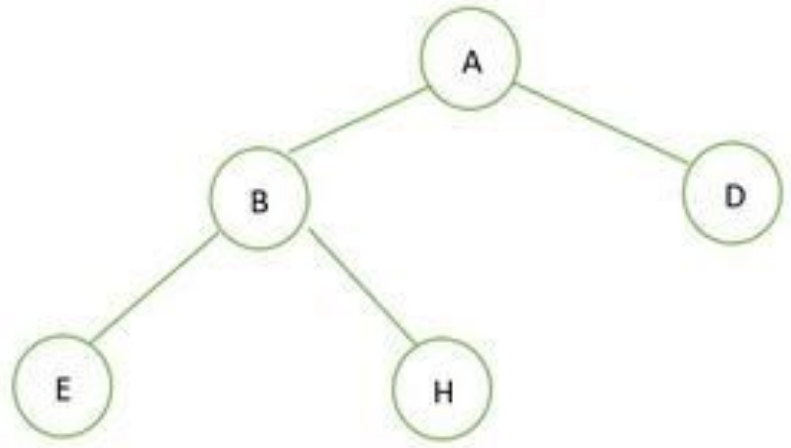
Perfect vs Complete





Full and Complete Binary Tree





Tree Traversal

The most famous two tree traversal technique is DFS and BFS.

DFS : Depth-first search is an algorithm for traversing or searching tree or graph data structures. The algorithm starts at the root node (selecting some arbitrary node as the root node in the case of a graph) and explores as far as possible along each branch before backtracking.

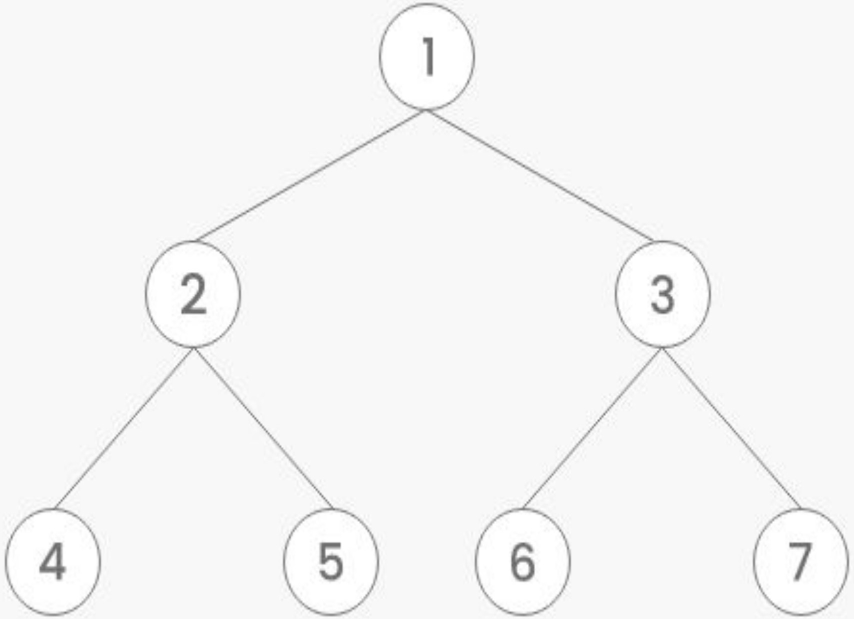
BFS : Breadth First Search or Level Order Traversal technique is defined as a method to traverse a Tree such that all nodes present in the same level are traversed completely before traversing the next level.

There are three types of BST traversal : In order, Pre order and Post order.

For In order algorithm is,

- Traverse left subtree.
- Visit the root and print the data.
- Traverse the right subtree.

Tree Traversal Techniques



Inorder Traversal

4	2	5	1	6	3	7

Preorder Traversal

1	2	4	5	3	6	7

Postorder Traversal

4	5	2	6	7	3	1

Tree Properties

- **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.

Application of Trees

- **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.

