Nonlinear Data Structure

Unit#3

Marwadi University

Department of Computer Engineering

Data Structure
01CE0301 / 3130702

Ravikumar Natarajan

Highlights

- Trees
- Graphs

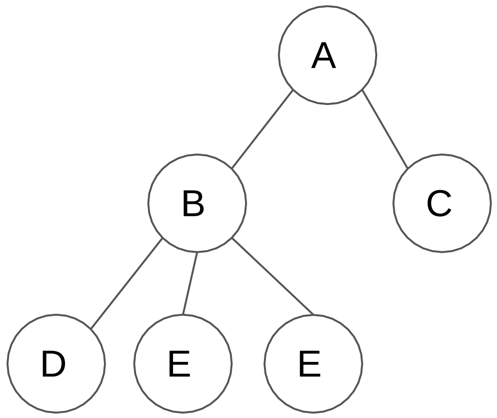
Highlights

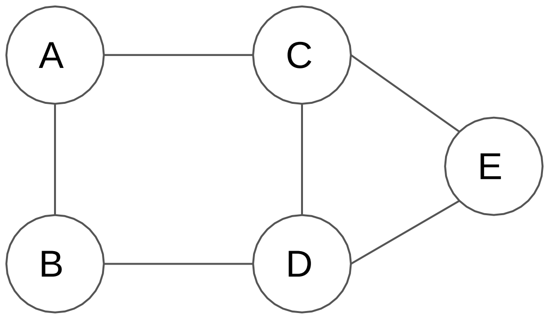
- **Trees**
- Graphs

- Tree definitions and their concepts
- Representation of binary tree
- Binary tree traversal
- Binary search trees
- General trees vs binary trees
- Threaded binary tree
- Applications of Trees
- Balanced tree and its mechanism
- Height and Weight Balanced Trees

Non-Linear Data Structure

- Every data item is attached to several other data items in a way that is specific for reflecting relationships.
- The data items are **not** arranged in a **sequential** structure.
- Ex: Trees, Graphs



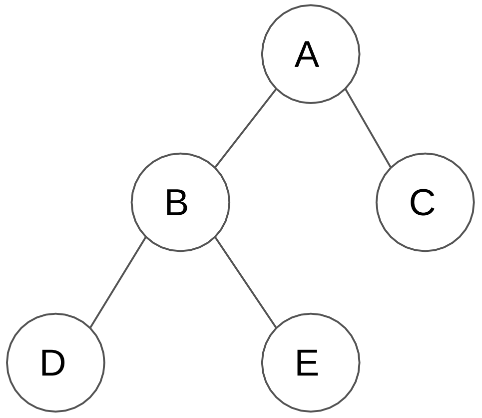


Non-Linear Data Structure

- Characteristics:
  - Not stored in sequential order
  - Branches to more than one node
  - Can't be traversed in a single run
  - Data members are not processed one after another

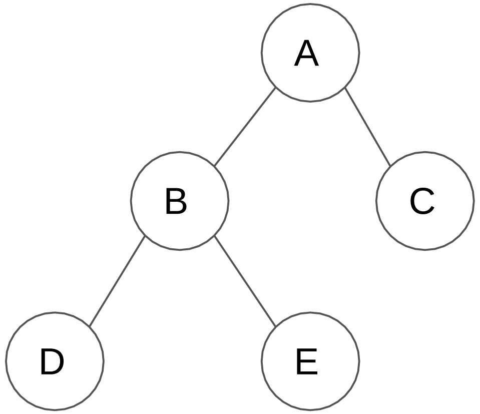
Tree

- A tree is a Multilevel data structure that represent a hierarchical relationship between the set of individual elements called nodes.

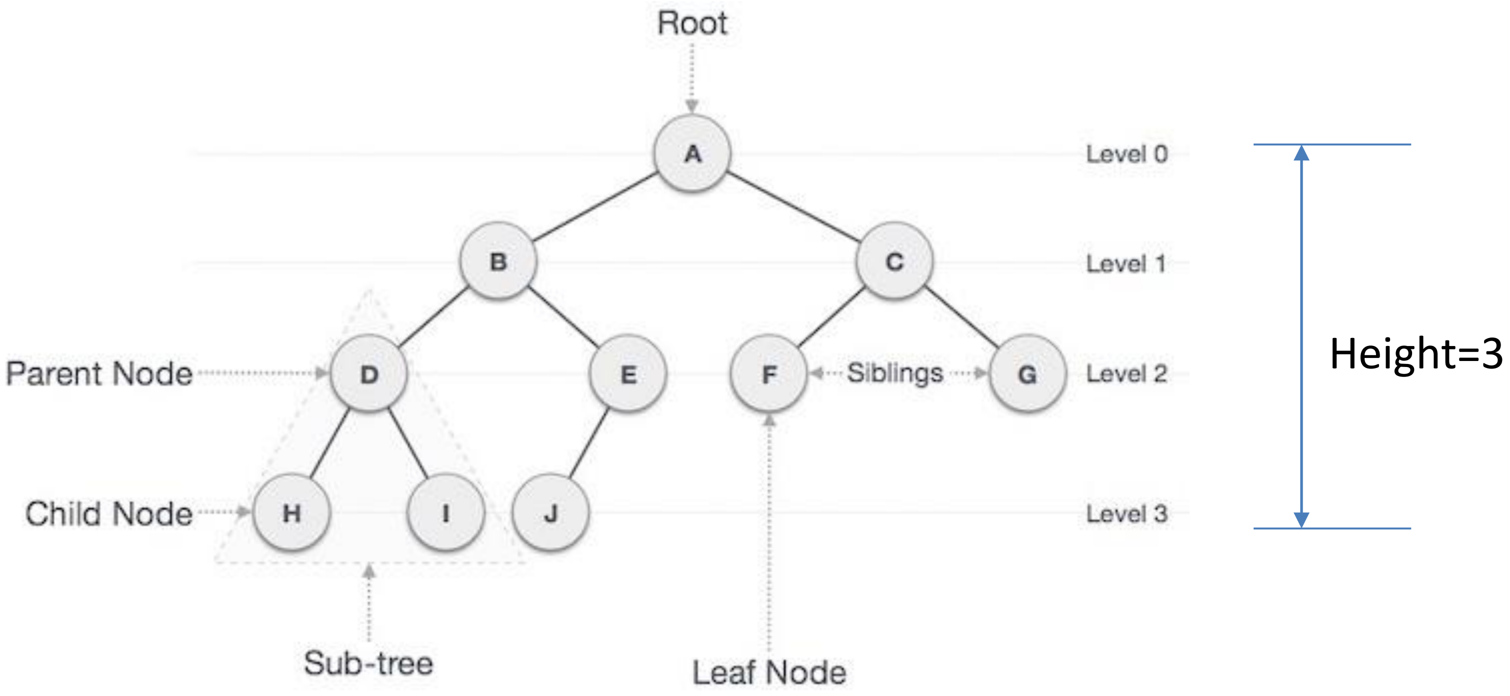


Tree

- A **vertex** (or **node**) is a simple object that can have a name and can carry other associated information. Left pointer, Right pointer and a data element.
- The first or top node in a tree is called the **root** node.
- An **edge** is a connection between two vertices.
- A **path** in a tree is a list of distinct vertices in which successive vertices are connected by edges in the tree.
- Nodes with no children are **leaves or terminal nodes**.

