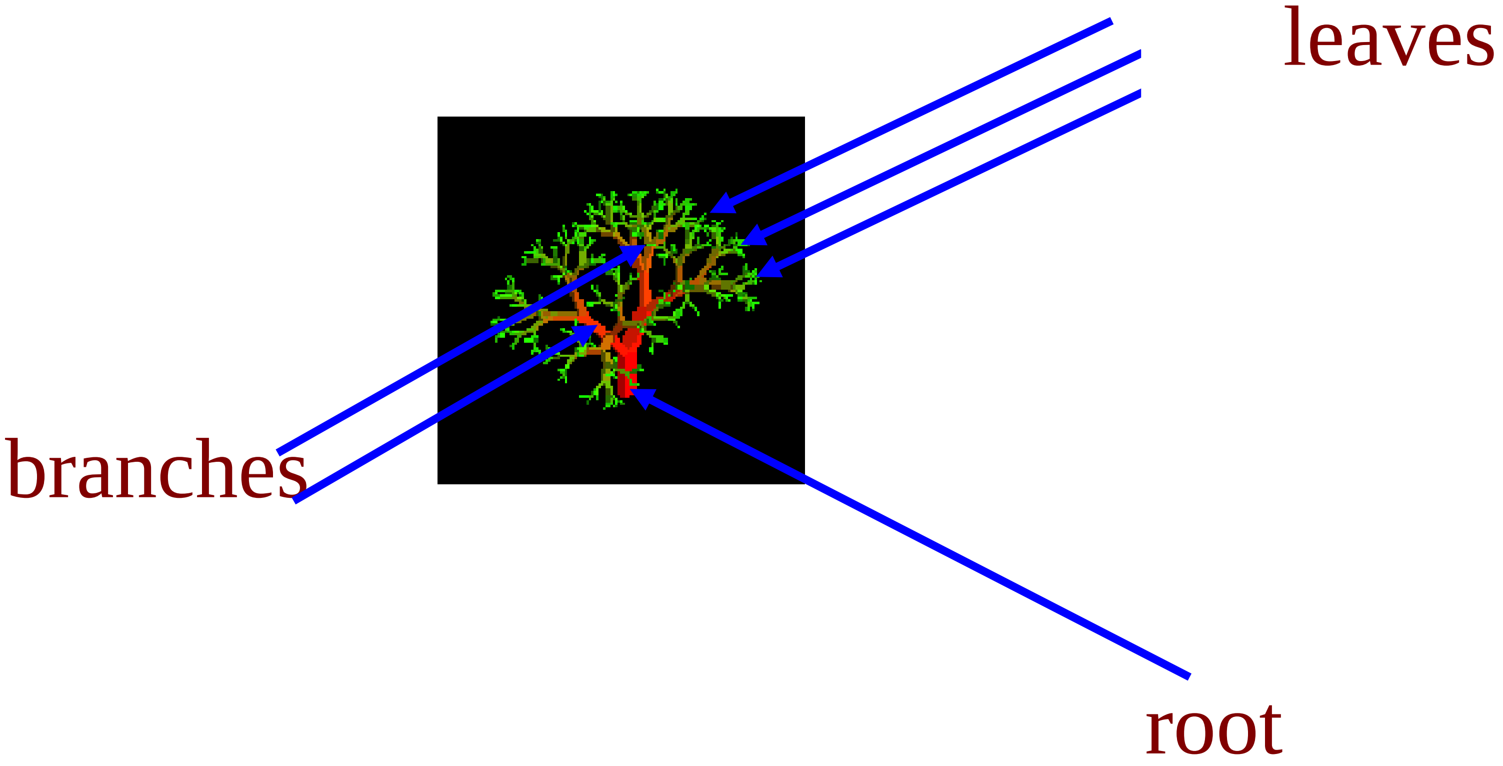
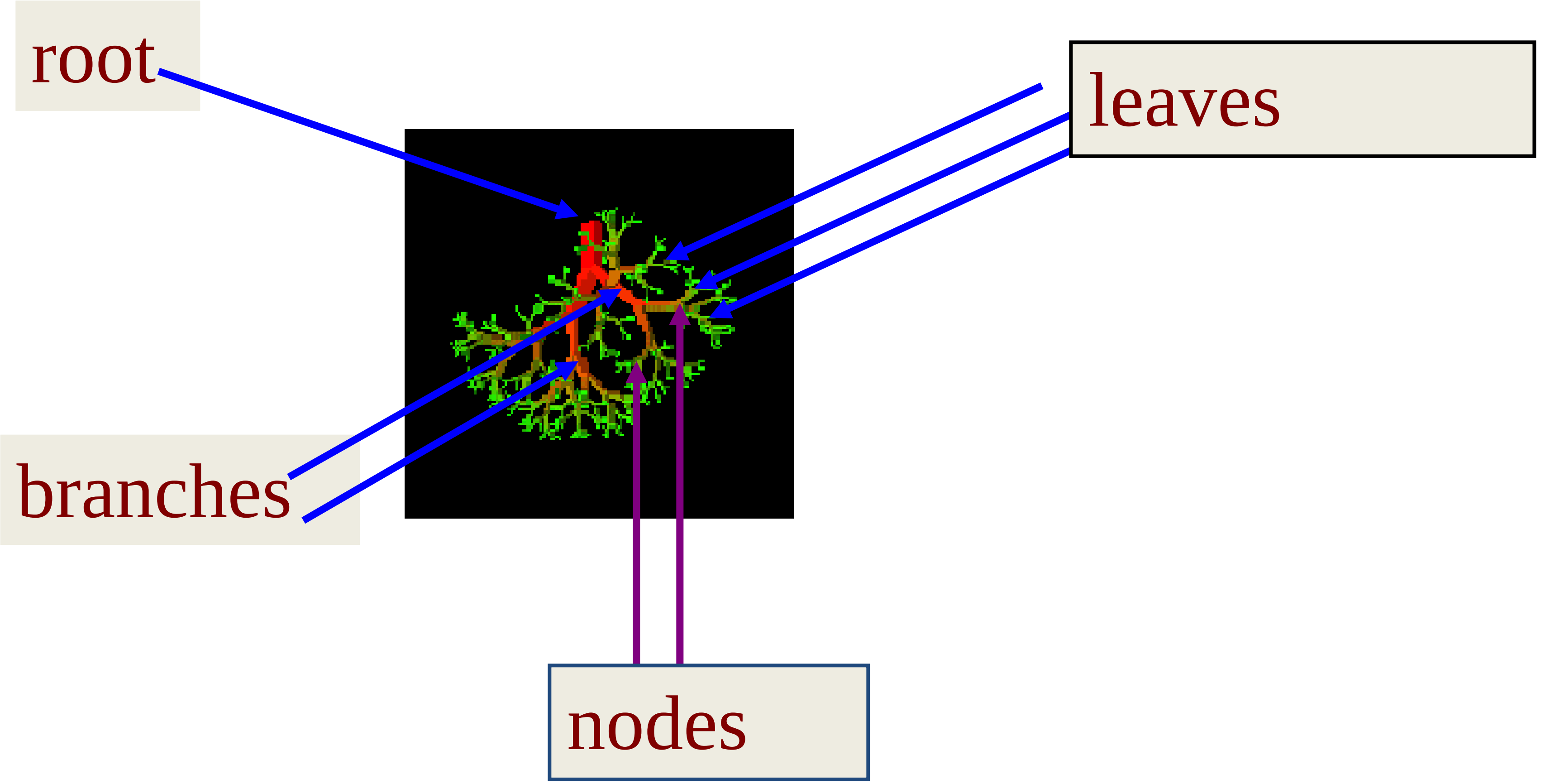
TREES