BST

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

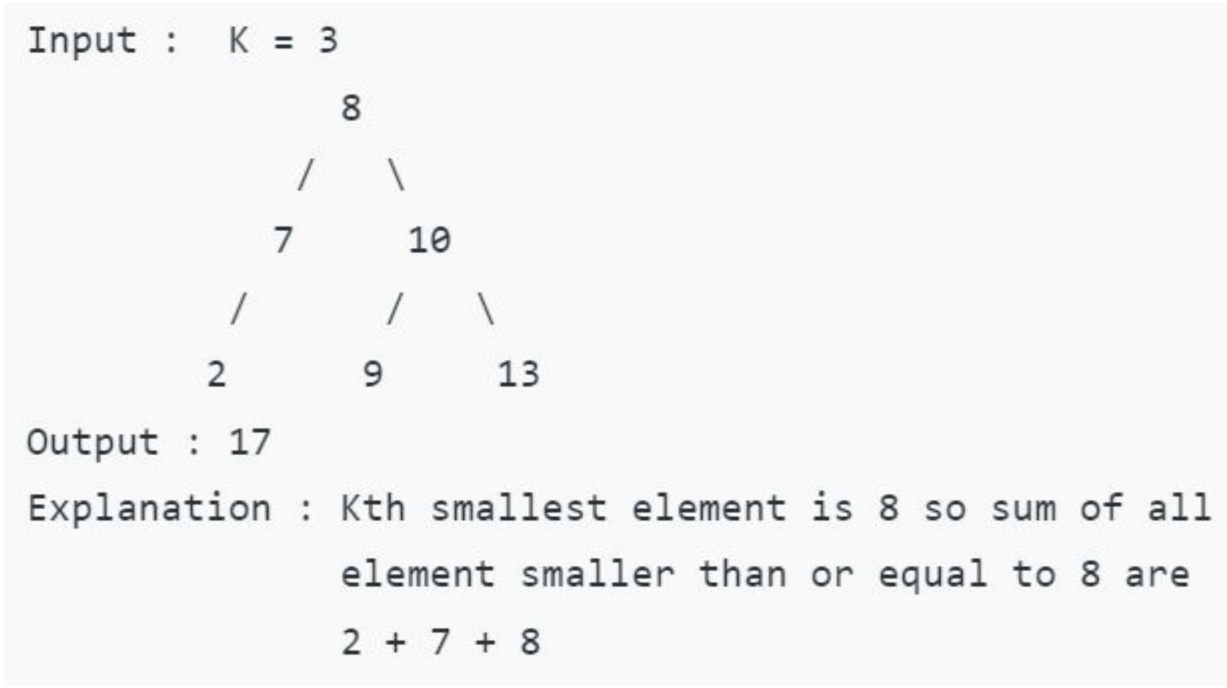
- The left subtree of a node contains only nodes with keys lesser than the node’s key.
- The right subtree of a node contains only nodes with keys greater than the node’s key.
- The left and right subtree each must also be a binary search tree.

Problem

Given Binary Search Tree.

The task is to find sum of all elements smaller than and equal to Kth smallest element.

Solution : Go for Inorder traversal (it would make our traversal sorted) and and count if we have already visited k nodes.



```
Input :  K = 3
              8
            /   \
           7     10
         /     /   \
        2     9     13
Output : 17
Explanation : Kth smallest element is 8 so sum of all
              element smaller than or equal to 8 are
              2 + 7 + 8
```

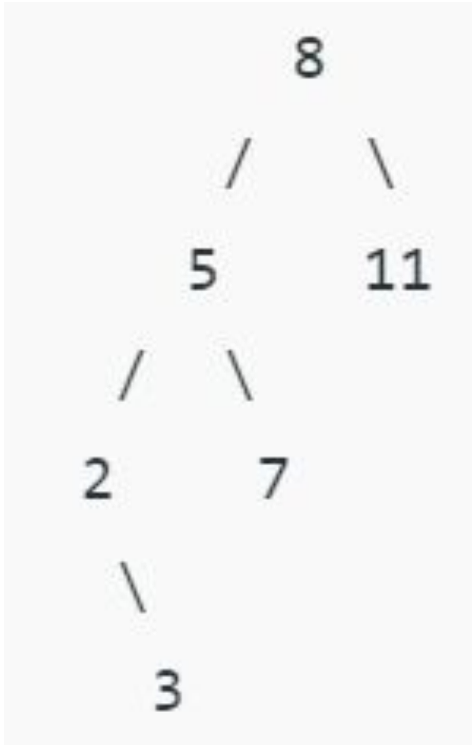
Continued

But the previous solution takes O(k) complexity. We want to do it in O(h) complexity where h means height of the binary tree.