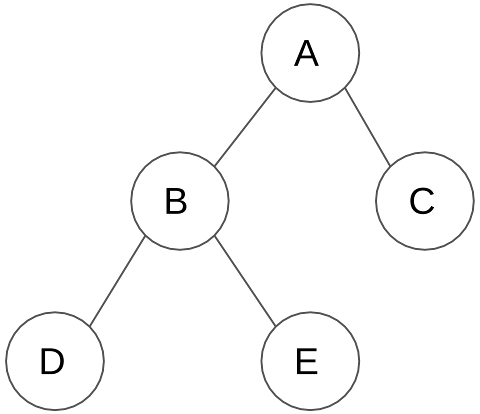


Tree


Root
A
B
C
D
E
F
G
H
I
J
Level 0
Level 1
Level 2
Level 3
Parent Node
Child Node
Siblings
Sub-tree
Leaf Node
Height=3

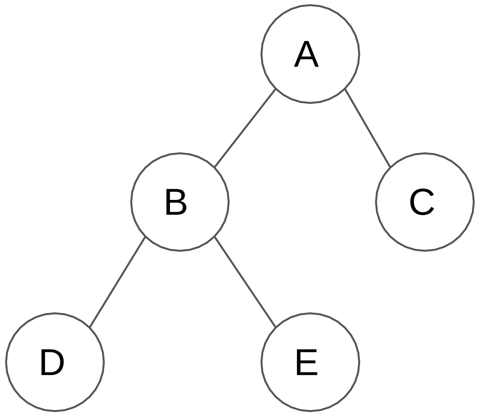
Binary Trees

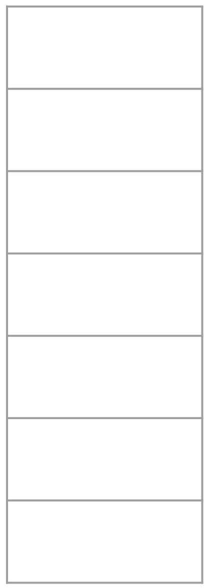
- A binary tree is a tree where each node has exactly zero, one or two children.
- Each parent can have no more than 2 children.



- D, E, C are having zero or no successors and thus are said to be empty sub-trees.

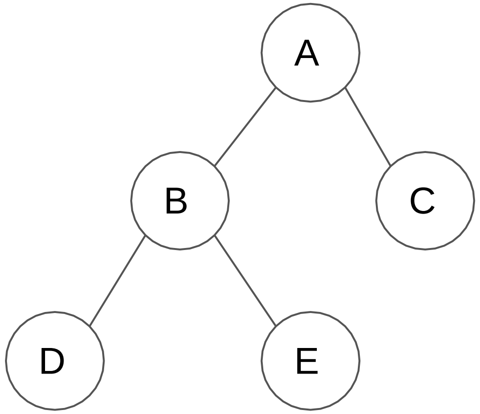
Array Representation of binary tree

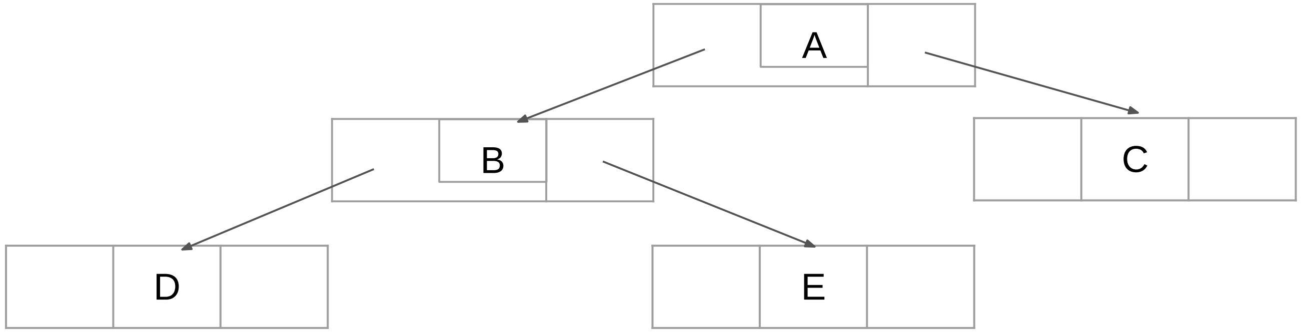




- If K is the parent, then (Consider K as root node)
  - Left child – 2*K = 2*1 = 2
  - Right child – (2*K)+1 = (2*1)+1 = 3

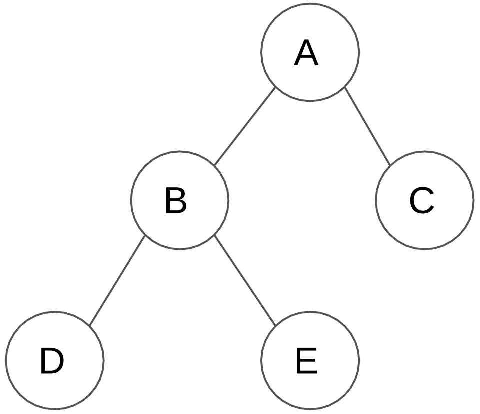
Linked List Representation of binary tree





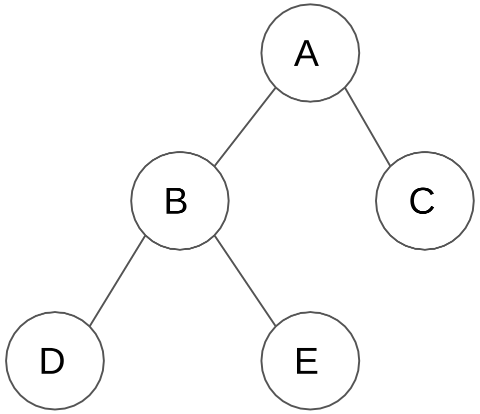
Siblings

- Siblings: B and C are said to be left and right child of A, so B and C are known as siblings.
- All nodes at same level and share the same parent are known as siblings.



Level Number

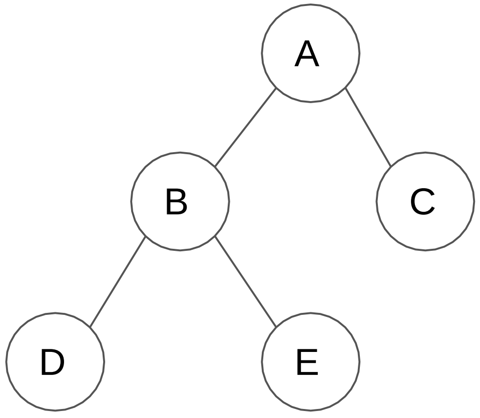
- Every node in a binary tree is having Level Number.
- Root node is defined at level 0.
- Left and Right child of Root node is at level 1.
- So, a child's level number is defined as parent's level number + 1.



Degree

- Degree of a node is equal to the number of children it has.
- Degree of leaf node is always zero.

