BITS Pilani
Hyderabad Campus

S4 Non-Linear Data Structures: Tree and Binary Tree

Dr. Rajib Ranjan Maiti
CSIS Dept, Hyderabad Campus

innovate achieve lead

BITS Pilani
Hyderabad Campus

Data Structures and Algorithms Design (Merged-SEZG519/SSZG519)

S4 Tree and Binary Tree

Content of L-3

3.1. Trees
- 3.1.1. Terms and Definition
- 3.1.2. Tree ADT
- 3.1.3. Applications

3.2. Binary Trees
- 3.2.1. Properties
- 3.2.2. Representations (Vector Based and Linked)
- 3.2.3. Binary Tree traversal (In Order, Pre Order, Post Order)
- 3.2.4. Applications

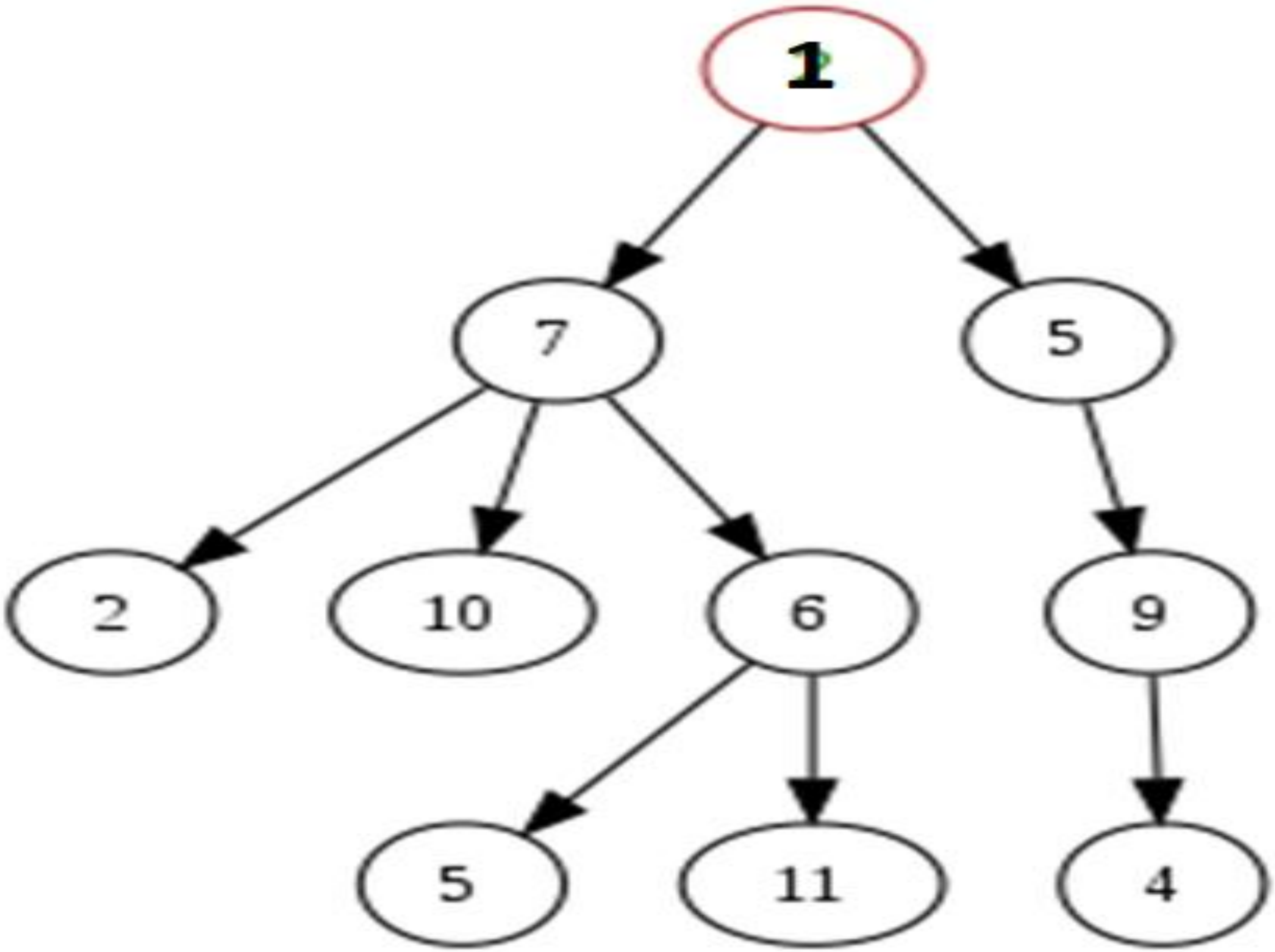
So far we studied

- Linear data structure:
    - A type of data structure where elements are arranged in a sequential order, and each element has a unique predecessor and successor (except for the first and last elements).
- E.g.: Array, Linked List, Stack, Queue

Non-linear data structure

- A non-linear data structure
  - A type of data structure in which elements are not organized sequentially, but can be connected in various ways.
- E.g. Tree, Graph

Tree



Basic terminologies

- Node:
  - *Each element in the tree is a node. Each node can have child nodes connected to it.*
- Parent and child:
  - *A node that is directly connected to another node is considered the parent of that node. Conversely, the connected node is the child of the parent node.*
  - *E.g. 7 is parent node, 2,10,6 are its child nodes.*

Basic terminologies

- Leaf node:
  - *A node with no children is called a leaf node. e.g., 2,10,5,11,4 are its leaf nodes.*
- Root:
  - *A node which has no parent is called a root node. e.g., 1 is the root node.*

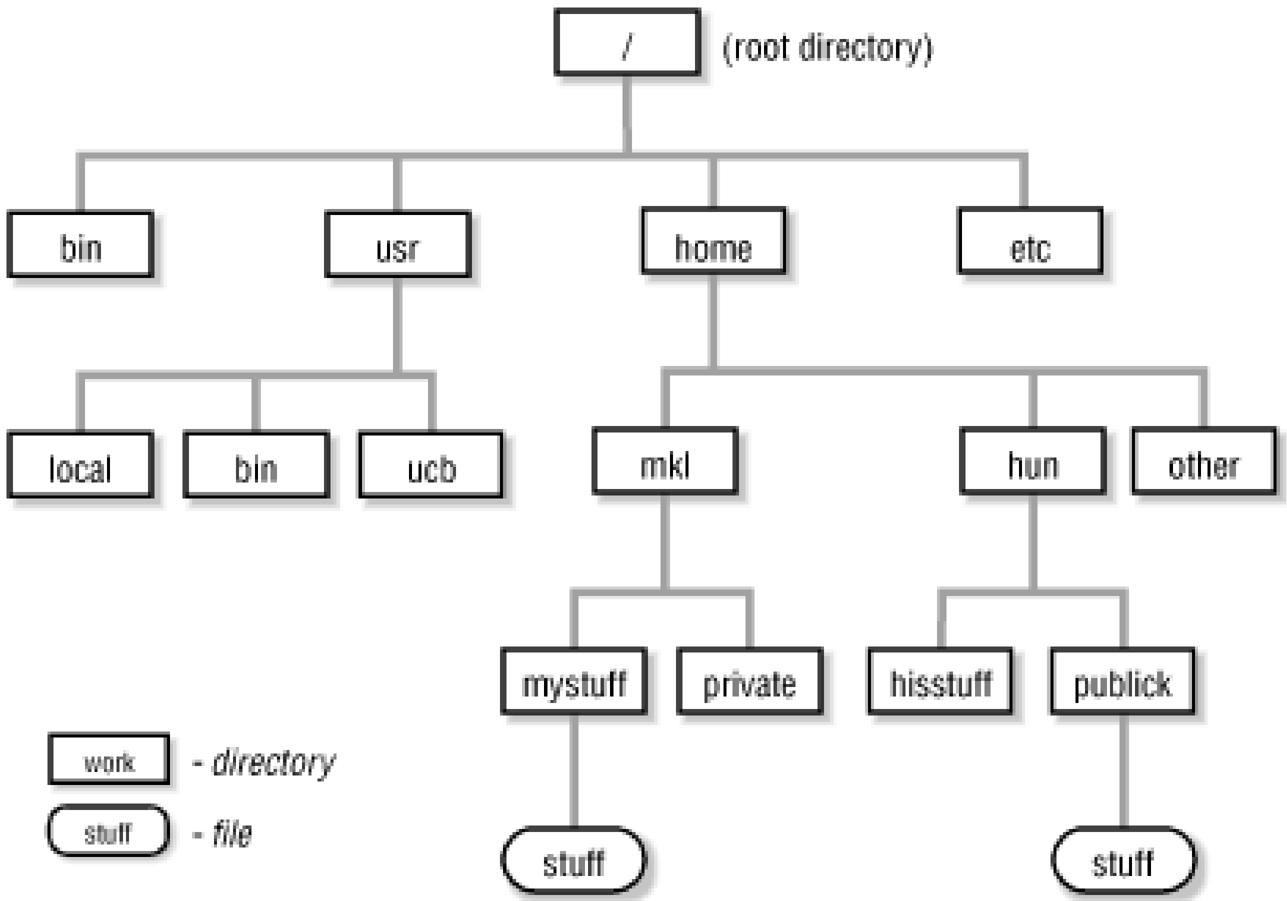
Basic terminologies

- Level:
  - *Level is the rank of the hierarchy and root node is termed as in level 0. If a node is at level x, then its parent is at level x-1 and its child nodes are at level x+1. e.g. Node 10 is at level 2.*
- Height:
  - *Maximum number of nodes that is possible in a path from root node to a leaf node is the height of a tree. e.g. Height of tree in example = 4*
- Degree:
  - *Maximum number of child nodes possible for a node is called degree. e.g., Degree is 3 in example.*
- Sibling:
  - *The nodes which have the same parent are called siblings. E.g. 2, 10, and 6 are siblings.*