The right side pseudo code snippet can solve the problem in O(h).

```
BST Node contain to extra fields : Lcount , Sum

For each Node of BST
lCount : store how many left child it has
Sum     : store sum of all left child it has

Find Kth smallest element
[ temp_sum store sum of all element less than equal to K ]

ksmallestElementSumRec(root, K, temp_sum)

  IF root -> lCount == K + 1
      temp_sum += root->data + root->sum;
      break;
  ELSE
     IF k > root->lCount   // Goto right sub-tree
        temp_sum += root->data + root-> sum;
        ksmallestElementSumRec(root->right, K-root->lcount+1, temp_sum)
     ELSE
        // Goto left sub-tree
        ksmallestElementSumRec( root->left, K, temp_sum)
```

Another Test Case with K = 3



```
        8
      /   \
    5       11
  /   \
2       7
  \
    3
```

Segment Tree

A Segment Tree is a data structure that **stores information about a range of elements in its nodes.**

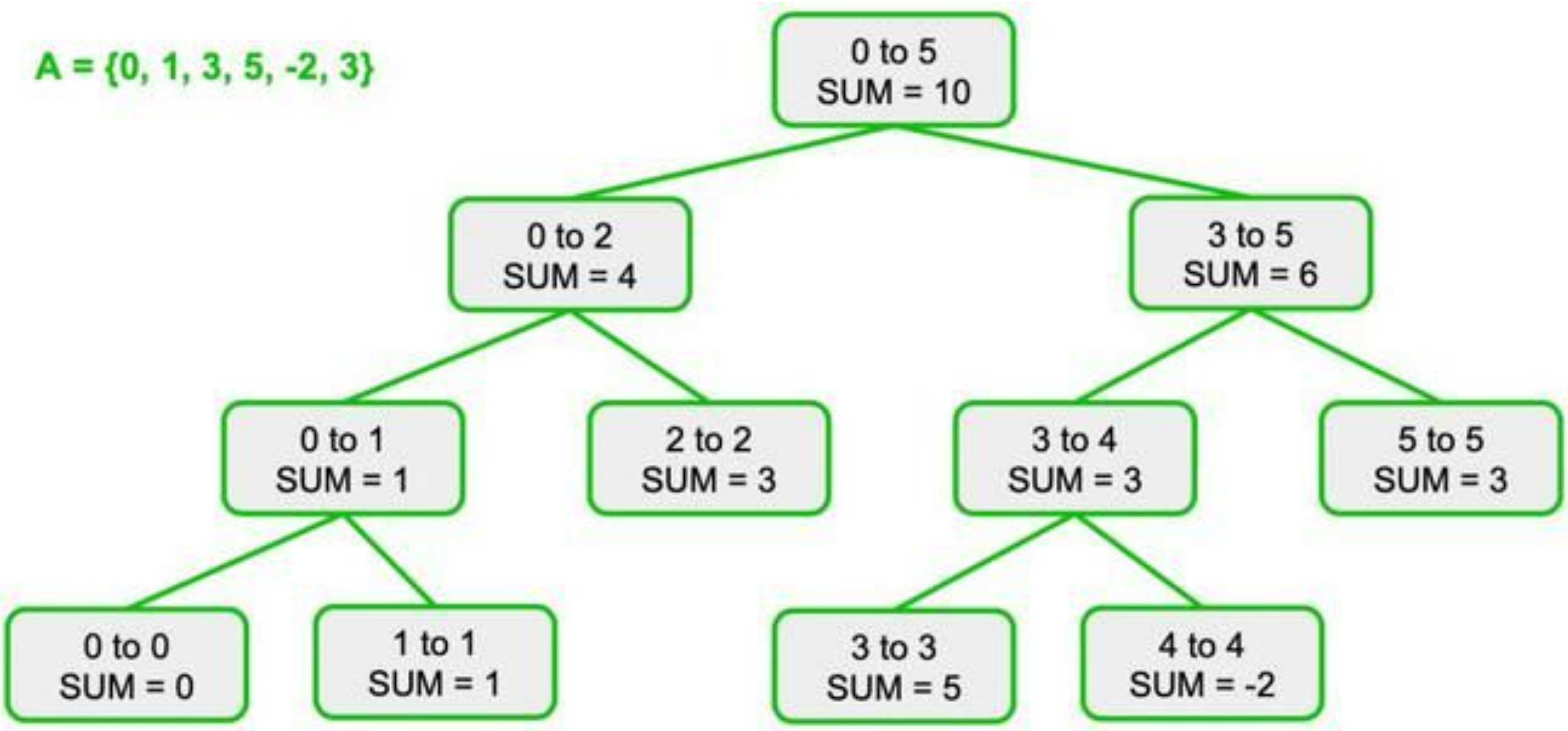
It also allows users to **modify the array** and **perform range queries in smaller complexity.**

For example, we can perform a range summation of an array between the range L to R while also modifying the array from range **L** to **R** all in **log(N)** time complexity.