- Degree of A =
- Degree of B =
- Degree of C =
- Degree of D =
- Degree of E =



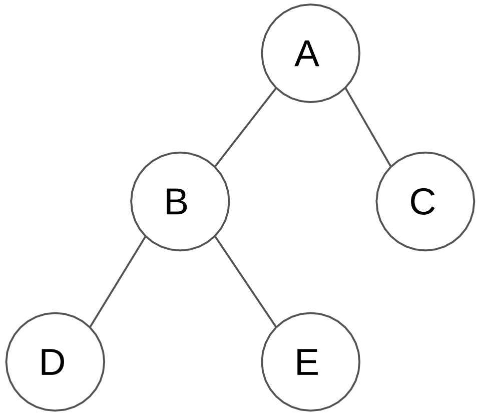
Depth and Height

- The **depth** of a node is the number of edges present in path from the root node of a tree to that node.

- The **height** of a node is the number of edges present in the longest path connecting that node to a leaf node.

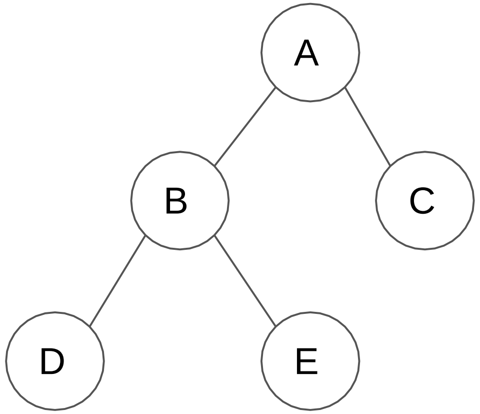
- Depth of B = 1
- Height of B = 1

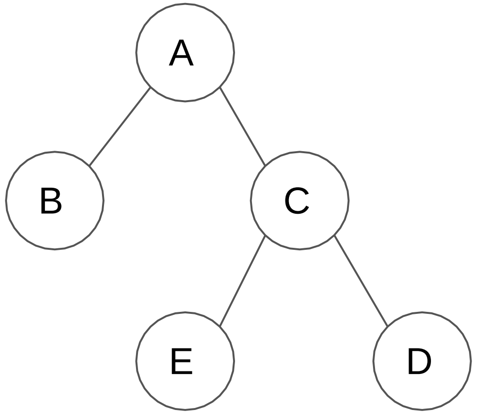
- Depth of D = 2
- Height of D = 0



Strictly Binary Trees

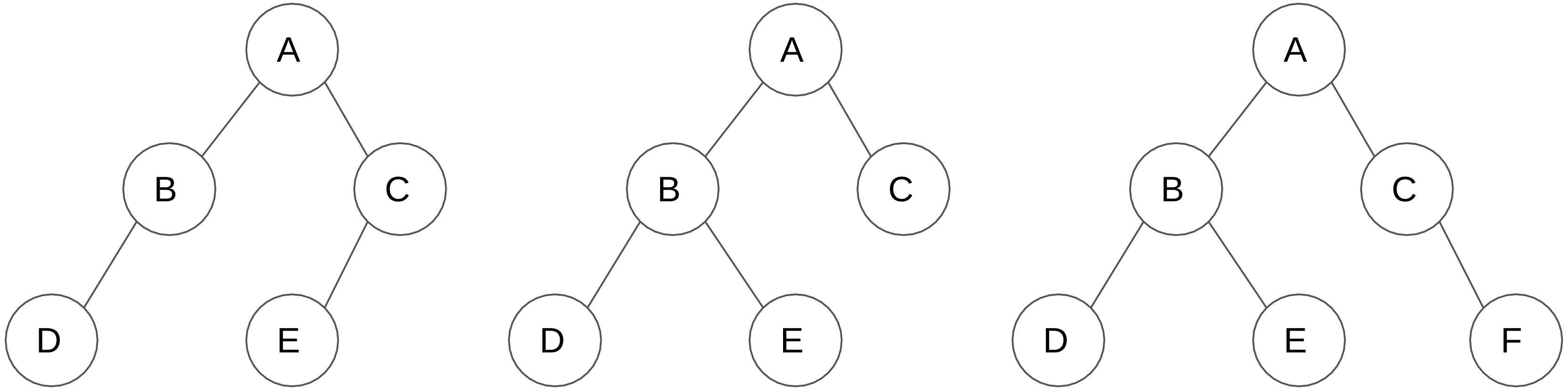
- If every non leaf node in a binary tree has nonempty left and right subtrees, then tree is called a strictly binary tree.
- A strictly binary tree with n leaves always contains 2n-1 nodes.
- A **strictly binary tree** is a tree in which every node other than the leaves has two children.



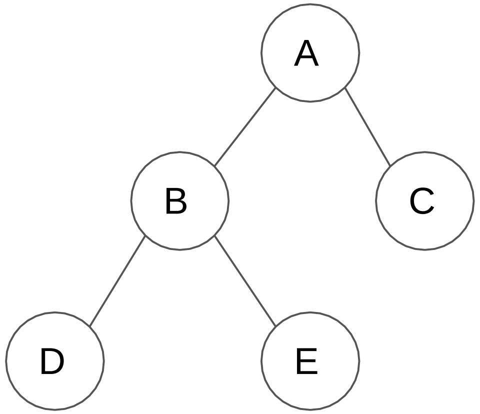


Complete Binary Tree

- A binary tree T with n levels is complete if all levels except possibly the last are completely full, and the last level has all its nodes to the left side.



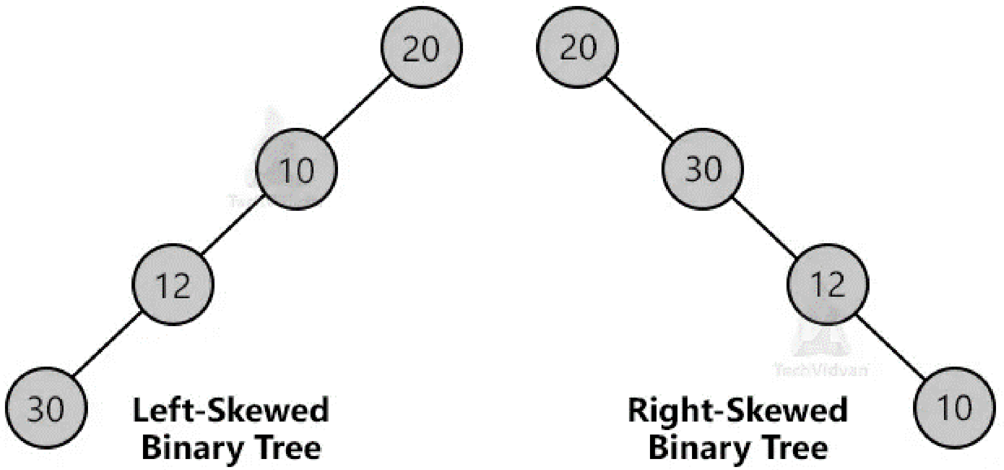
Complete Binary Tree



- Max #of Nodes at Level(I):
  - Level 0 has 2^0 = 1 Node
  - Level 1 has 2^1 = 2 Node
  - Level 2 has 2^2 = 4 Node