Nature Lover’s View Of A Tree



Computer Scientist’s View



Linear Lists And Trees

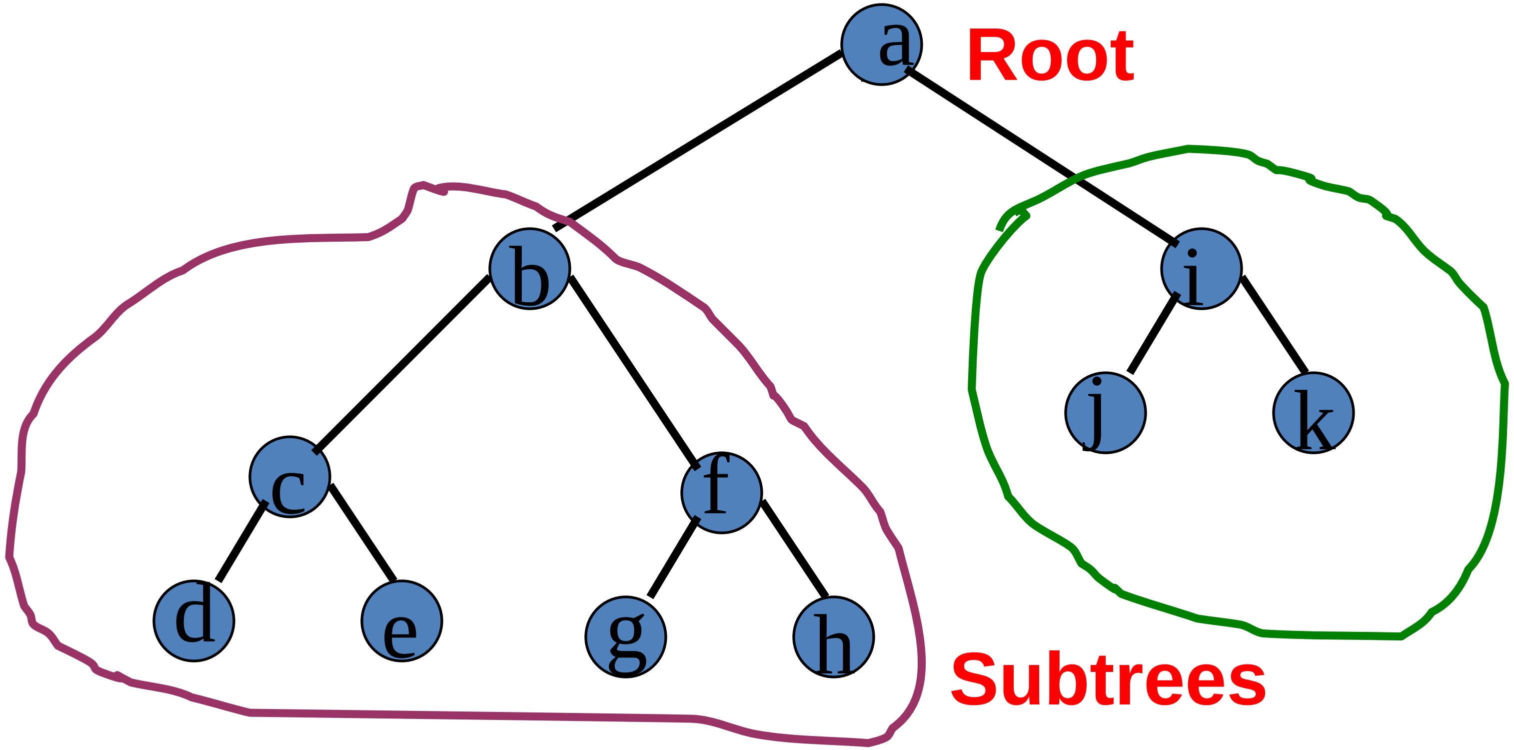
- Linear lists are useful for serially ordered data
  - $(e_0, e_1, e_2, \ldots, e_{n-1})$
  - Days of week
  - Months in a year
  - Students in this class
- Trees are useful for hierarchically ordered data
  - Employees of a corporation
    - President, vice presidents, managers, and so on

Hierarchical Data And Trees

- The element at the top of the hierarchy is the root
- Elements next in the hierarchy are the children of the root
- Elements next in the hierarchy are the grandchildren of the root, and so on.
- Elements that have no children are leaves

Definition

- A tree T is connected acyclic graph



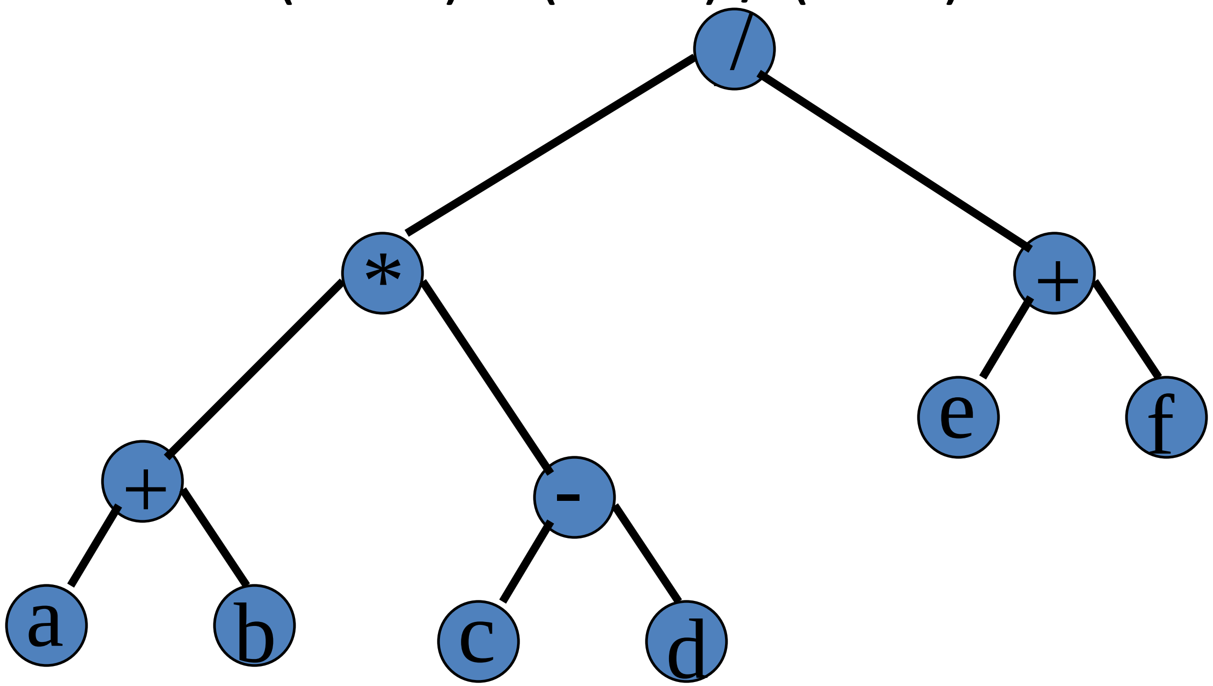
Tree & Binary Tree

- No node in a binary tree may have a degree more than 2, whereas there is no limit on the degree of a node in a tree

- are different when viewed as ordered trees
- are the same when viewed as trees

Binary Tree Form and its Merits

- (a + b) * (c – d) / (e + f)