Basic terminologies

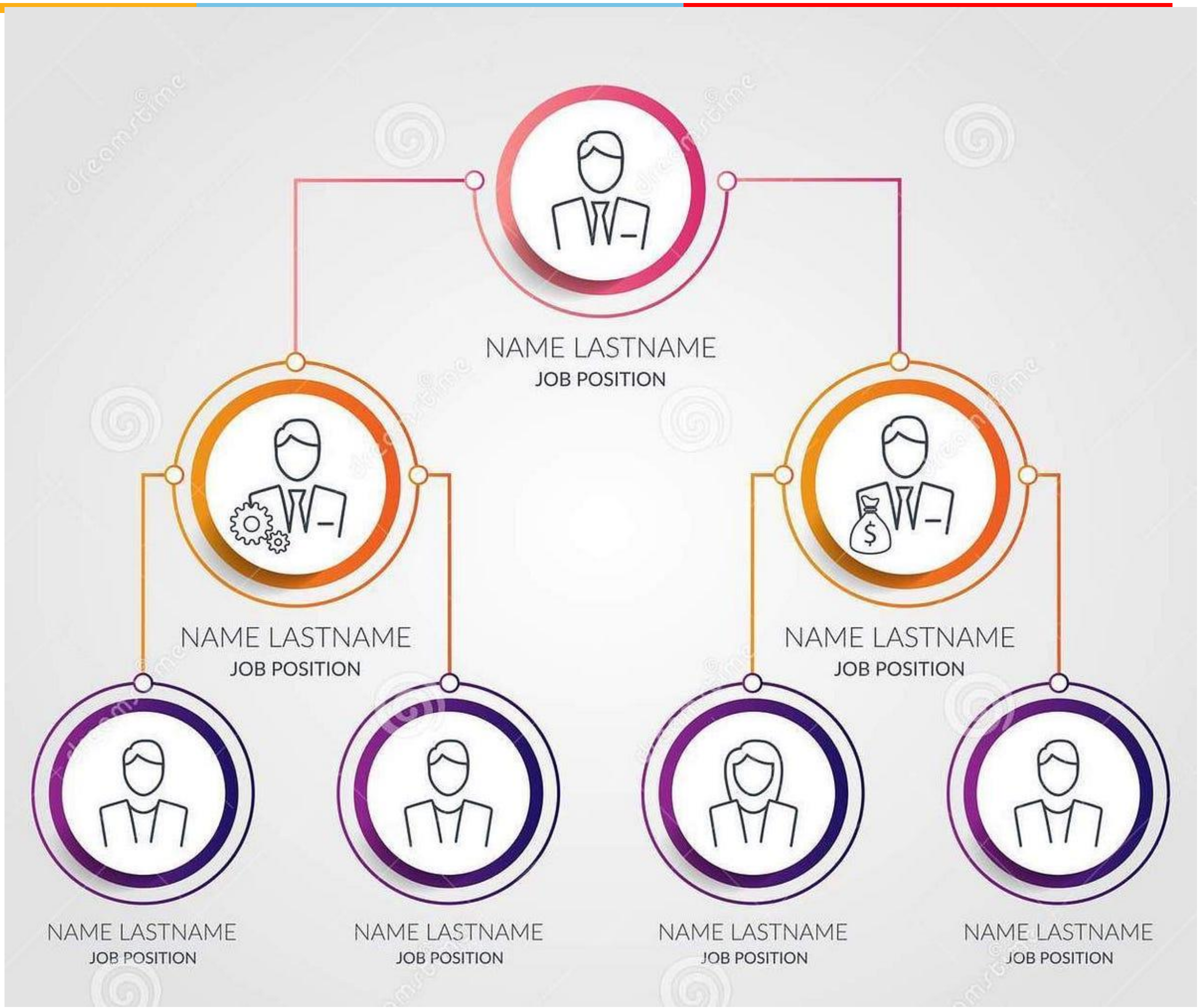
- Level:
  - *Level is the rank of the hierarchy and root node is termed as in level 0. If a node is at level x, then its parent is at level x-1 and its child nodes are at level x+1. e.g. Node 10 is at level 2.*
- Height:
  - *Maximum number of nodes that is possible in a path from root node to a leaf node is the height of a tree. e.g. Height of tree in example = 4*
- Degree:
  - *Maximum number of child nodes possible for a node is called degree. e.g., Degree is 3 in example.*
- Sibling:
  - *The nodes which have the same parent are called siblings. E.g. 2, 10, and 6 are siblings.*

Applications of tree data structure



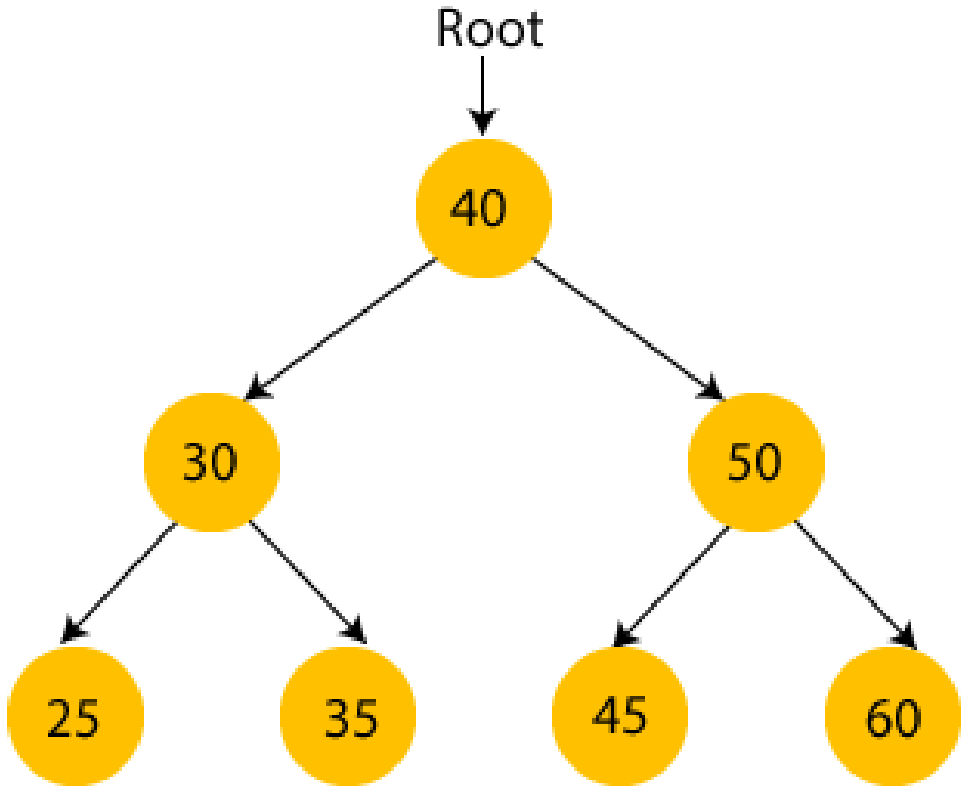
Representation of File systems

Applications of tree data structure



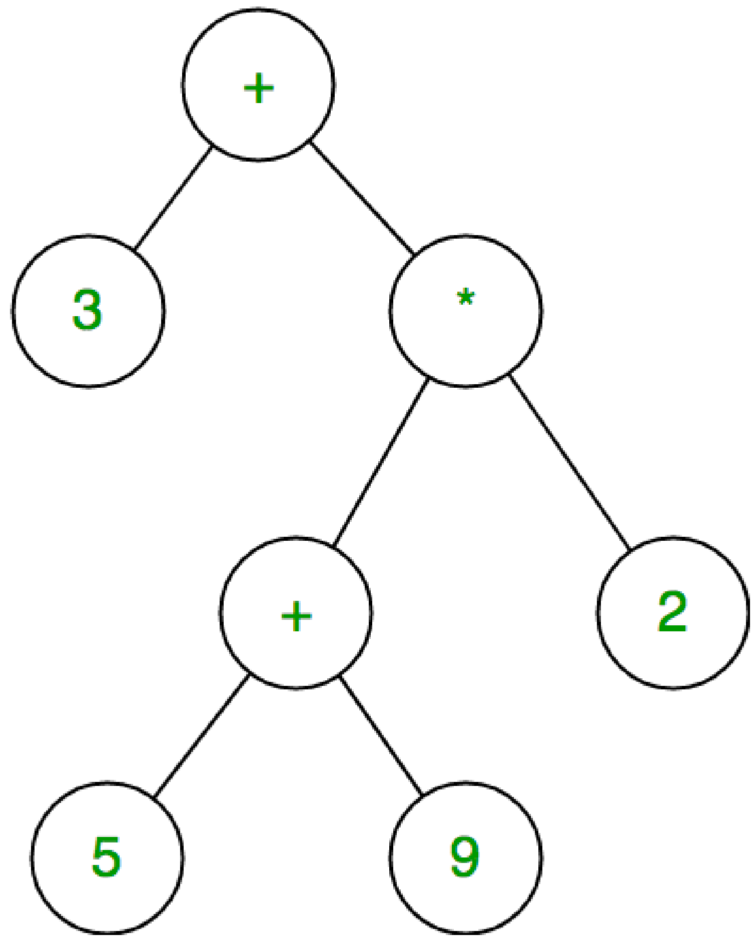
Representation organizational hierarchies