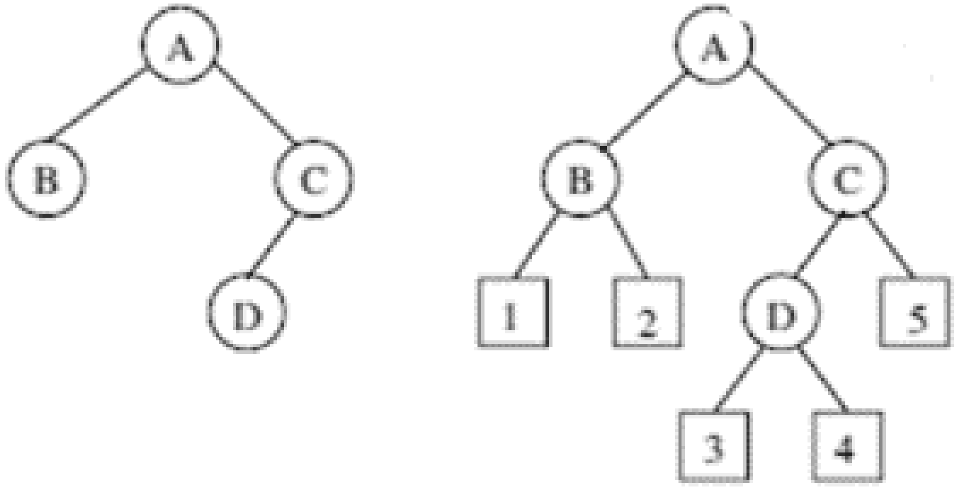
Skewed Binary Tree

- A **skewed binary tree** could be skewed to left or right. In left skewed, most of the nodes have left child without corresponding right child. Similarly, for the right skewed.



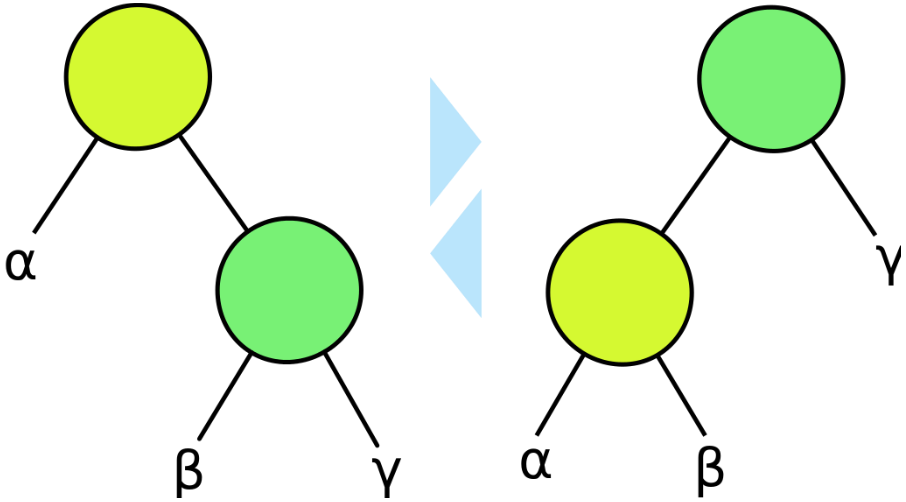
Extended Binary Tree

- In extended **binary tree** each sub tree is replaced by a failure node. A failure node is represented as □
- Any binary tree can be converted into a extended binary tree by replacing each empty subtree by a failure node.



Weight Balanced Tree

- If the ratio of the weight of the left subtree of every node to the weight of the subtree rooted at the node is between a and 1-a then the tree is WBT of ratio a.
- Weight-balanced binary trees (WBTs) are a type of self-balancing binary search trees that can be used to implement dynamic sets, dictionaries (maps) and sequences.



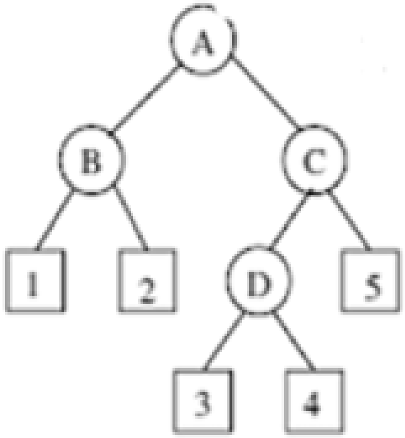
Types of Binary tree

- A **strictly binary tree** is a tree in which every node other than the leaves has two children.
- A **perfect binary tree** is a **full** binary tree in which all leaves are at the same depth or same level. 2 or no children. Node $= 2^{h+1} - 1$.
- A **complete binary tree** is a binary tree in which every level, except possibly the last, is completely filled, and all nodes are as far left as possible.
- A **skewed binary tree** is either left or right skewed.
- A **extended binary tree**'s subtree is replaced by failure node.
- A **weight balanced tree** self balancing tree. Rotations are applied to restore weight balance.

Examples of Full, Complete, Perfect, Extended Binary tree

```
        18
       /  \
     15    20
    /  \
  40    50
 /  \
30   50
```

```
          18
        /    \
      15      30
     /  \    /  \
   40   50  100  40
  / \   /
 8   7 9
```



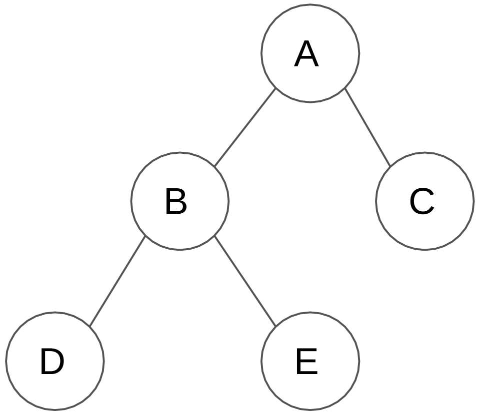

```
        18
      /    \
    15      30
   /  \    /  \
 40   50  100  40
```

Binary Tree Traversal

- Often, one wishes to visit each of the nodes in a tree and examine the value there, a process called Traversal.
- There are several common orders in which the nodes can be visited, and each has useful properties that are exploited in algorithms based on binary trees.
  1. **Pre-Order:** Root, Left child, Right child
  2. **In-Order:** Left child, Root, Right child
  3. **Post-Order:** Left Child, Right child, Root

Pre-order Traversal

- **Pre-Order:** Root, Left child, Right child



A, B, D, E, C

Pre-order Algorithm

PREORDER(TREE):

Step 1: IF TREE != NULL perform step 2, 3 and 4

Step 2: Write "TREE->DATA"

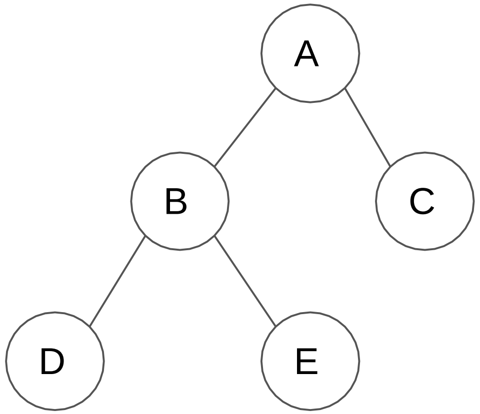
Step 3: PREORDER(TREE->LEFT)

Step 4: PREORDER(TREE->RIGHT)

Step 5: END

In-order Traversal

- **In-Order:** Left child, Root, Right child



D, B, E, A, C

In-order Algorithm

INORDER(TREE):

Step 1: IF TREE != NULL perform step 2, 3 and 4

Step 2: INORDER(TREE->LEFT)