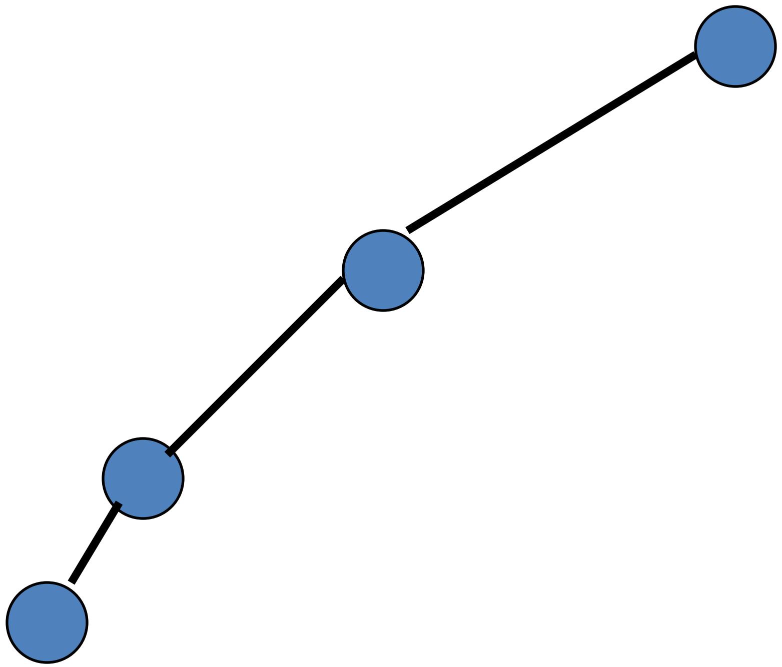


The terms that we introduced for trees, such as degree, level, height, leaf, child etc. all apply to binary tree in the same way

- Left and right operands are easy to visualize
- Code optimization algorithms work with the binary tree form of an expression
- Simple recursive evaluation of expression

Minimum Number Of Nodes

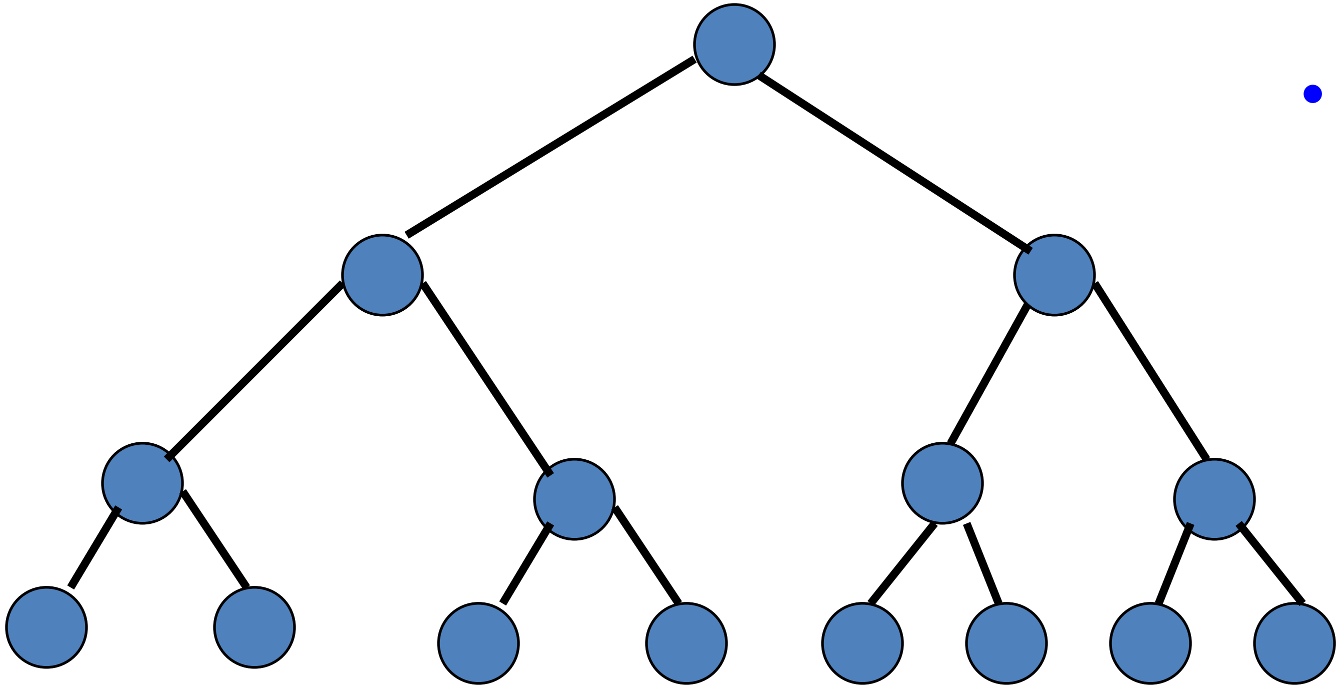
- Minimum number of nodes in a binary tree whose height is h.
- At least one node at each of first h levels.



minimum number of nodes is h+1=O(h)

Maximum Number Of Nodes

- All possible nodes at first h levels are present



- The maximum number of nodes on level i of a binary tree is 2^i

Maximum number of internal nodes of a binary tree of height h

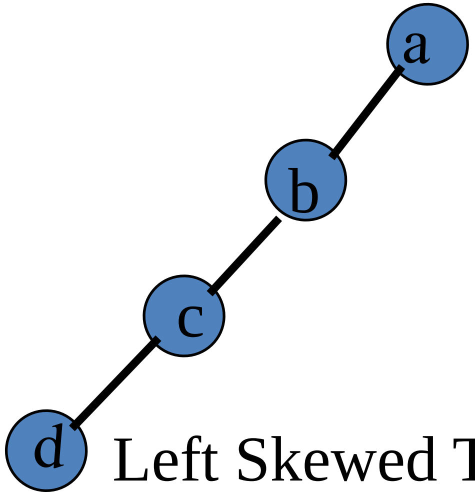
$$= 1 + 2 + 4 + 8 + \ldots + 2^{h-1} = 2^h - 1$$

Number Of Nodes & Height

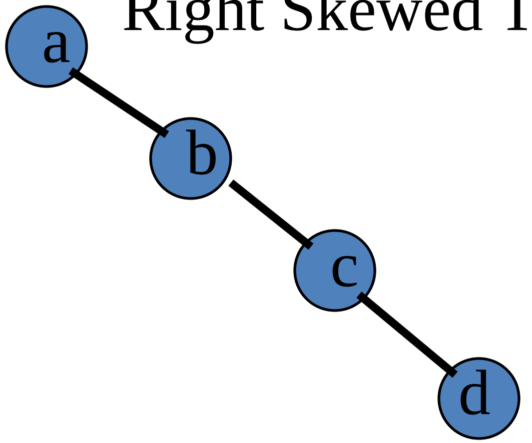
- Height of a complete binary tree with n leaves is   lg n.

Special kinds of Binary Trees

- Extended Binary Trees (2 - Trees)
- Full Binary Tree
- Complete Binary Tree
- Skewed Tree

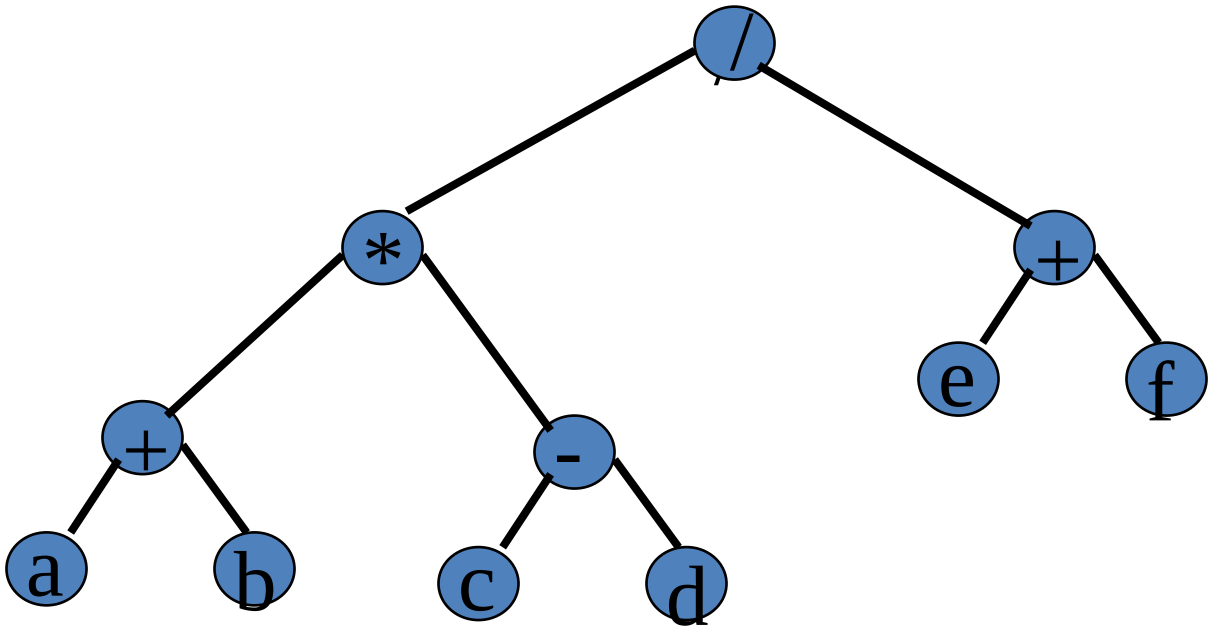


Left Skewed Tree

Right Skewed Tree


Extended Binary Trees (2 - Trees)