Applications of tree data structure



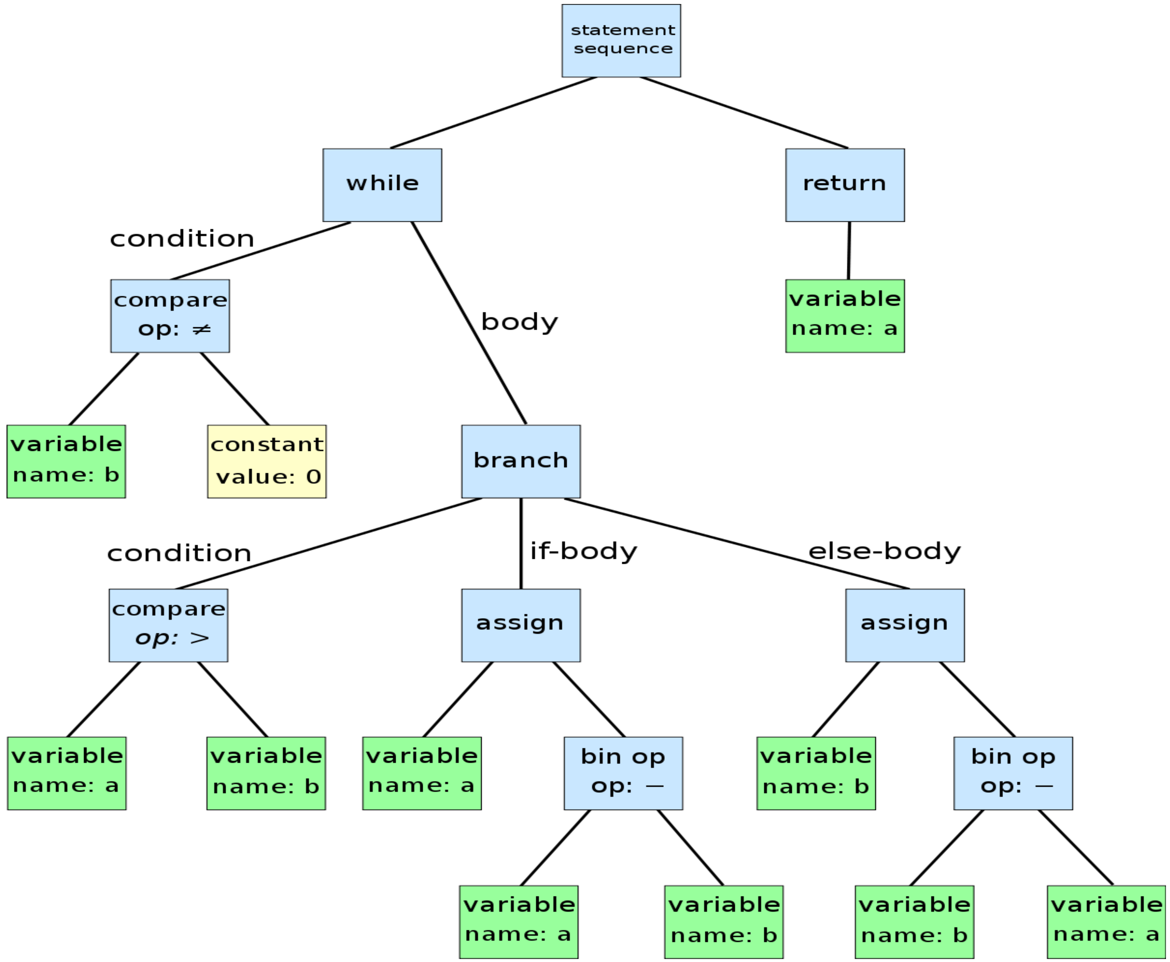
Binary Search Tree for efficient search

Applications of tree data structure



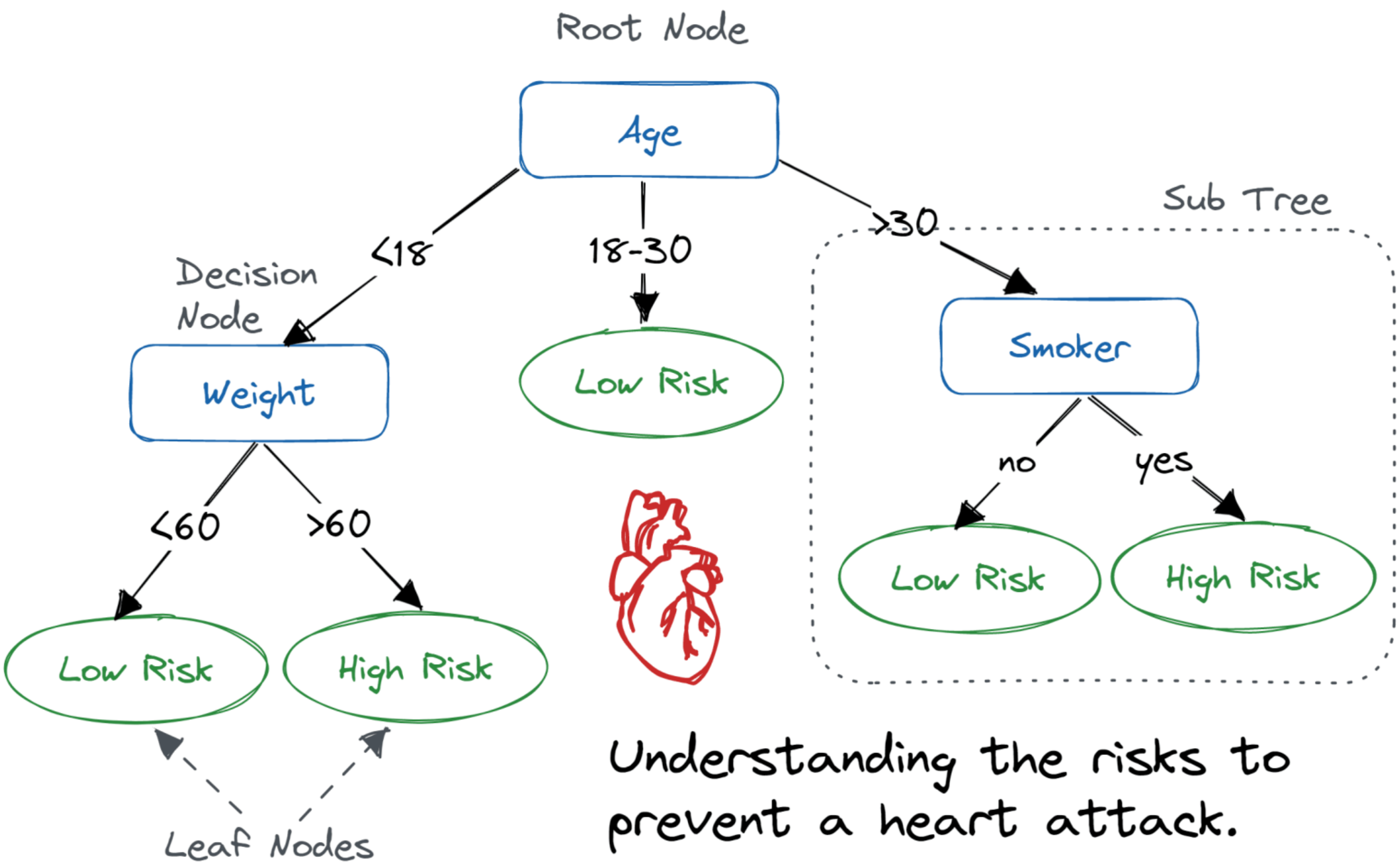
Tree for parsing and evaluation of expression