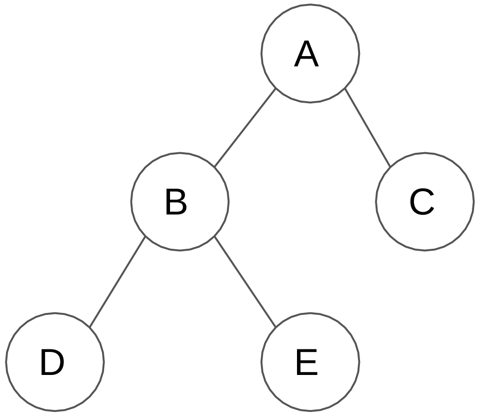
Step 3: Write "TREE->DATA"

Step 4: INORDER(TREE->RIGHT)

Step 5: END

Post-order Traversal

- **Post-Order:** Left child, Right child, Root



D, E, B, C, A

Post-order Algorithm

POSTORDER(TREE):

Step 1: IF TREE != NULL perform step 2, 3 and 4

Step 2: POSTORDER(TREE->LEFT)

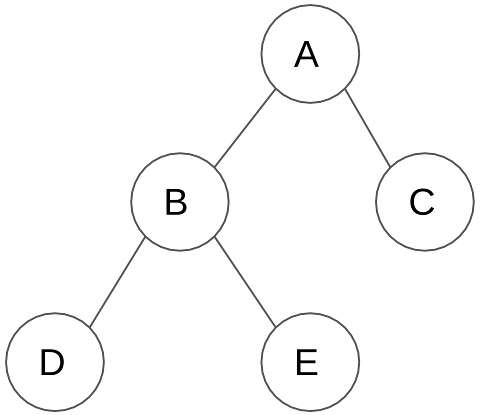
Step 3: POSTORDER(TREE->RIGHT)

Step 4: Write “TREE->DATA”

Step 5: END

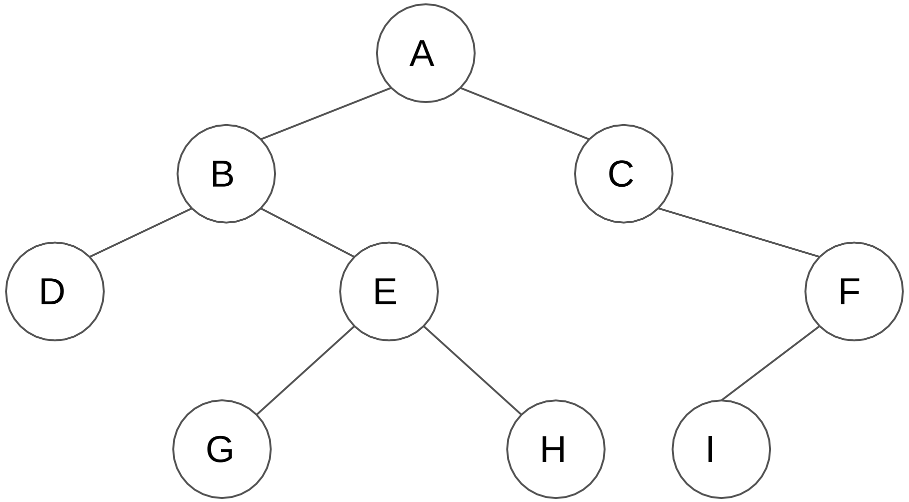
Level-order traversal

- In Level-order traversal, all the nodes at a level are accessed before going to the next level.
- This is also known as breadth-first traversal algorithm.



A, B,C, D, E

Different Traversals



- Depth-first:
  - Pre-order (root, left, right):
  - In-order (left, root, right):
  - Post-order (left, right, root):
- Breadth-first:
  - Level-order:

Traversal Examples

Construction of Tree from Travarsal

- Inorder sequence: D B E A F C (Left, root, right)
- Preorder sequence: A B D E C F (Root, Left, Right)

Construction of Tree from Travarsal

- Preorder: 1, 2, 4, 8, 9, 10, 11, 5, 3, 6, 7
- Inorder: 8, 4, 10, 9, 11, 2, 5, 1, 6, 3, 7

Construction of Tree from Travarsal

- Inorder: 7 4 2 5 1 8 6 9 3
- Postorder: 7 4 5 2 8 9 6 3 1

Construct Tree

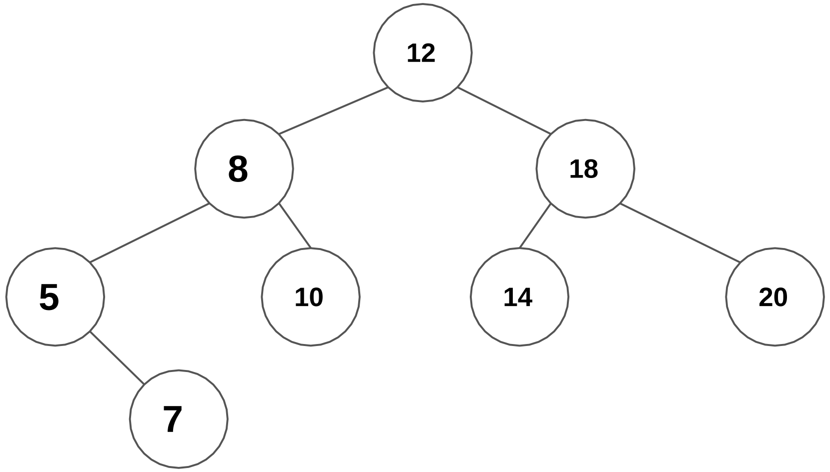
- Exercise:
  - Inorder: 15, 30, 35, 40, 45, 50, 60, 70, 72, 75, 77, 80
  - Preorder: 50, 30, 15, 40, 35, 45, 70, 60, 80, 75, 72, 77

Binary Search Tree

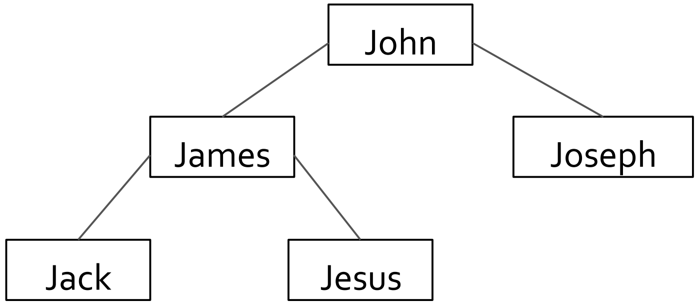
- A binary tree which conforms to the following properties is called a binary search tree.
- Properties:
  - Each value (key) in the tree exists at most once (i.e. no duplicates).
  - The "greater-than" and "less-than" relations are well defined for the data value.
  - Sorting constraints:- for every node n:
    - All data in the **left subtree** of n is **less** than the data in the root of that subtree.
    - All data in the **right subtree** of n is **greater** than the data in the root of that subtree.

Binary Search Tree

- Trees (with some ordering e.g., BST) provide moderate access/search (quicker than Linked List and slower than arrays).
- Trees provide moderate insertion/deletion (quicker than Arrays and slower than Unordered Linked Lists).
- Like Linked Lists and unlike Arrays, Trees don't have an upper limit on number of nodes as nodes are linked using pointers.