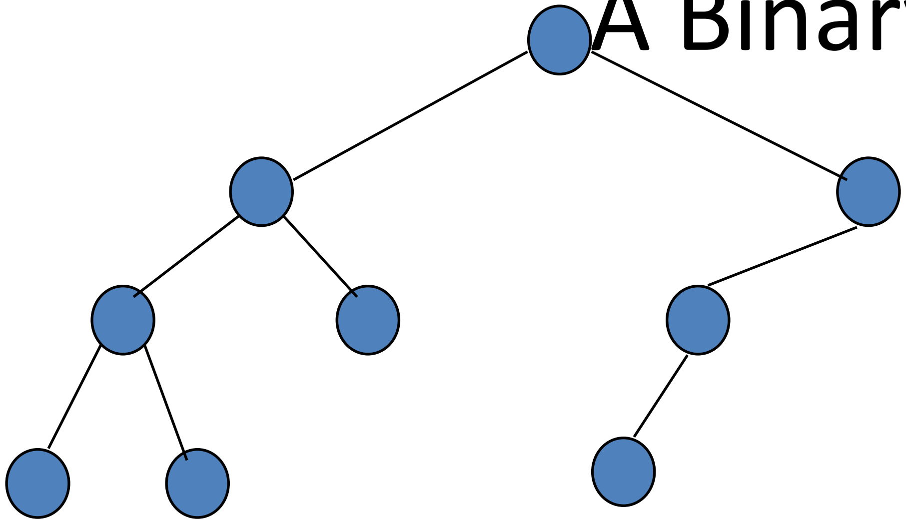
- A binary tree T is said to be a 2 – tree, if each node has either 0 or 2 children
- Tree corresponding to any Algebraic Expression which uses only binary operations



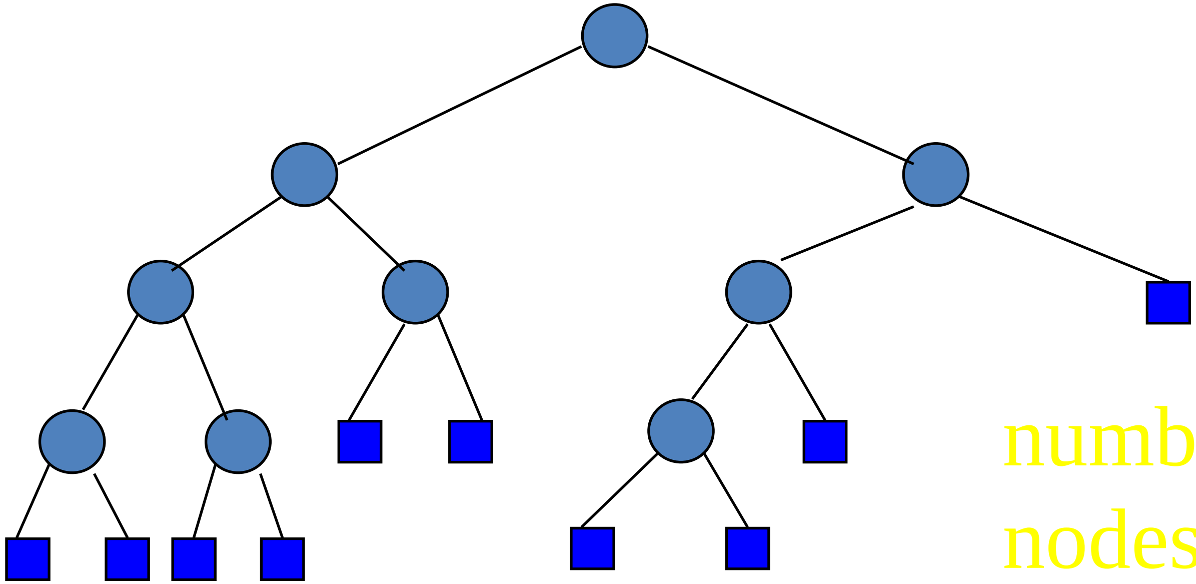
Extended Binary Trees

- We can always get an extended binary tree from a binary tree
- Start with any binary tree and add an external node wherever there is an empty subtree
- Result is an **extended binary tree**