Applications of tree data structure



Representation of the syntax of source code

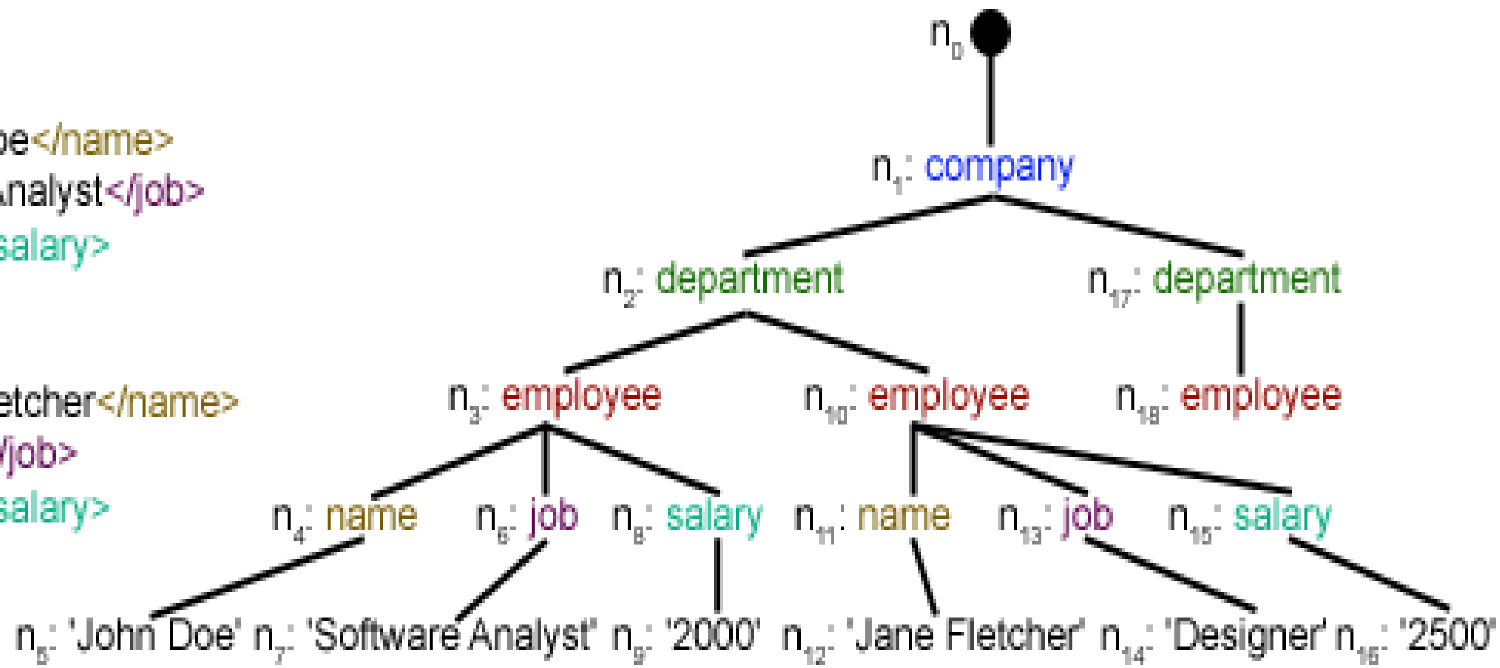
Applications of tree data structure



Decision tree

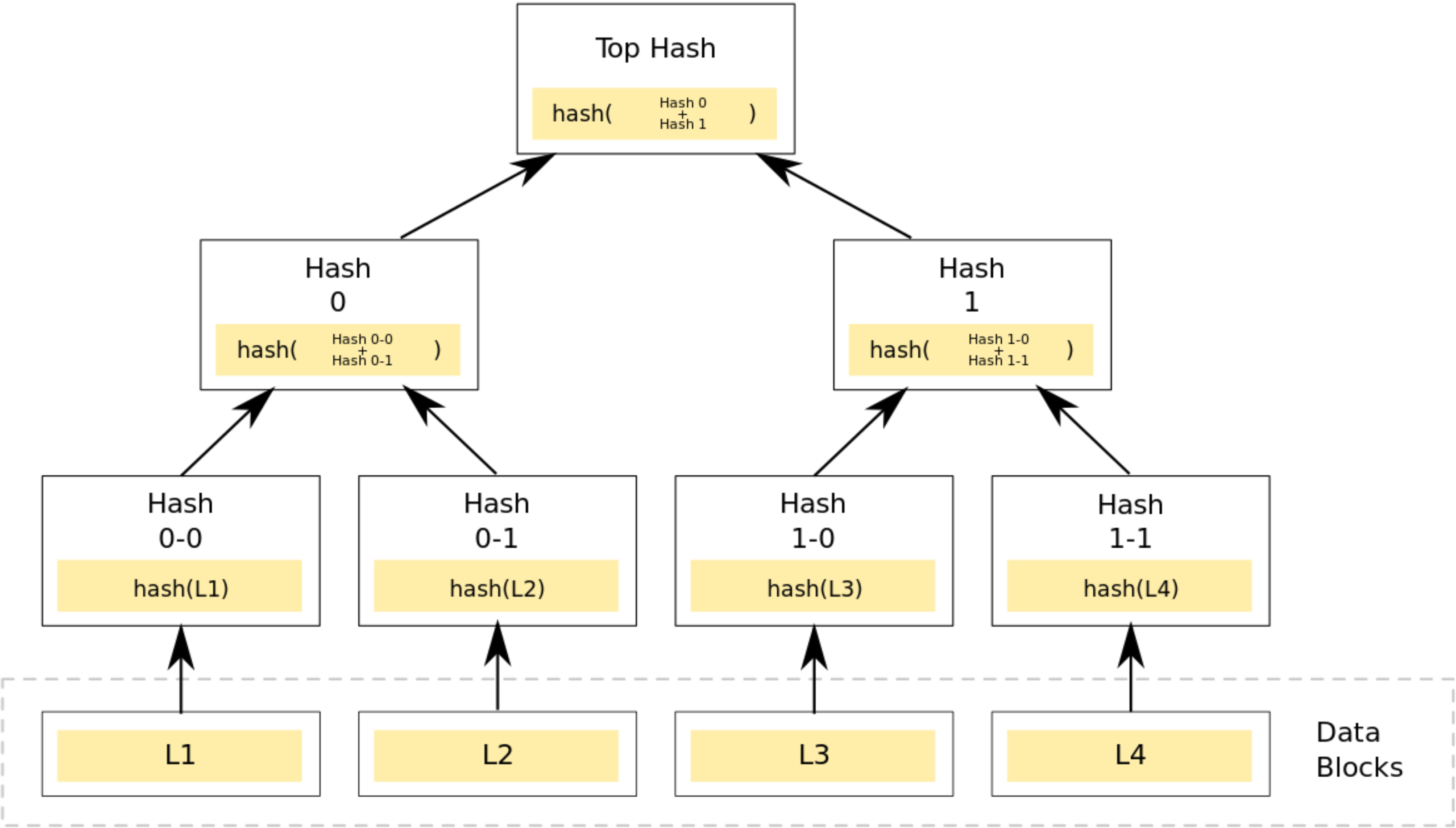
Applications of tree data structure

```
<company>
  <department>
    <employee>
      <name>John Doe</name>
      <job>Software Analyst</job>
      <salary>2000</salary>
    </employee>
    <employee>
      <name>Jane Fletcher</name>
      <job>Designer</job>
      <salary>2500</salary>
    </employee>
  </department>
  <department>
    <employee>
    </employee>
  </department>
</company>
```



XML Parsing Tree

Applications of tree data structure



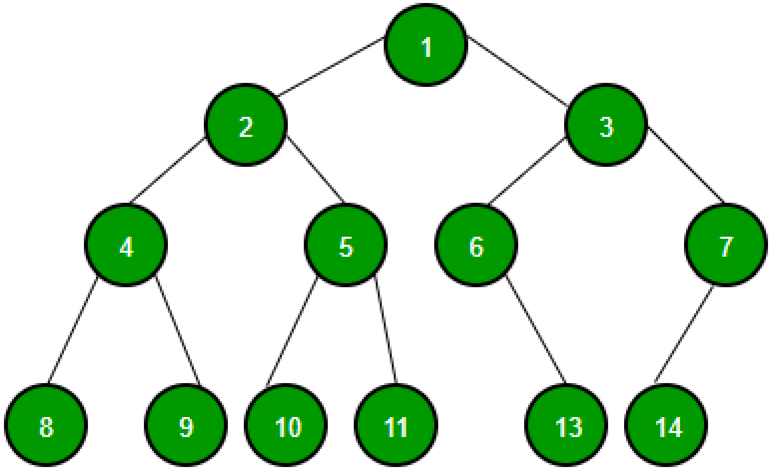
Merkle tree in Blockchain

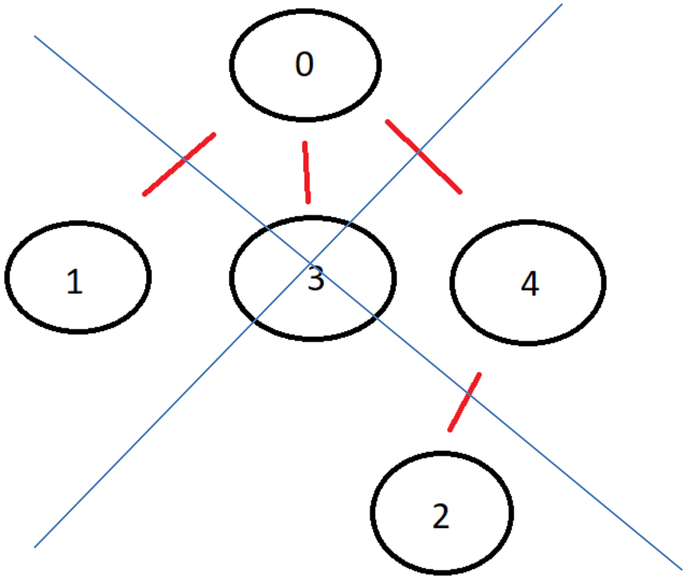
Types of Tree

- Binary Tree
- Binary Search Tree
- Heap tree
- AVL Tree
- B-Tree
- Red-black Tree

Binary Tree

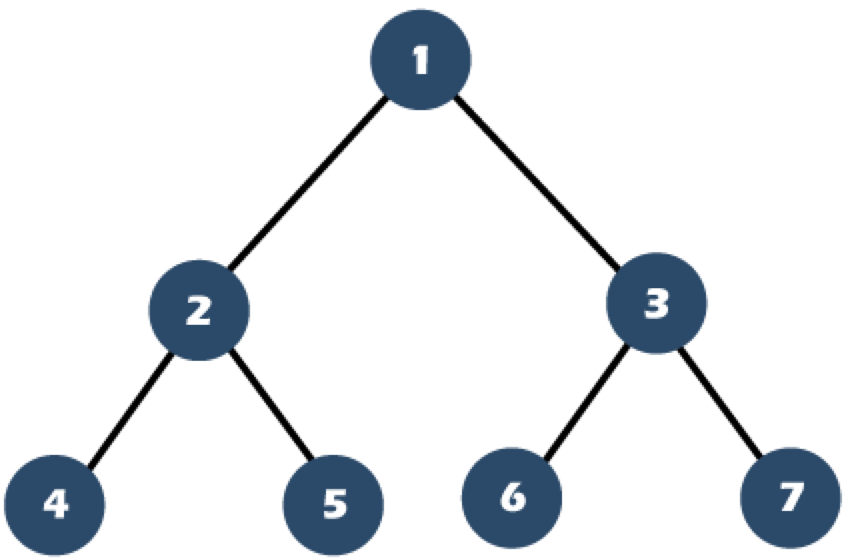
- A binary tree is a tree structure in which each node has at most two children.

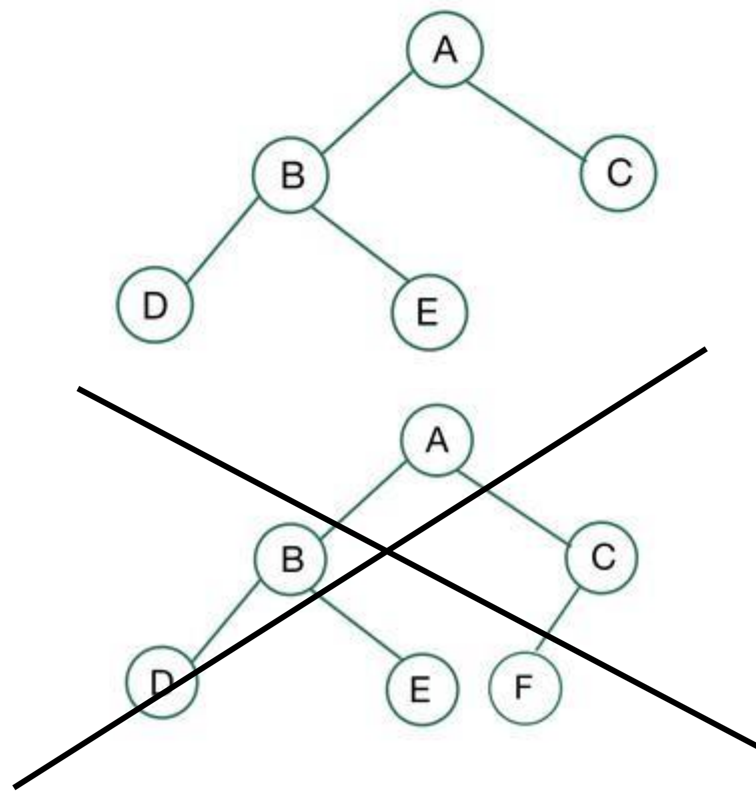




Full Binary Tree

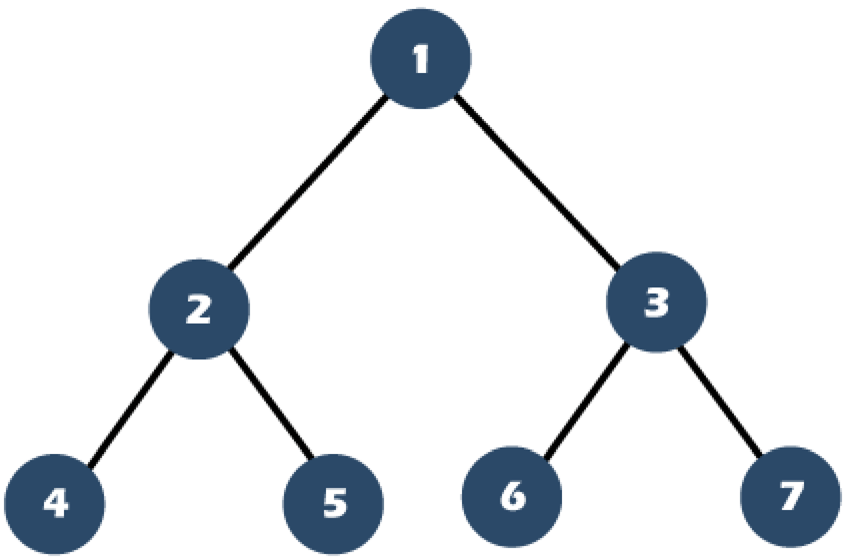
A full binary tree is a binary tree in which every node has either zero or two children.