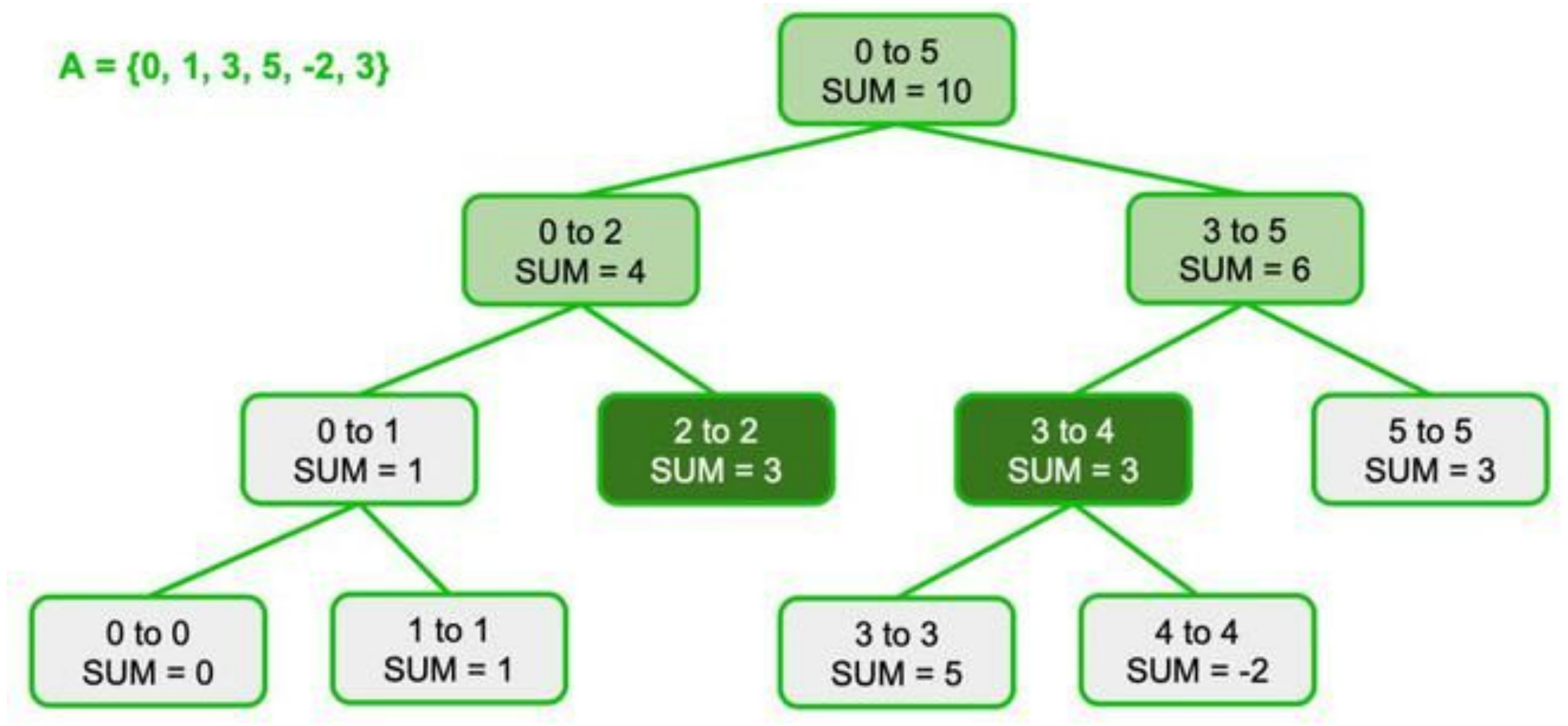
Some of the examples of operations are:

- Addition/Subtraction
- Maximum/Minimum
- GCD/LCM
- AND/OR/XOR

Building Segment Tree O(nlogn)



Range query O(logn)



For sum from range 2 to 4 nodes that are highlighted are traversed. The answer is collected from the dark highlighted nodes.