A Binary Tree



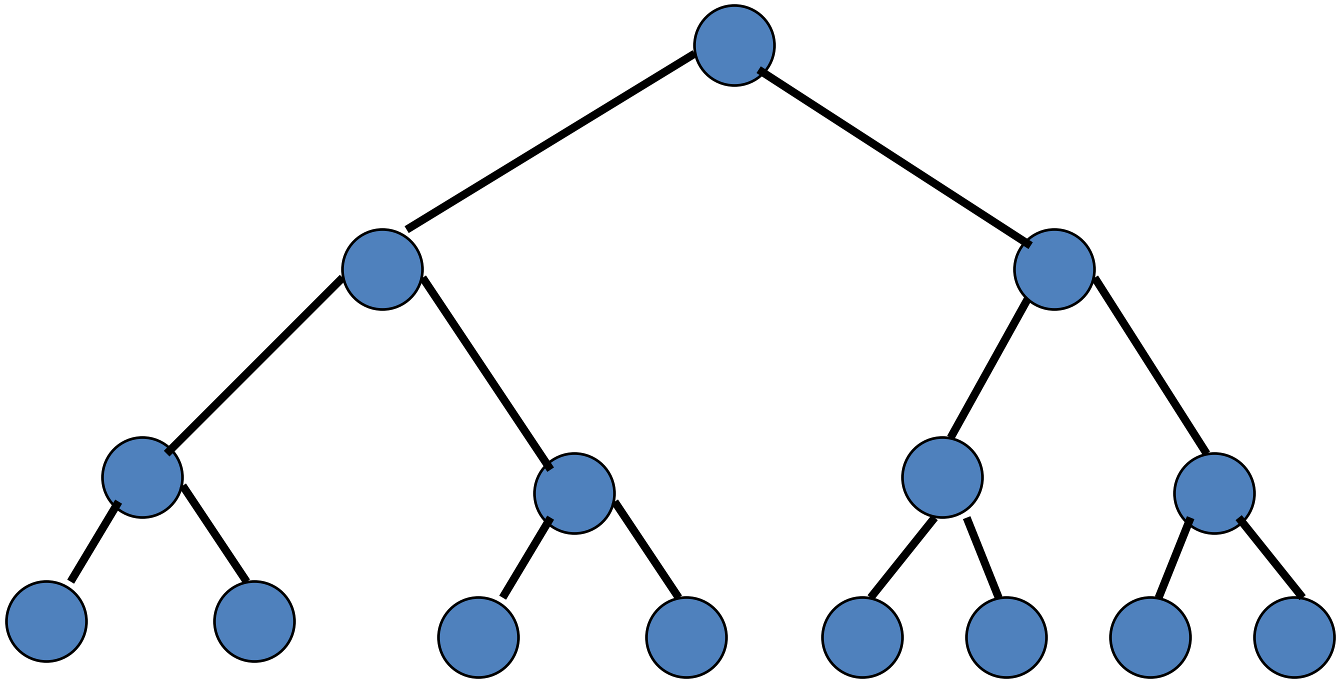
An Extended Binary Tree



number of external nodes is n+1

Full Binary Tree

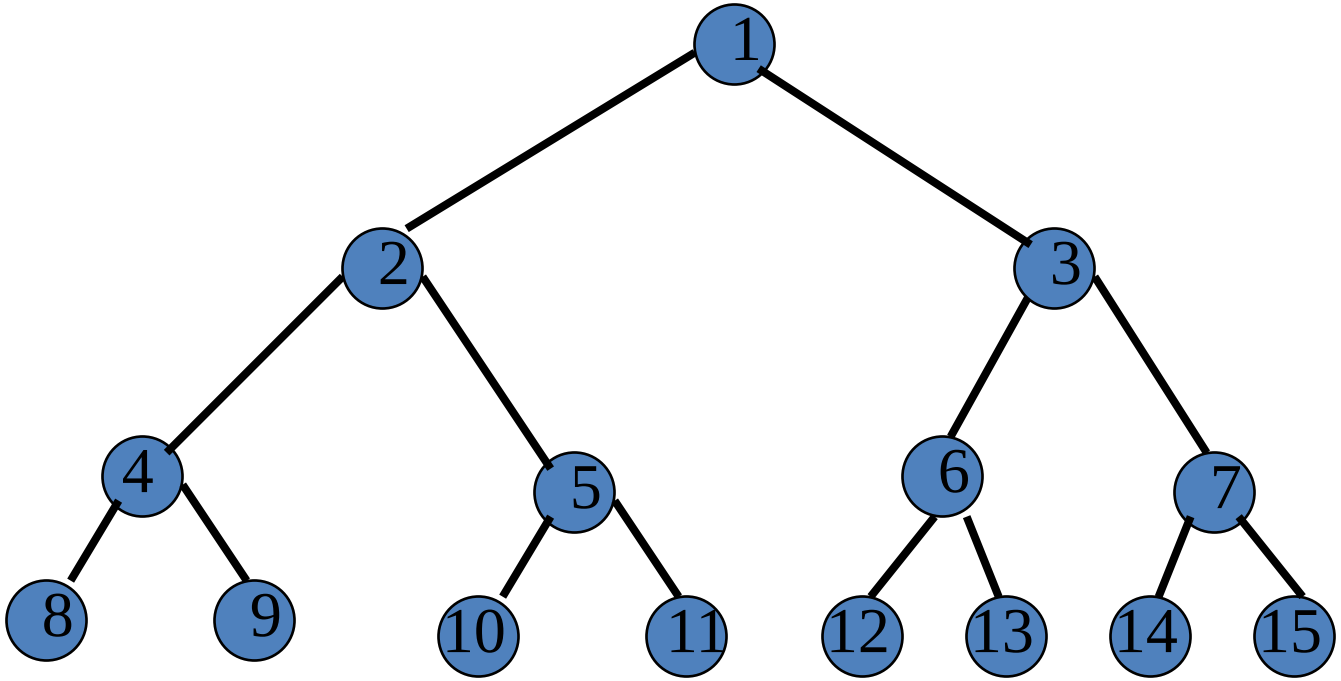
- A full binary tree of a given height h has $2^h - 1$ internal nodes



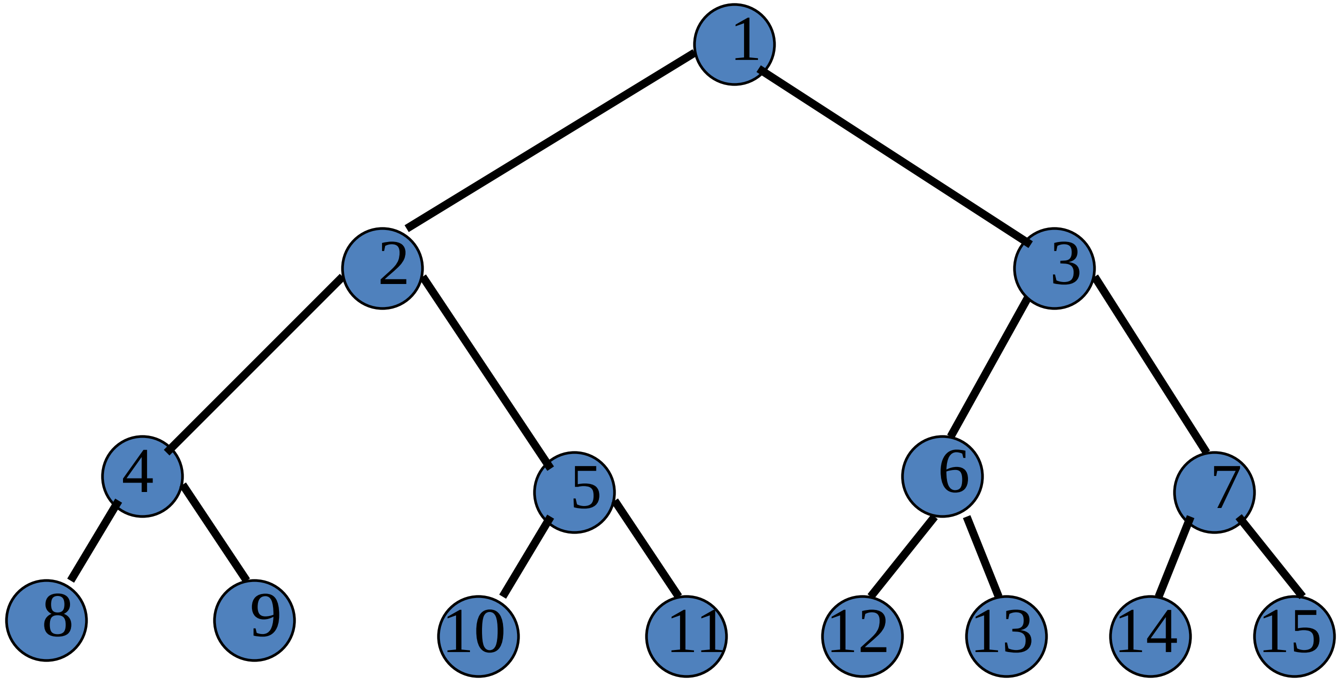
Height 3 full binary tree

Numbering Nodes In A Full Binary Tree

- Number the nodes 1 through $2^{h+1} - 1$
- Number by levels from top to bottom
- Within a level number from left to right