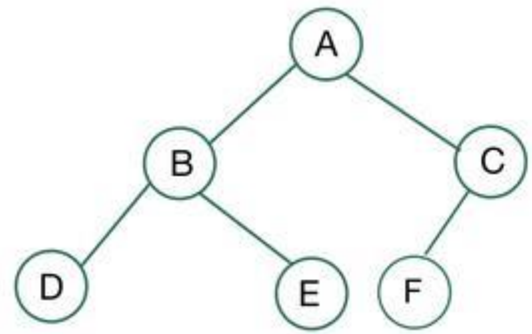


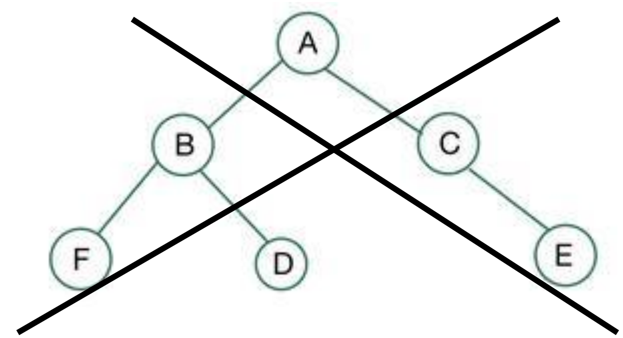


Complete Binary Tree

A complete binary tree is a binary tree in which all levels except last level have maximum number of possible nodes. Moreover, nodes in last level are appear as far left as possible.

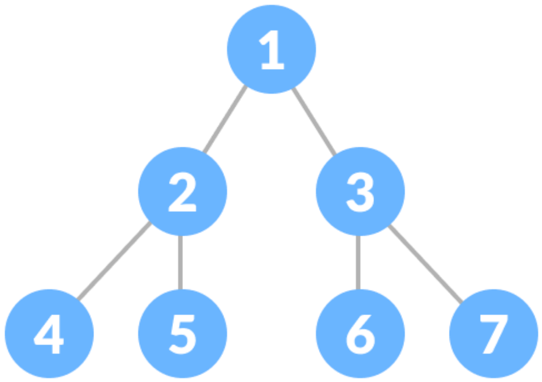


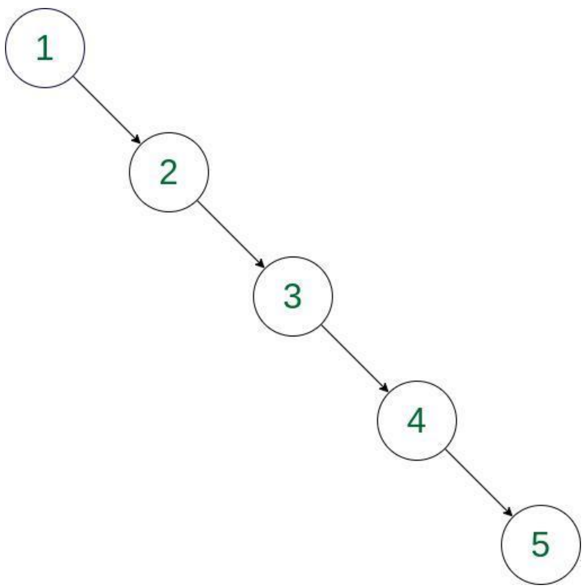




Properties of a binary tree

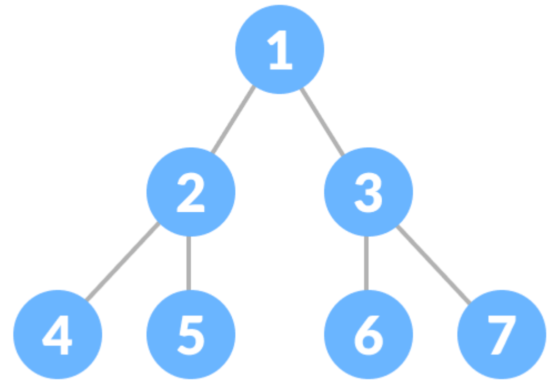
- Max no. of nodes at x^{th} level = 2^x
- Max no. of nodes in a binary tree of height h = $2^h – 1$
- Min no. of possible nodes in binary tree of height h = h
- Max no. of leaf nodes in a binary tree of height h = 2^{h-1}
- No. of edges in a binary tree with n number nodes = n – 1
- No. of leaf nodes in a binary tree with n no. of internal node (degree = 2) = n + 1

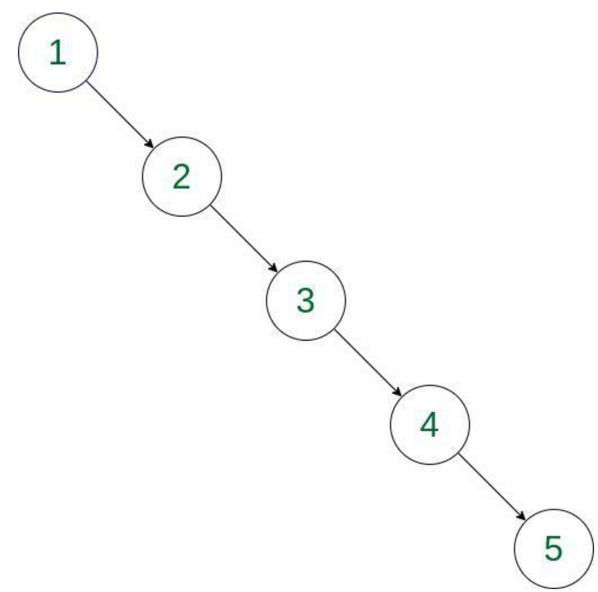




Properties of a binary tree

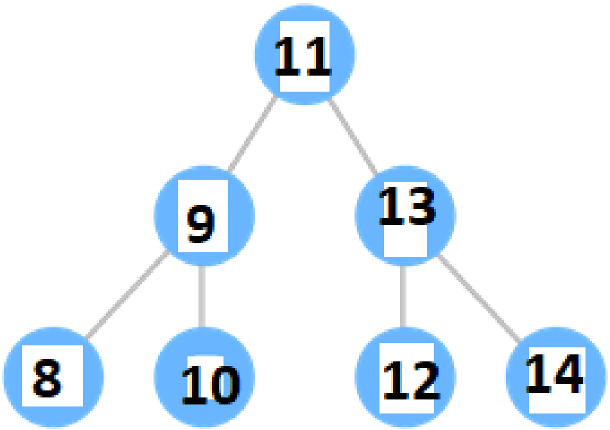
- Height of a complete binary tree with n number of nodes $= \lceil log_2(n+1) \rceil$
- Total number of binary tree with n nodes $= \frac{1}{n+1} 2nCn$
- Max no. of levels of binary tree with n nodes = n -1
- Min no. of levels of binary tree with n nodes = $\lceil log_2(n+1) - 1 \rceil$





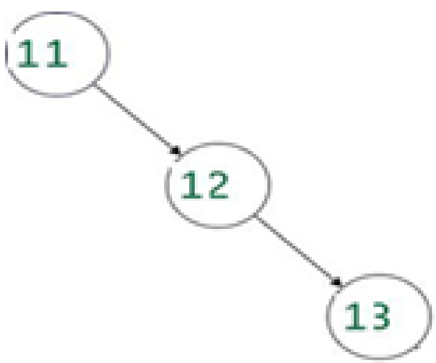
Array representation of Binary Tree

- Left child of i^{th} node is at (2*i) index.
- Right child of i^{th} node is at ((2*i)+1) index.
- Parent of i^{th} node is at $\left\lfloor \frac{i}{2} \right\rfloor$.



1	2	3	4	5	6	7
11	9	13	8	10	12	14

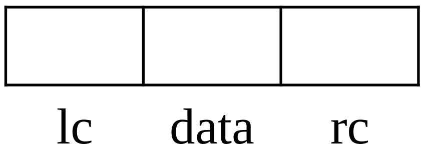
1	2	3	4	5	6	7
11	NULL	12	NULL	NULL	NULL	13



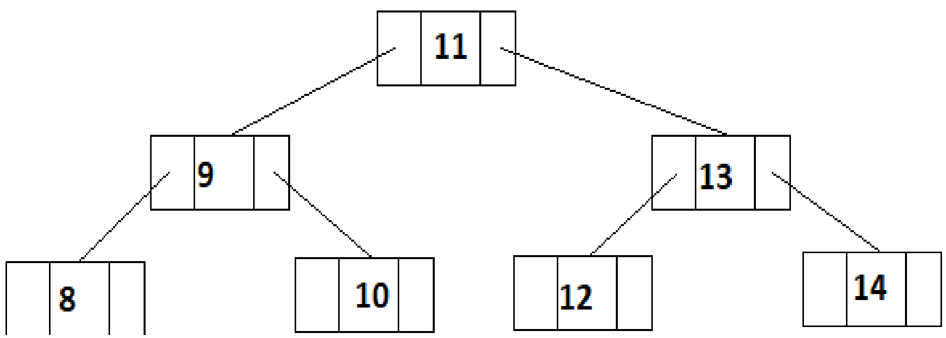
For n number of nodes,

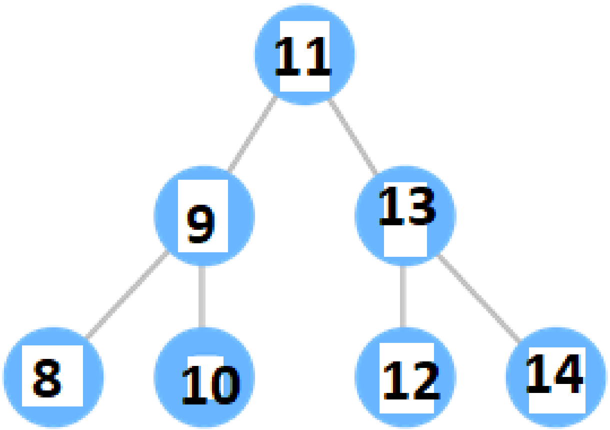
- Max size of array = $2^n - 1$
- Min size of array = $2^{\lceil \log_2(n+1) \rceil} - 1$