BST of Numbers



BST of Names

Operations on BST - Search

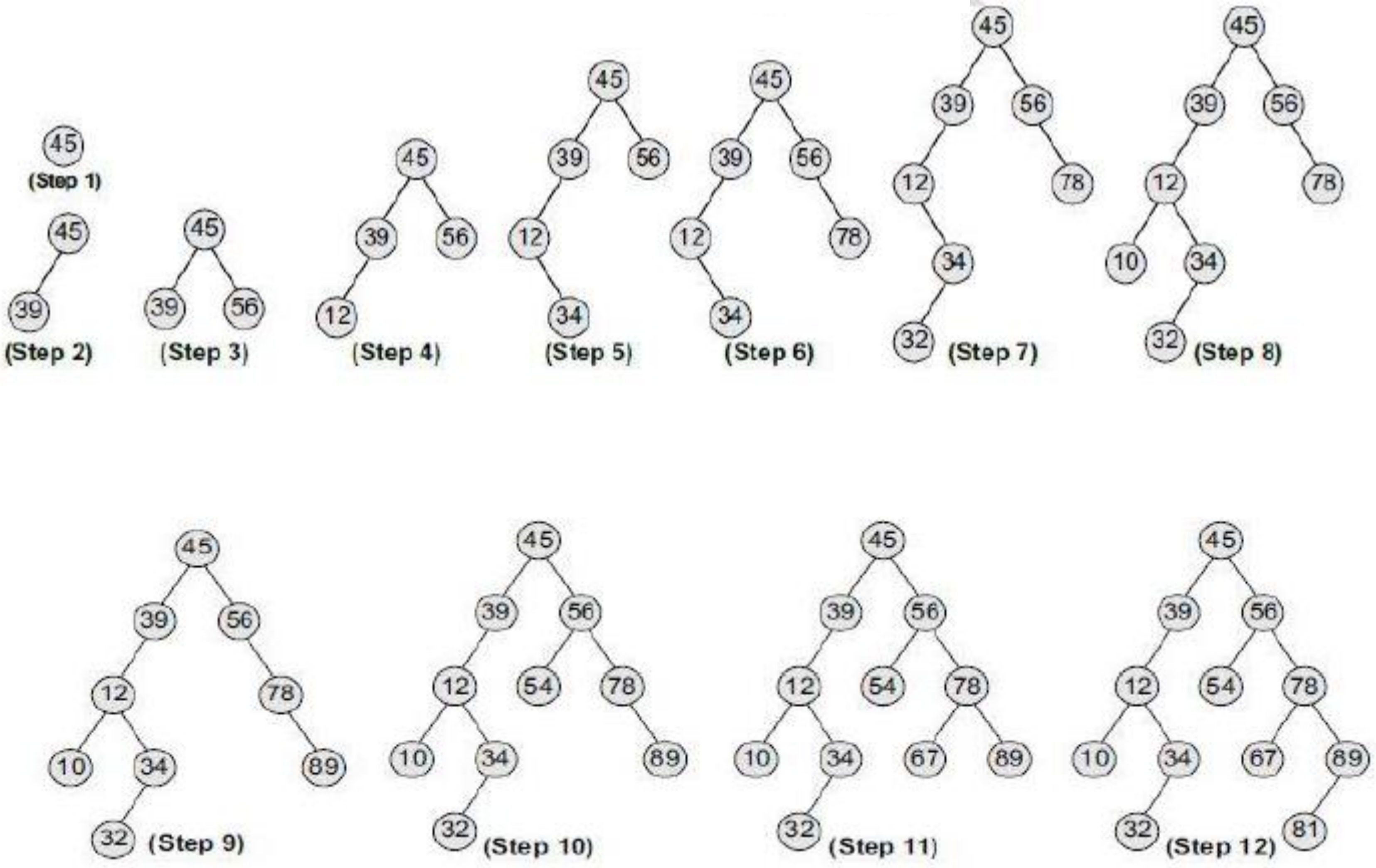
```
SearchBST(TREE)

IF TREE->DATA = VAL OR TREE = NULL
THEN Return TREE
ELSE IF VAL < TREE->DATA
    Return SearchBST(TREE->LEFT, VAL)
ELSE
    Return SearchBST(TREE->RIGHT, VAL)
```

Binary Search Tree

- Create a BST with following elements.
- 45, 39, 56, 12, 34, 78, 32, 10, 89, 54, 67, 81

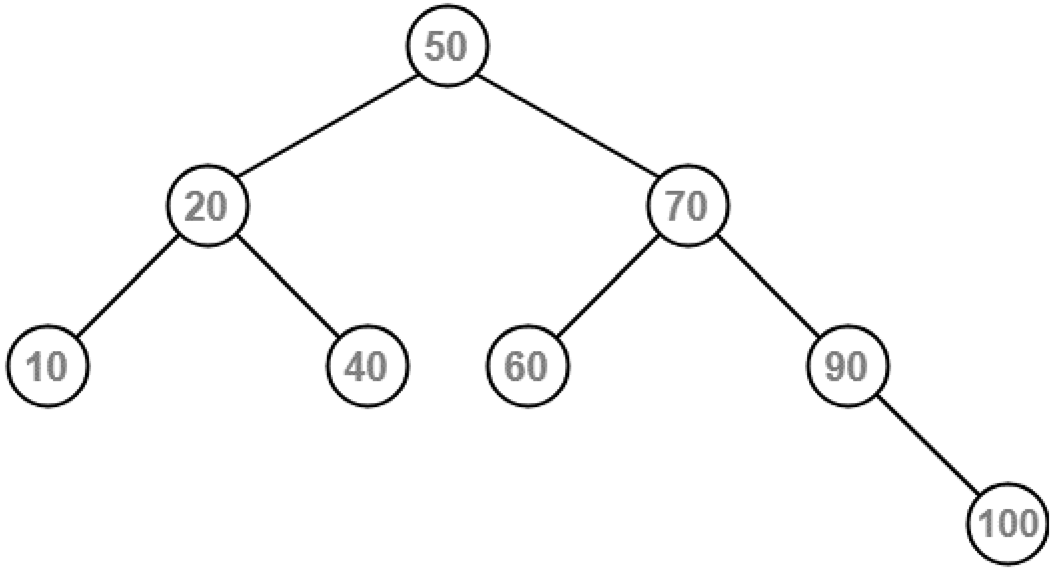
Binary Search Tree



Binary Search Tree

- Create a BST with following elements.
- 50, 70, 60, 20, 90, 10, 40, 100

Binary Search Tree



Operations in BST

BST Operations:

1. Initialize
2. Find
3. Make empty
4. Insert
5. Delete
6. Create
7. Find min
8. Find max

Operations on BST - Insert

```
InsertBST(TREE)

IF TREE = NULL THEN
    Allocate memory for TREE
    SET TREE -> DATA = VAL
    SET TREE -> LEFT = TREE -> RIGHT = NULL
    Return TREE
ELSE IF VAL < TREE -> DATA
    TREE->LEFT = InsertBST(TREE->LEFT, VAL)
ELSE
    TREE->RIGHT = InsertBST(TREE->RIGHT, VAL)
```