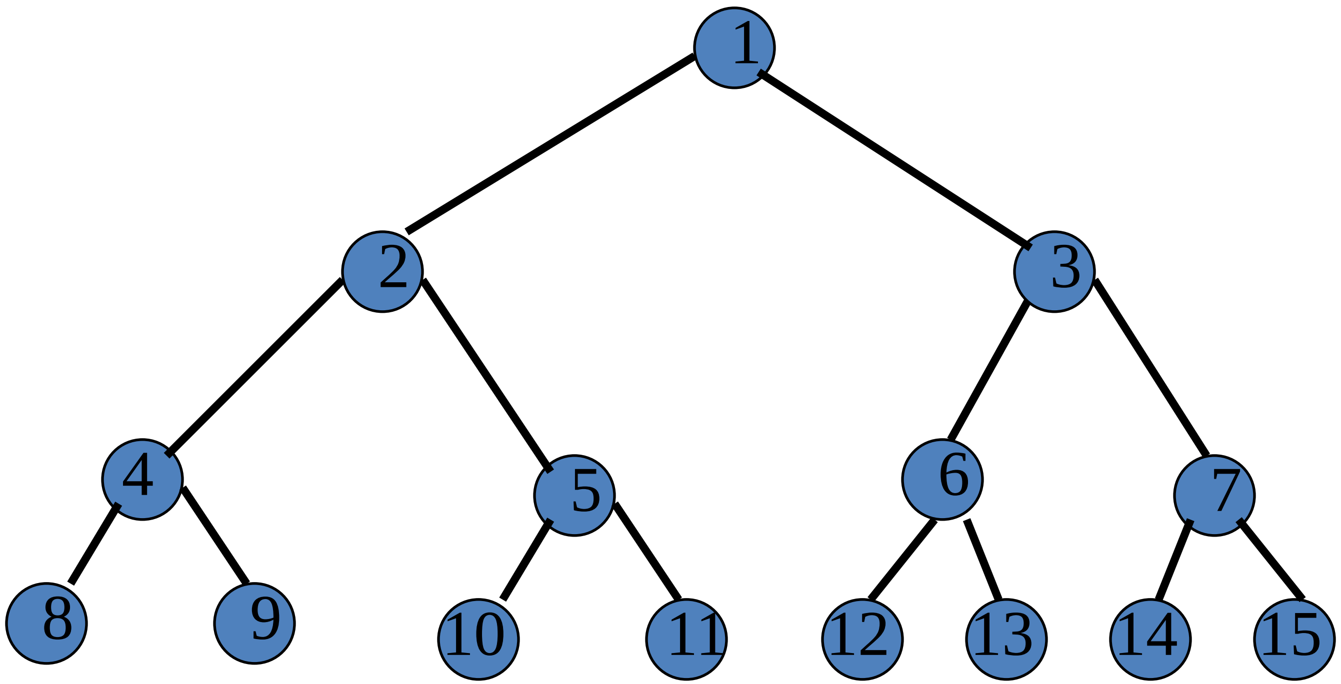


Node Number Properties



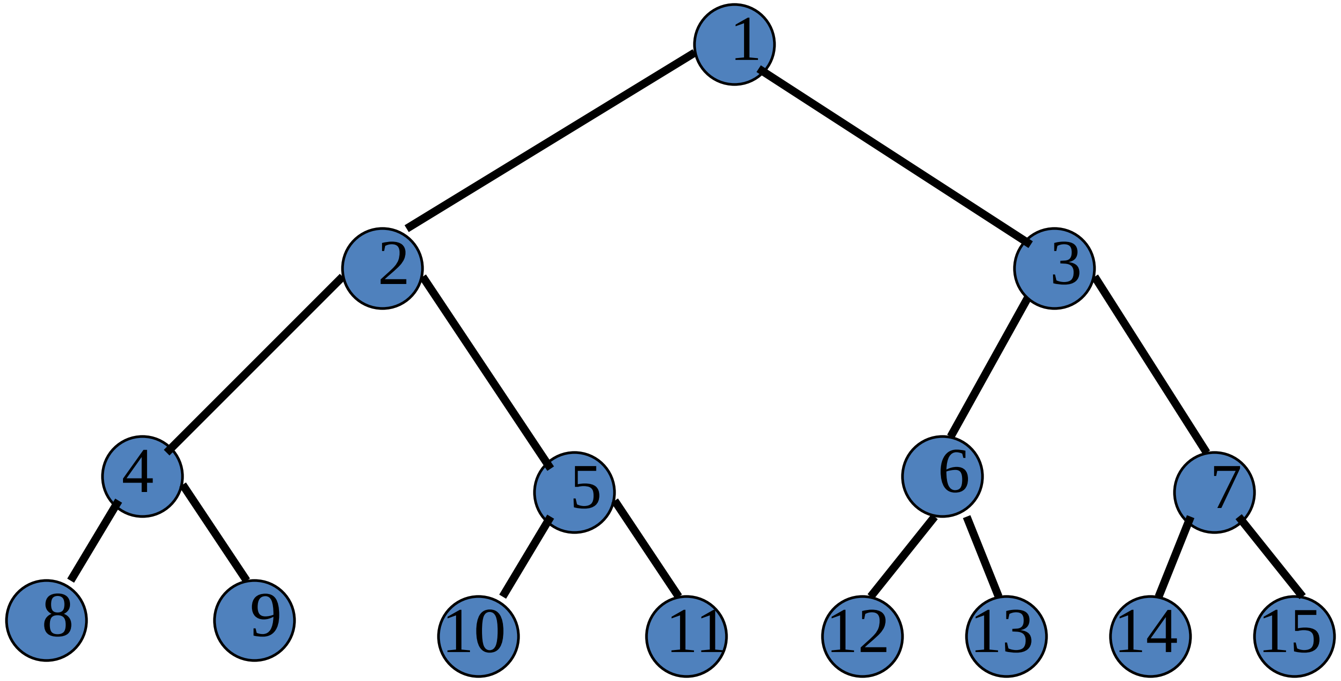
- Parent of node i is node i / 2, unless i = 1
- Node 1 is the root and has no parent

Node Number Properties



- Left child of node i is node 2i, unless 2i > n, where n is the number of nodes
- If 2i > n, node i has no left child

Node Number Properties

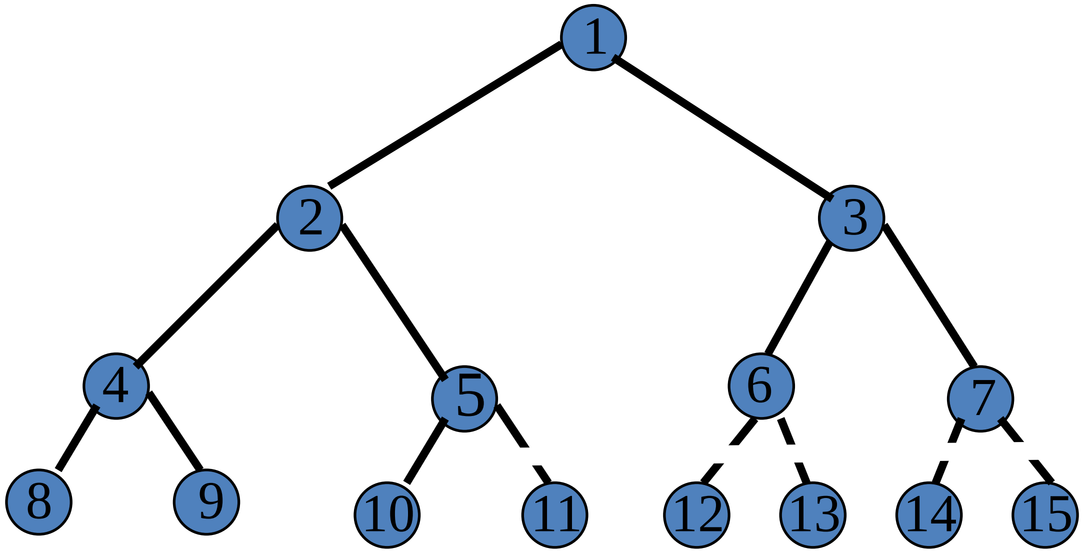


- Right child of node i is node 2i+1, unless 2i+1 > n, where n is the number of nodes
- If 2i+1 > n, node i has no right child

Complete Binary Tree with n Nodes

- Start with a full binary tree that has at least n nodes
- Number the nodes as described earlier
- The binary tree defined by the nodes numbered 1 through n is the unique n node complete binary tree
- In other words: Complete Binary Tree
  - If all its levels, except possibly the last, have the max no. of possible nodes, and
  - If all the nodes at the last level appear as far left as possible