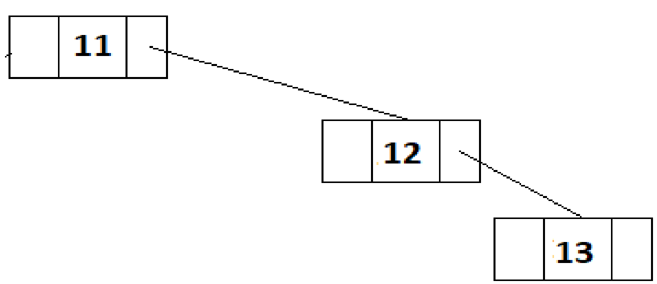
Linked List representation of Binary Tree

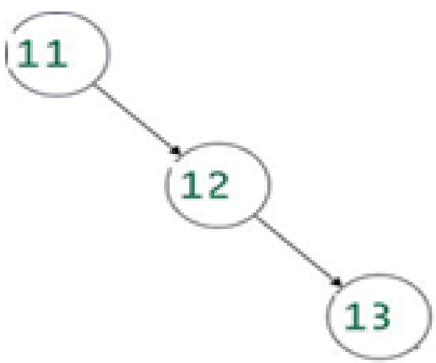


lc = link to left child
rc = link to right child









Array vs Linked List Representation

Array Representation

- ✓ Any node can be accessed by calculating index
- ✓ No storage needed for pointers
- × Empty entries may be there
- × Static representation

Linked List Representation

- × Traversal required
- × Storage needed for pointers
- ✓ No Empty entries
- ✓ Dynamic representation

Basic operations

- Insertion
- Deletion
- Traversal
- Merge

Search (Array representation)

SEARCH(rootind, key): Returns index of Key in tree

i ← rootind

if A[i] = key **then**

    Return i

else

    if 2*i < = SIZE(A) **then**

        SEARCH(2*i, key)

    if (2*i)+1 < = SIZE(A) **then**

        SEARCH((2*i)+1, key)

    else

        Return -1

Insertion (Array representation)

Insert(key, object): Insert Object as left/right child of Key

ind ← SEARCH (1, key)

if ind = -1 **then**

    print "Search unsuccessful"

    Exit

else

    if A[2*ind] = NULL **then**

        A[2*ind] ← object

    else if A[(2*ind) + 1] = NULL **then**

        A[(2*ind) + 1] ← object

    else

        print "Object cannot be inserted at desired location"

        Exit

Deletion (Array representation)

Delete(key): Delete key object from tree if it is leaf

```
ind ← SEARCH (1, key)
if ind = -1 then
        print "Search unsuccessful"
        Exit
if A[2*ind] = NULL and A[(2*ind)+1] = NULL then
        A[ind] = NULL
else
        print "Key is not at leaf node"
        Exit
```

Homework

Write algorithms for Search, Insert, Delete for linked list representation of a binary tree.

Traversal (Linked List representation)

Preorder:

- Visit root node R
- Traverse left subtree of R
- Traverse right subtree of R

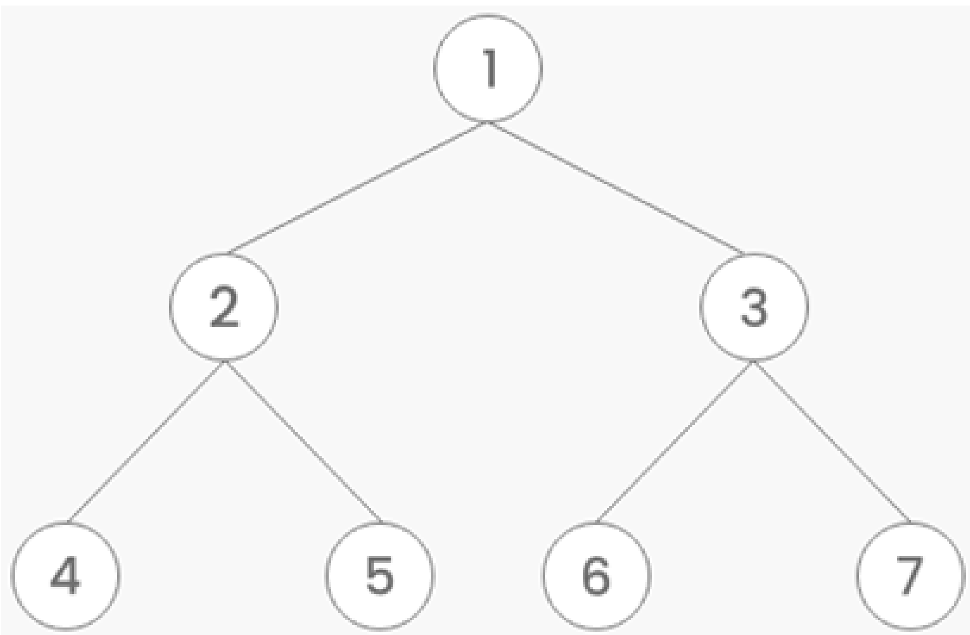
Inorder:

- Traverse left subtree of R
- Visit root node R
- Traverse right subtree of R

Postorder:

- Traverse left subtree of R
- Traverse right subtree of R
- Visit root node R

Traversal (Linked List representation)



Inorder Traversal

4	2	5	1	6	3	7

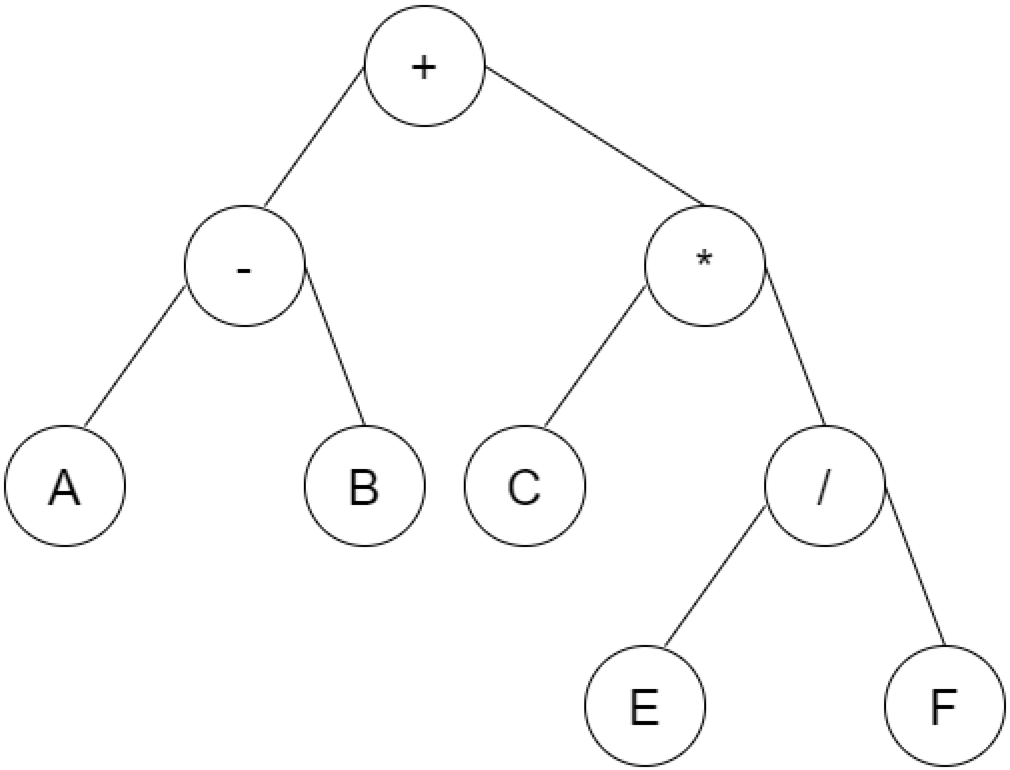
Preorder Traversal

1	2	4	5	3	6	7

Postorder Traversal

4	5	2	6	7	3	1

Traversal (Linked List representation)



Preorder:
+-AB*C/EF

Inorder:
A-B+C*E/F

Postorder:
AB-CEF/*+

Traversal (Linked List representation)

(A-B)+(C*(E/F))

Prefix:

+-AB*C/EF

Postfix:

AB-CEF/*+

Traversal (Linked List representation)

inorder(root)

ptr ← root

if ptr ≠ NULL **then**

inorder(ptr.left)

print(ptr.data)

inorder(ptr.right)

preorder(root)

ptr ← root

if ptr ≠ NULL **then**

print(ptr.data)

preorder(ptr.left)

preorder(ptr.right)

postorder(root)

ptr ← root

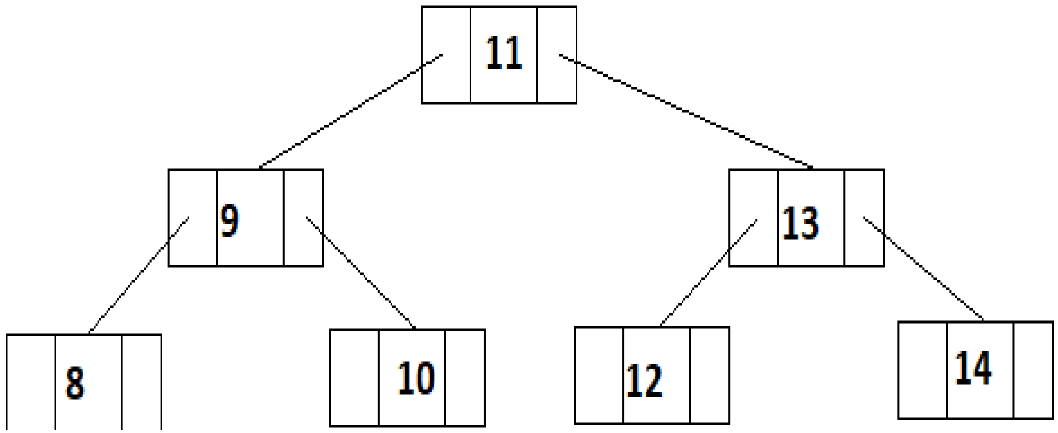
if ptr ≠ NULL **then**

postorder(ptr.left)

postorder(ptr.right)

print(ptr.data)