Operations on BST - Delete

- CASE 1: Delete a node that has no children
- CASE 2: Delete a node that has one child
- CASE 3: Delete a node that has two children
  - In order predecessor- Find largest in left subtree & replace
  - In order successor- Find smallest in right subtree & replace

Operations on BST - Delete

- CASE 1: Delete a node that has no children

Operations on BST - Delete

- CASE 2: Delete a node that has one child

Operations on BST - Delete

- CASE 3: Delete a node that has two children

Operations on BST - Delete

```
DeleteBST(TREE)

IF TREE = NULL THEN
    Write 'Value not found'
ELSE IF VAL < TREE -> DATA THEN
    TREE->LEFT = DeleteBST(TREE->LEFT, VAL)
ELSE IF VAL > TREE -> DATA THEN
    TREE->RIGHT = DeleteBST(TREE->RIGHT, VAL)
ELSE IF TREE->LEFT=NULL AND TREE->RIGHT=NULL
THEN
    free(TREE)
    Return NULL
```

Operations on BST - Delete

```
DeleteBST(TREE)

ELSE IF TREE->RIGHT=NULL THEN
    tmp=TREE->LEFT
    free(TREE)
    Return tmp
ELSE IF TREE->LEFT=NULL THEN
    tmp=TREE->RIGHT
    free(TREE)
    Return tmp
ELSE IF TREE->LEFT!=NULL AND TREE->RIGHT!=NULL
THEN
    tmp=getPreDec(TREE);
    TREE->DATA=tmp;
    TREE->LEFT=DeleteBST(TREE->LEFT,tmp);
    Return TREE;
```

Binary Search Tree Complexities

- Time Complexity
  - For Insertion, deletion, search the best and average case complexity is O(log n)
  - Worst case is O(n)
- Space Complexity
  - For all operations is O(n)
  - n is number of nodes in the tree.

BST Task

1. Construct BST for the following data:

10, 3, 15, 22, 6, 45, 65, 23, 78, 34, 5
Traverse the final tree in Pre-order, In-order, Post-order

2. Insert the following in BST

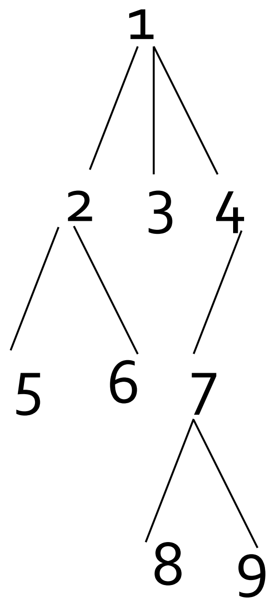
7, 39, -2, 0, 3, 42, 20, 5, 40
Perform deletion operation

General Trees

- General trees are those in which the number of subtrees for any node is not required to be 0, 1, or 2.
- The tree may be highly structured and therefore have 3 subtrees per node in which case it is called a ternary tree.
- However, it is often the case that the number of subtrees for any node may be variable. Some nodes may have 1 or no subtrees, others may have 3, some 4, or any other combination.

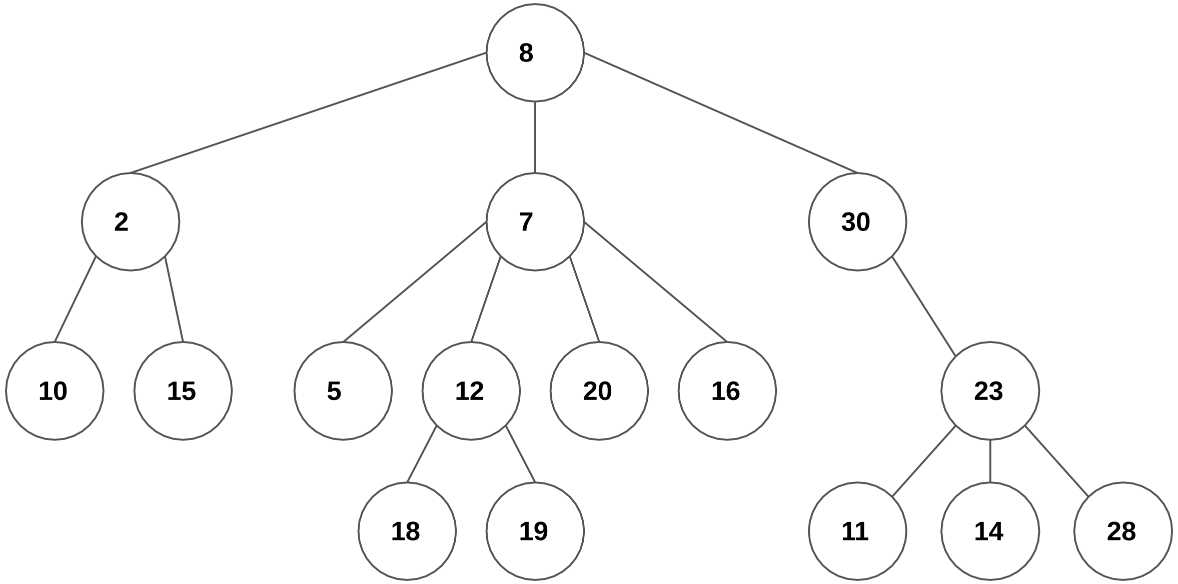
General Trees

- Root of binary tree BT = Root of general tree GT
- Left child of node in BT = Leftmost child of the node in GT
- Right child of node in BT = Right sibling of the node in GT



General Trees

Convert General Tree to Binary Tree



Threaded Binary Tree

- Threaded binary tree is a simple binary tree but they have a speciality that null pointers of leaf node of the binary tree is **set to inorder predecessor or inorder successor.**
- The main idea behind setting such a structure is to make the inorder and preorder traversal of the tree faster without using any additional data structure(e.g auxilary stack) or memory to do the traversal.
- A Threaded Binary Tree is a binary tree in which every node that does not have a right child has a THREAD (in actual sense, a link) to its INORDER successor.
- By doing this threading we avoid the recursive method of traversing a Tree, which consumes a lot of memory and time.

Types of Threaded Binary Tree

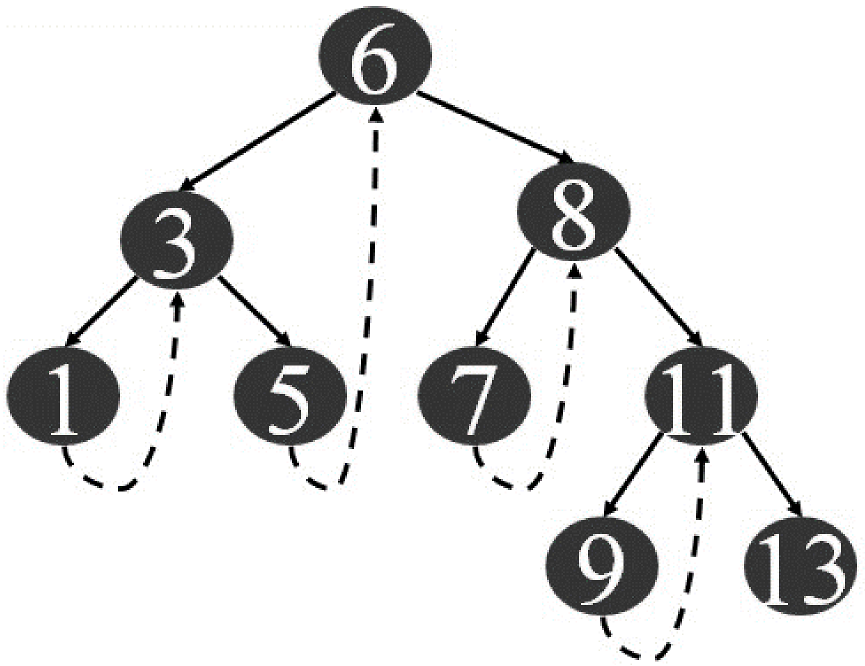
- Single Threaded Binary Tree
- Double Threaded Binary Tree

- Single Threaded Binary Tree: Here only the right NULL pointer are made to point to inorder successor.