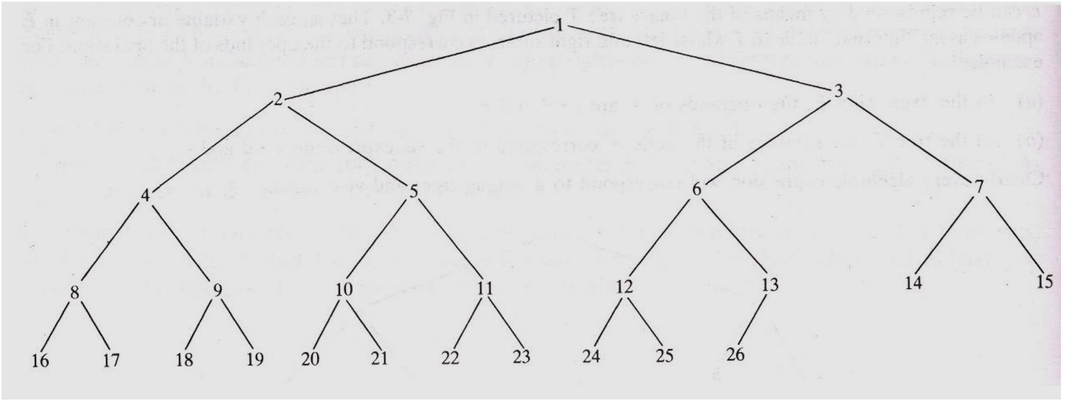
Example



- Complete binary tree with 10 nodes

The depth of a Complete Binary Tree with N nodes is given by $\lfloor \log_2 N \rfloor$

If N=1 000 000, then its depth is 21

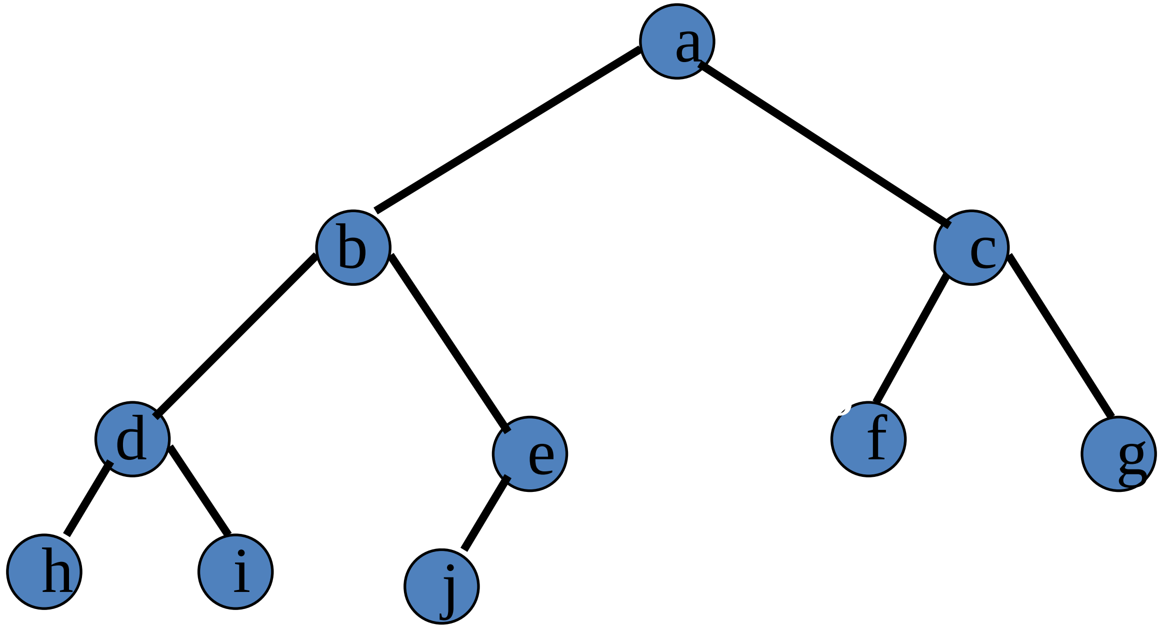


Binary Tree Representation

- Array/Sequential Representation
- Linked Representation

Array Representation

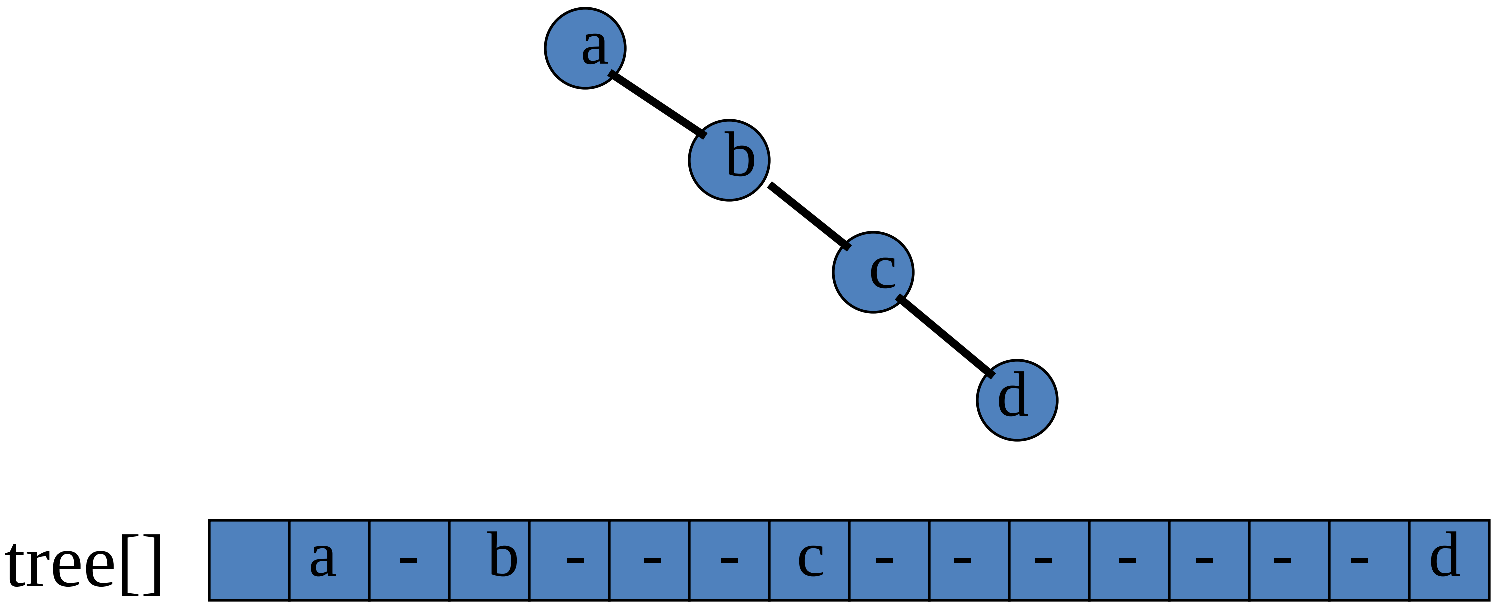
- Number the nodes using the numbering scheme for a full binary tree. The node that is numbered i is stored in tree[i].



tree[]

	a	b	c	d	e	f	g	h	i	j

Right-Skewed Binary Tree



- An n node binary tree needs an array whose length is between n+1 and 2^n.