Traversal (Linked List representation)



preorder(root)

ptr ← root

if ptr ≠ NULL **then**

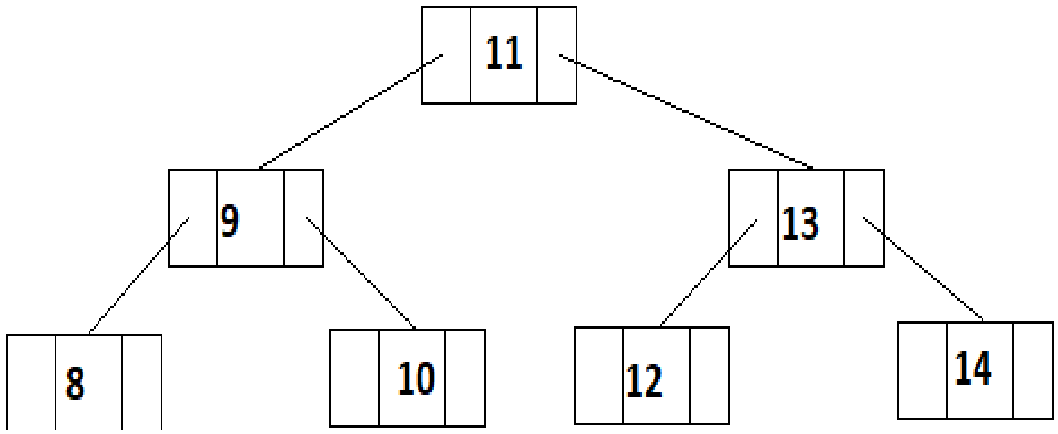
print(ptr.data)

preorder(ptr.left)

preorder(ptr.right)

11
9
8
10
13
12
14

Traversal (Linked List representation)



inorder(root)

ptr ← root

if ptr ≠ NULL **then**

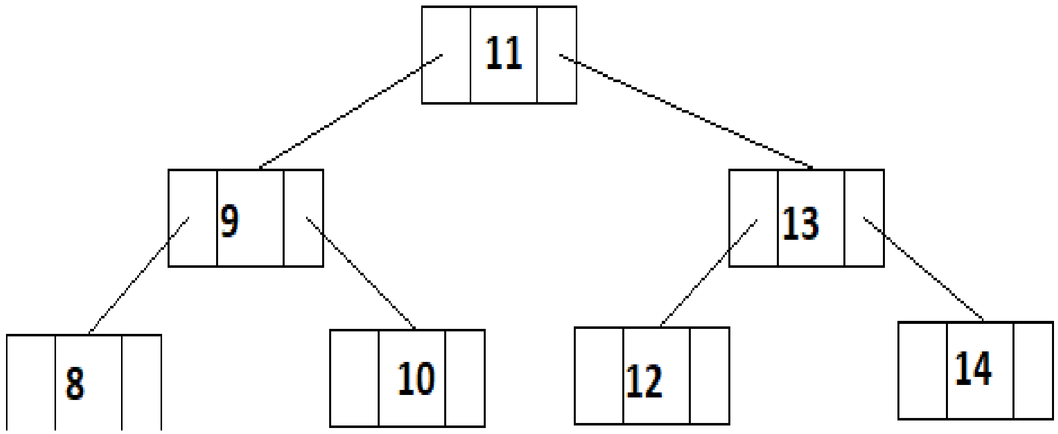
inorder(ptr.left)

print(ptr.data)

inorder(ptr.right)

8
9
10
11
12
13
14

Traversal (Linked List representation)



postorder(root)

ptr ← root

if ptr ≠ NULL **then**

  postorder(ptr.left)

  postorder(ptr.right)

  print(ptr.data)

8
10
9
12
14
13
11

Traversal (Linked List representation) Non-recursive

```
preorder(root)
PUSH(root)
while Stack.empty() = FALSE do
   ptr ← POP( )
   if ptr ≠ NULL then
      print(ptr.data)
      PUSH(ptr.right)
      PUSH(ptr.left)
```

```
inorder(root)
ptr ← root
while Stack.empty() = FALSE
          or  ptr ≠ NULL do
   if ptr ≠ NULL then
      PUSH(ptr)
      ptr ← ptr.left
   else
      ptr ← POP ( )
      print(ptr.data)
      ptr ← ptr.right
```

postorder(root)

homework

Formation of binary tree from traversals

We can form the binary tree from any two traversals.

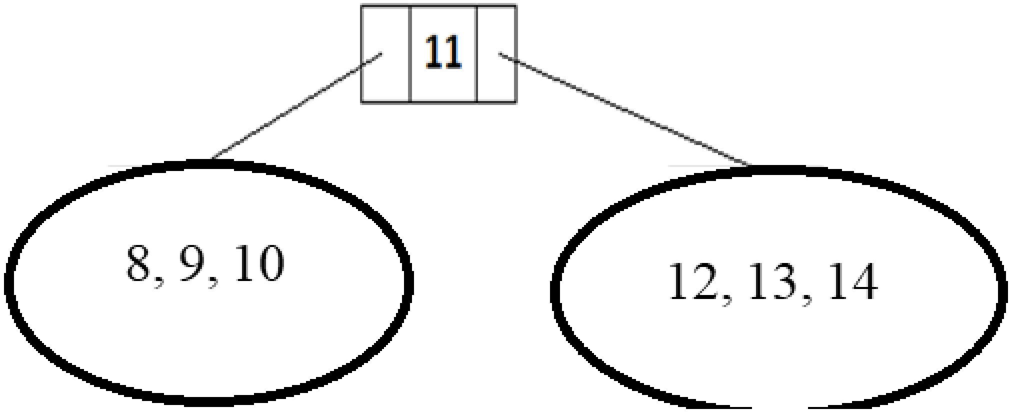
- If preorder traversal is given, the first node is *root node*.
- If postorder traversal is given, the last node is *root node*.
- Once root node is found, left subtree and right subtree are to be identified.
- Repeat same method for left subtree and right subtree.

Note: inorder is required to generate unique binary tree.

Preorder-Inorder

Preorder: 11, 9, 8, 10, 13, 12, 14
Inorder: 8, 9, 10, 11, 12, 13, 14

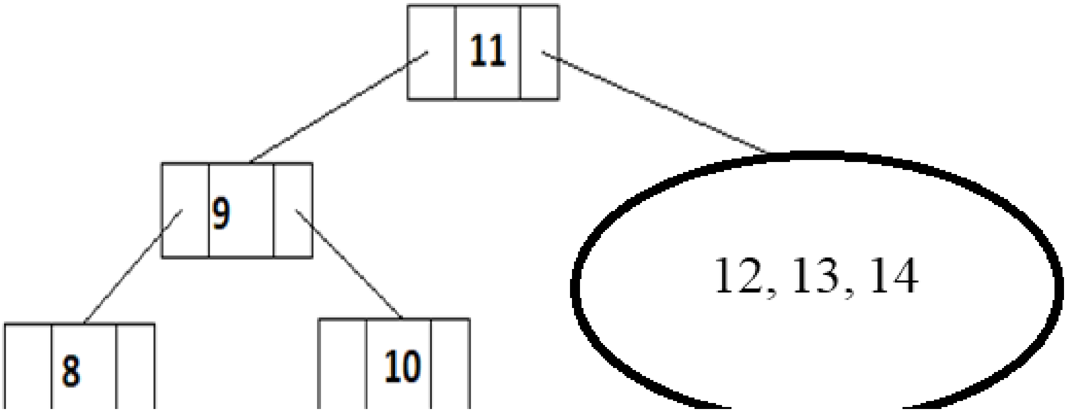
Root = 11 (From Preorder)
Left subtree contains 8, 9, 10 (From inorder)
Right subtree contains 12, 13, 14 (From inorder)



Preorder-Inorder

Preorder: 11, 9, 8, 10, 13, 12, 14
Inorder: 8, 9, 10, 11, 12, 13, 14

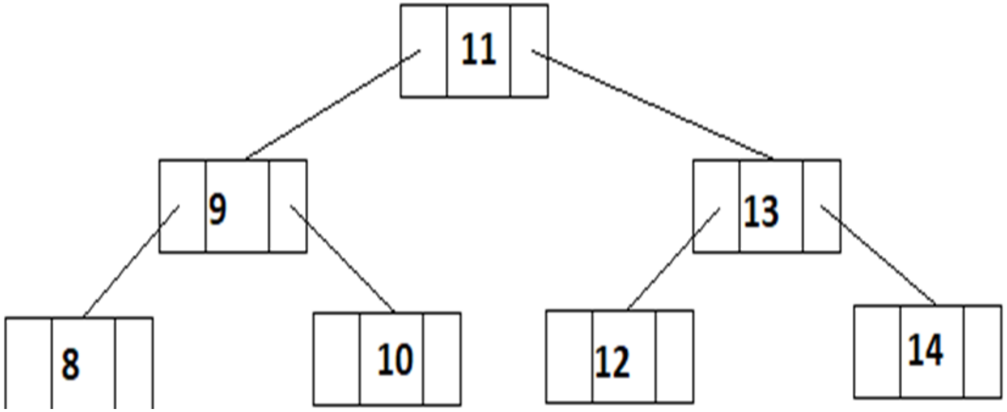
Subtree contains 8,9,10
Root node: 9 (From preorder)
Left subtree: 8 (From inorder)
Right subtree: 10 (From inorder)



Preorder-Inorder

Preorder: 11, 9, 8, 10, 13, 12, 14
Inorder: 8, 9, 10, 11, 12, 13, 14

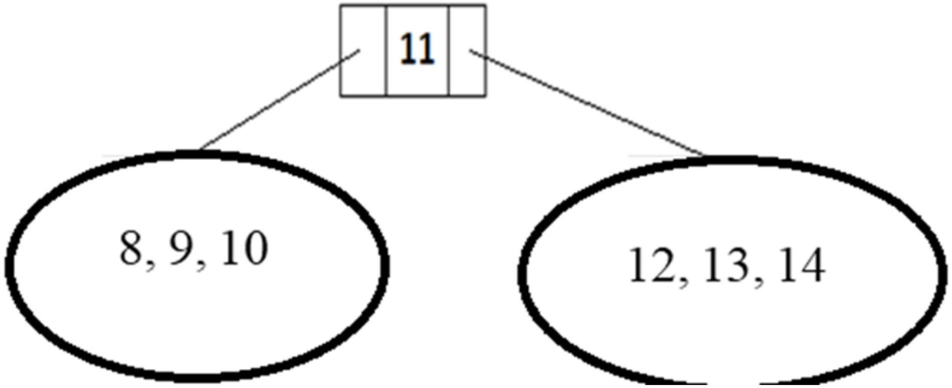
Subtree contains 12,13,14
Root node: 13 (From preorder)
Left subtree: 12 (From inorder)
Right subtree: 14 (From inorder)