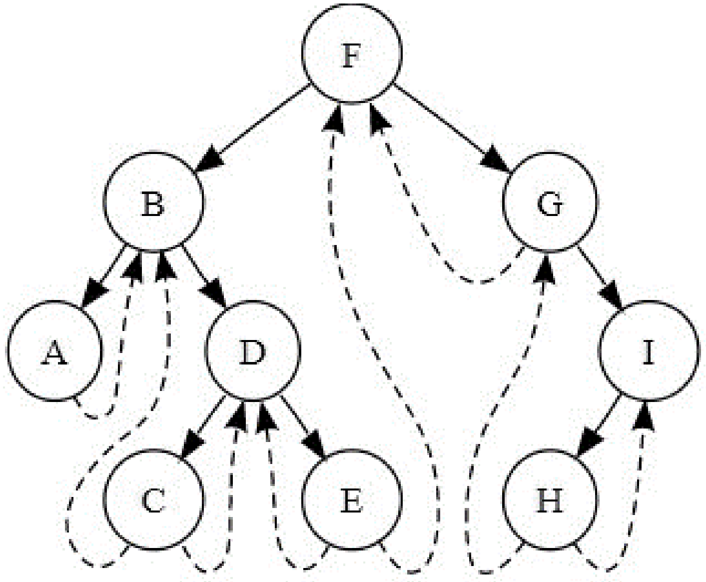
- Double Threaded Binary Tree: Here both the right as well as the left NULL pointers are made to point inorder successor and inorder predecessor respectively. (here the left threads are helpful in reverse inorder traveral of the tree)

Types of Threaded Binary Tree

Left Thread Flag	Left Link	Data	Right Link	Right Thread Flag



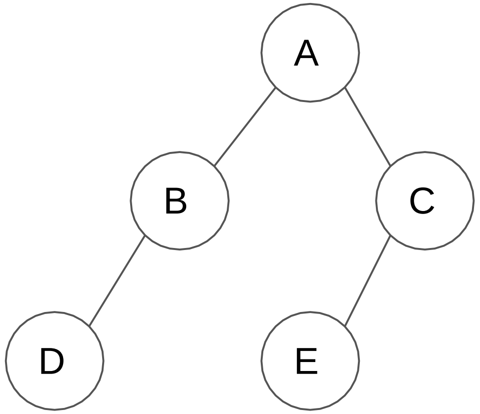
Single Threaded Binary Tree



Double Threaded Binary Tree

Threaded Binary Tree

- Let's make the Threaded Binary Tree out of a normal binary tree:

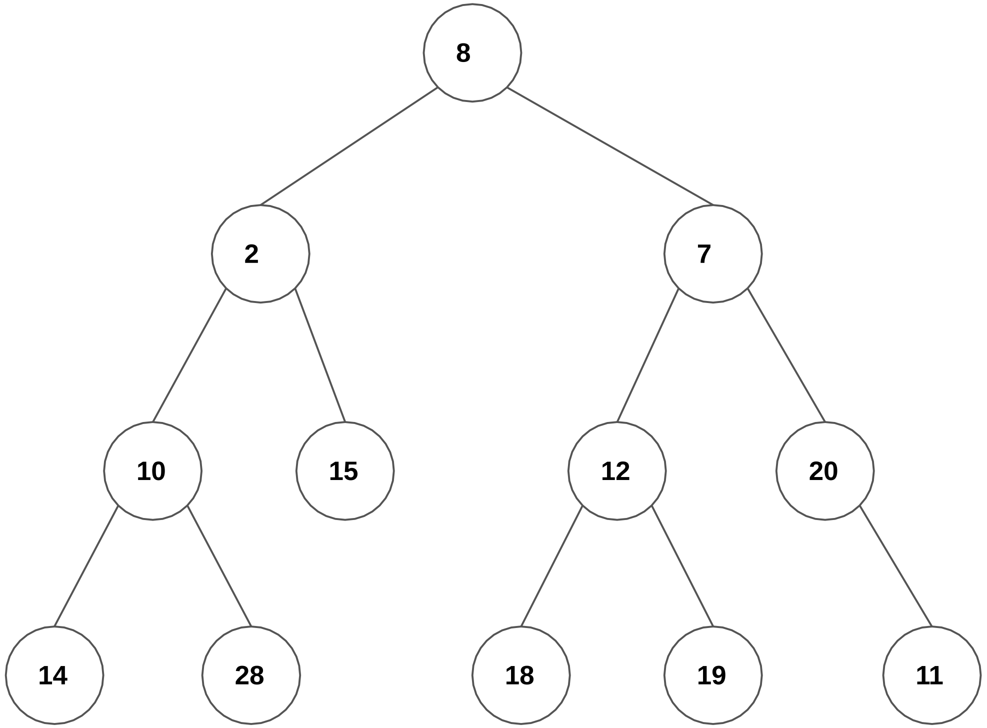


Threaded Binary Tree

- Advantage
  - By doing threading we can avoid a lot of memory and time caused in recursion.
  - The node can keep record of its root .
- Disadvantage
  - This makes the Tree more complex.
  - More prone to errors.

Threaded Binary Tree

- One more example:



Application of Trees

- **Expression Trees**
- Binary Trees are widely used to store algebraic expressions.
- Binary tree is a tree in which all nodes contain zero, one or two children.
- The expression trees have been implemented as binary trees mainly because binary trees allows you to quickly find what you are looking for.
- Expression Tree example: (a-b)+(c*d)

Application of Trees

- **Tournament Trees**
- In tournament of Chess or Cricket, n number of players participate.
- To declare a winner among them, a couple of matches are played.
- If there are 8 players then Round 1 will be having 4 matches and 4 winners.
- Within Round 2, 2 matches and 2 winners.
- At Round 3, 1 match and 1 winner as root node of our tree.

Application of Trees

- Manipulate hierarchical data.
- Make information easy to search (see tree traversal).
- Router algorithms
- Trees can hold objects that are sorted by their keys.

Till Now

- **Trees**
- Graphs

- **Tree definitions and their concepts**
- **Representation of binary tree**
- **Binary tree traversal**
- **Binary search trees**
- **General trees vs binary trees**
- **Threaded binary tree**
- **Applications of Trees**
- Balanced tree and its mechanism
- Height and Weight Balanced Trees

Up Next

- **Trees**
- Graphs

- Tree definitions and their concepts
- Representation of binary tree
- Binary tree traversal
- Binary search trees
- General trees vs binary trees
- Threaded binary tree
- Applications of Trees
- **Balanced tree and its mechanism**
- **Height and Weight Balanced Trees**

Thank You.