Postorder-Inorder

Postorder: 8, 10, 9, 12, 14, 13, 11
Inorder: 8, 9, 10, 11, 12, 13, 14

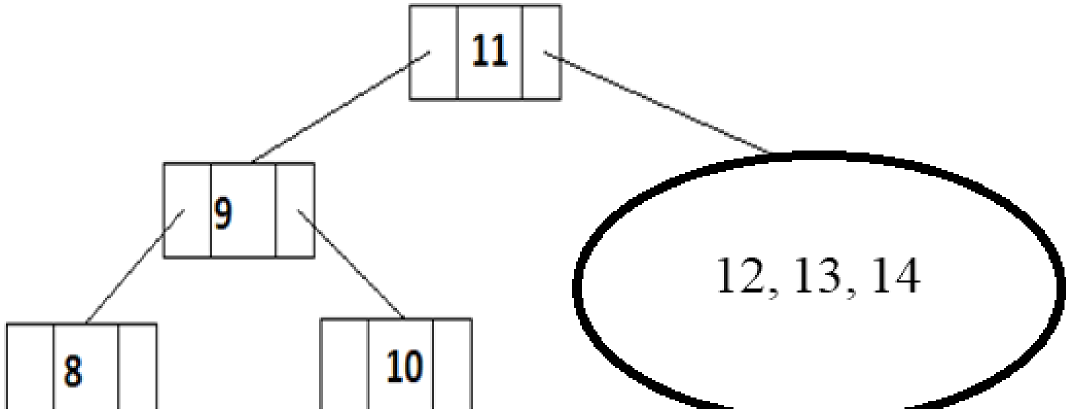
Root = 11 (From Postorder)
Left subtree contains 8, 9, 10 (From inorder)
Right subtree contains 12, 13, 14 (From inorder)



Postorder-Inorder

Postorder: 8, 10, 9, 12, 14, 13, 11
Inorder: 8, 9, 10, 11, 12, 13, 14

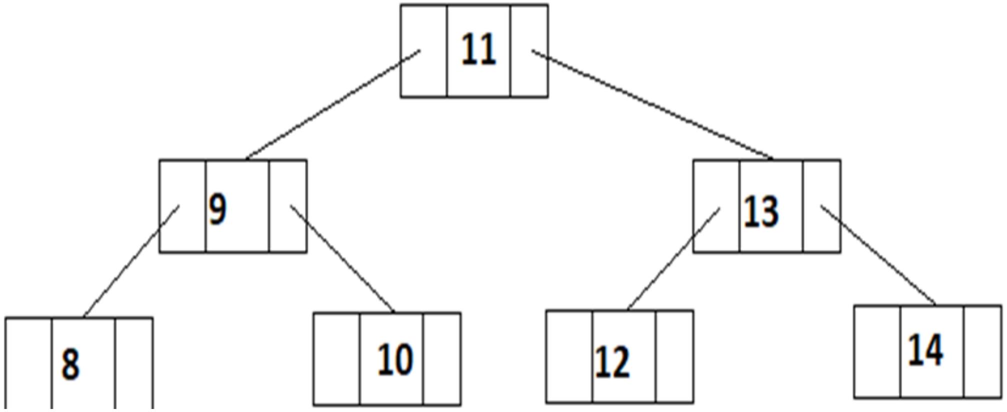
Subtree contains 8,9,10
Root node: 9 (From postorder)
Left subtree: 8 (From inorder)
Right subtree: 10 (From inorder)



Postorder-Inorder

Postorder: 8, 10, 9, 12, 14, 13, 11
Inorder: 8, 9, 10, 11, 12, 13, 14

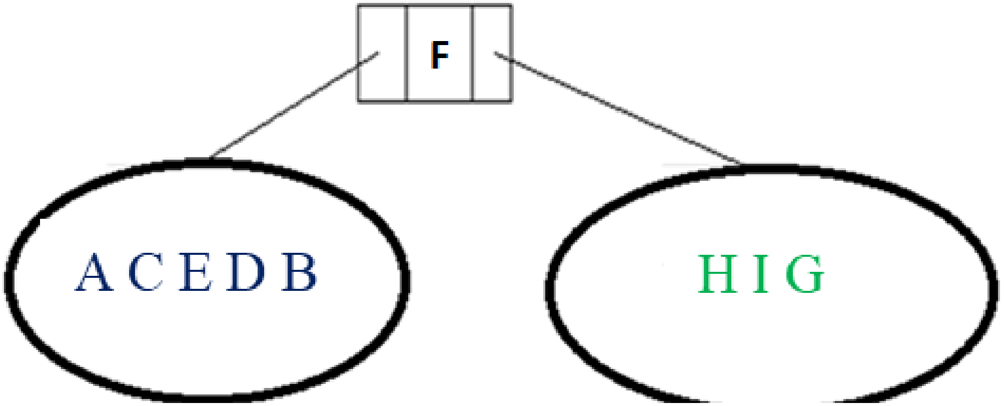
Subtree contains 12,13,14
Root node: 13 (From postorder)
Left subtree: 12 (From inorder)
Right subtree: 14 (From inorder)



Preorder-Postorder

Preorder: F B A D C E G I H
Postorder: A C E D B H I G F

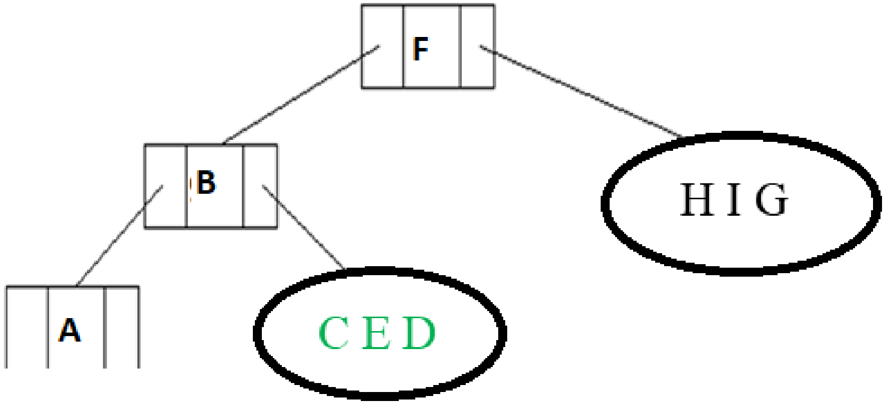
Root node: F (From preorder)
Left subtree: A, C, E, D, B (From preorder and postorder)
Right subtree: H I G (From preorder and postorder)



Preorder-Postorder

Preorder: F B A D C E G I H
Postorder: A C E D B H I G F

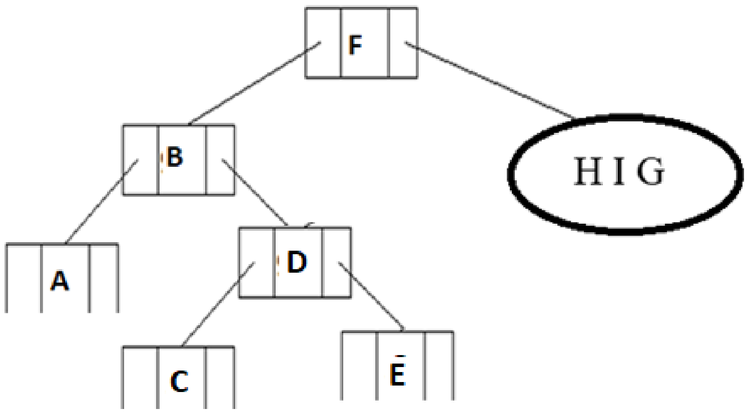
Root node: B (From preorder)
Left subtree: A (From preorder and postorder)
Right subtree: C E D (From preorder and postorder)



Preorder-Postorder

Preorder: F B A D C E G I H
Postorder: A C E D B H I G F

Root node: D (From preorder)
Left subtree: C (From preorder and postorder)
Right subtree: E (From preorder and postorder)

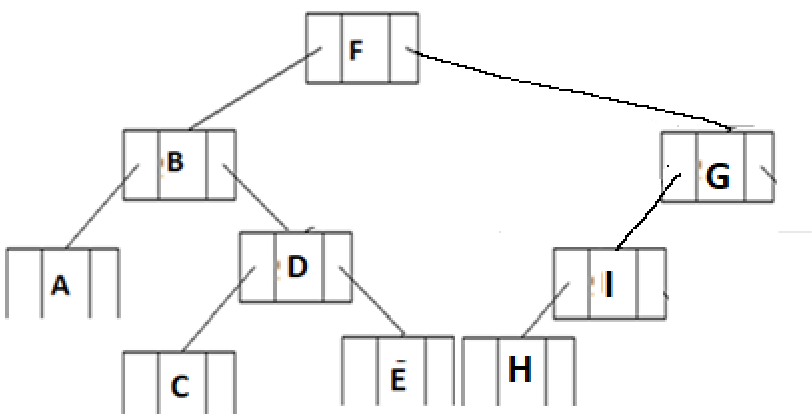


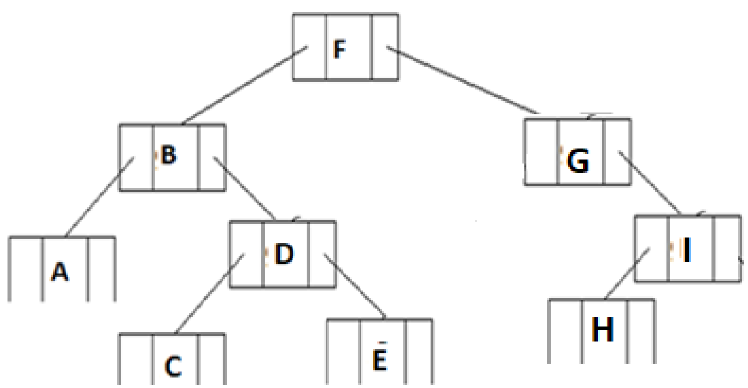
Preorder-Postorder

Preorder: F B A D C E G I H
Postorder: A C E D B H I G F

Not possible to construct unique binary tree when preorder and postorder given.

Root node: G (From preorder)
Left subtree: I H (From preorder and postorder)
or
Right subtree: I H (From preorder and postorder)





References

1. Algorithms Design: Foundations, Analysis and Internet Examples Michael T. Goodrich, Roberto Tamassia, 2006, Wiley (Students Edition)
2. Data Structures, Algorithms and Applications in C++, Sartaj Sahni, Second Ed, 2005, Universities Press
3. Introduction to Algorithms, TH Cormen, CE Leiserson, RL Rivest, C Stein, Third Ed, 2009, PHI

Any Question!!


Thank you!!
BITS Pilani
Hyderabad Campus