Binary Trees- Linked Representation

- Each node has 3 fields:

  leftChild: contains the location of the left child

  data/info: contains the data at this node

  rightChild: contains the location of the right child

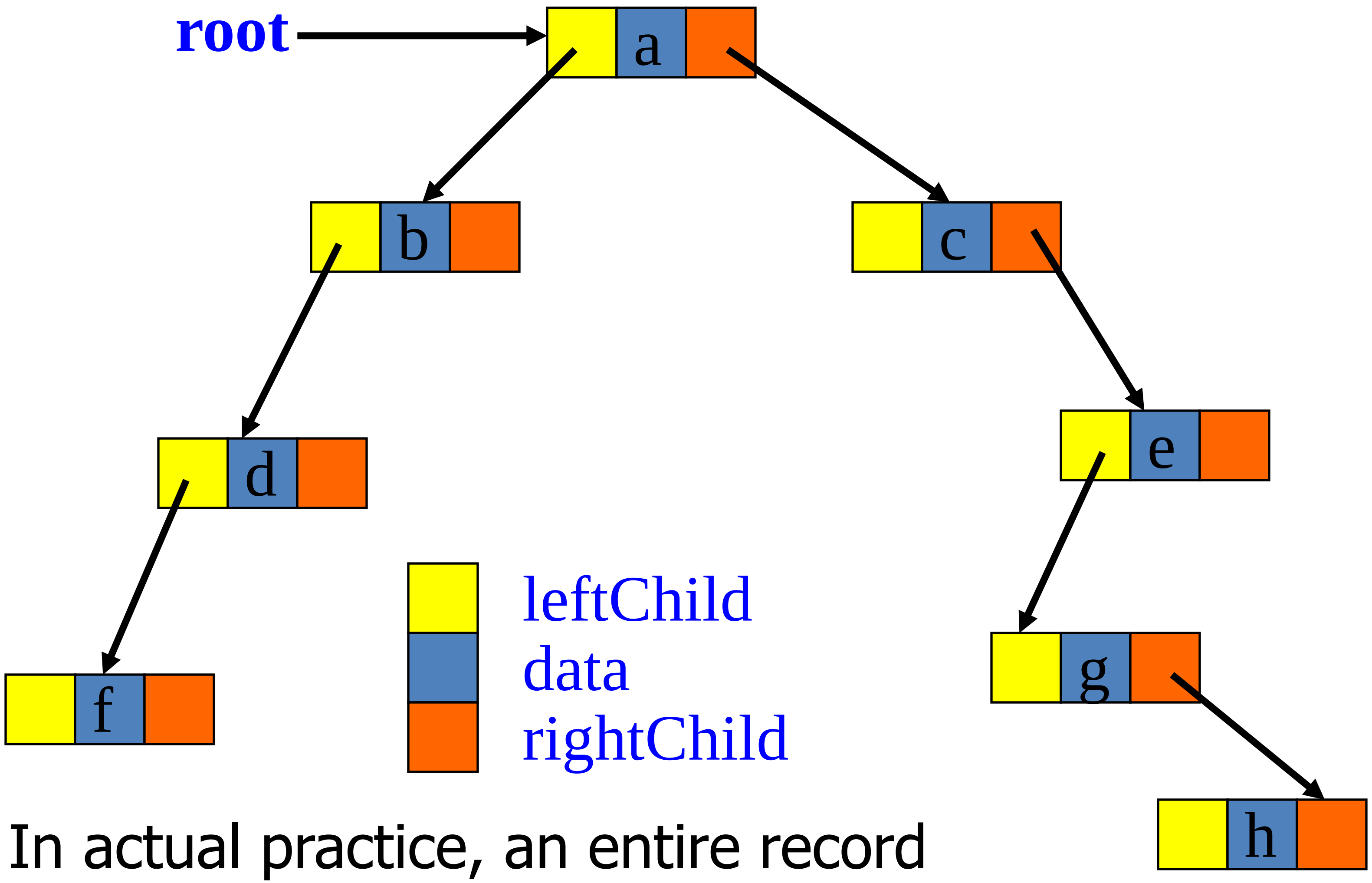
  We also need a pointer variable root or T

In actual practice, an entire record may be stored at the node

Binary Tree

```
struct node {
        int  data;
        struct node *rchild;
        struct node *lchild;
  } ;
typedef struct node* ptrnode;
ptrnode root;
```

Linked Representation Example



In actual practice, an entire record may be stored at the node

Some Binary Tree Operations

- Determine the height.
- Determine the number of nodes.
- Make a clone.
- Determine if two binary trees are clones.
- Display the binary tree.
- Evaluate the arithmetic expression represented by a binary tree.
- Obtain the infix form of an expression.
- Obtain the prefix form of an expression.
- Obtain the postfix form of an expression.

Binary Tree Traversal

- Many binary tree operations are done by performing a traversal of the binary tree
- In a traversal, each element of the binary tree is visited exactly once
- During the visit of an element, all action (make a clone, display, evaluate the operator, etc.) with respect to this element is taken

Binary Tree Traversal Methods

- Preorder
- Inorder
- Postorder
- Level order

Traversing Binary Trees

- 3 standard ways of traversing:
- **Preorder**: Process root R

  Traverse the left subtree of R in preorder

  Traverse the right subtree of R in preorder
- **Inorder**: Traverse the left subtree of R in inorder

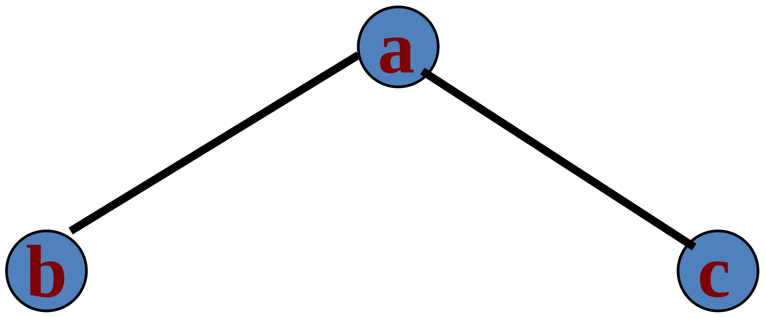
  Process root R

  Traverse the right subtree of R in inorder
- **Postorder**: Traverse the left subtree of R in postorder

  Traverse the right subtree of R in postorder

  Process root R

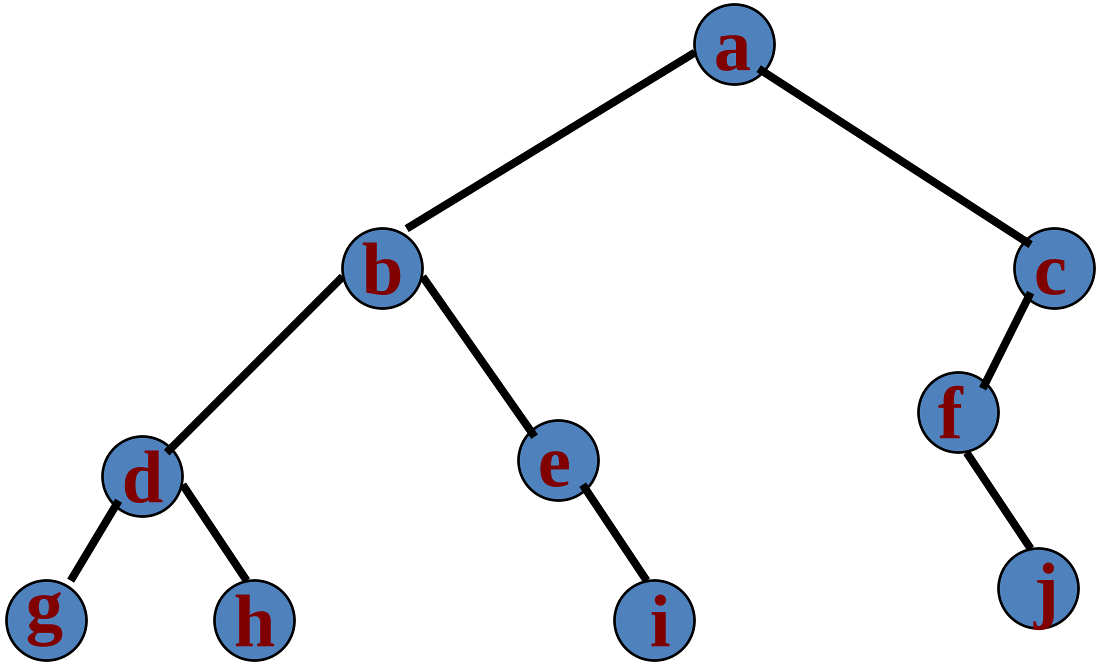
Preorder Example (visit = print)



a b c

Preorder: Process root R
Traverse the left subtree of R in preorder
Traverse the right subtree of R in preorder

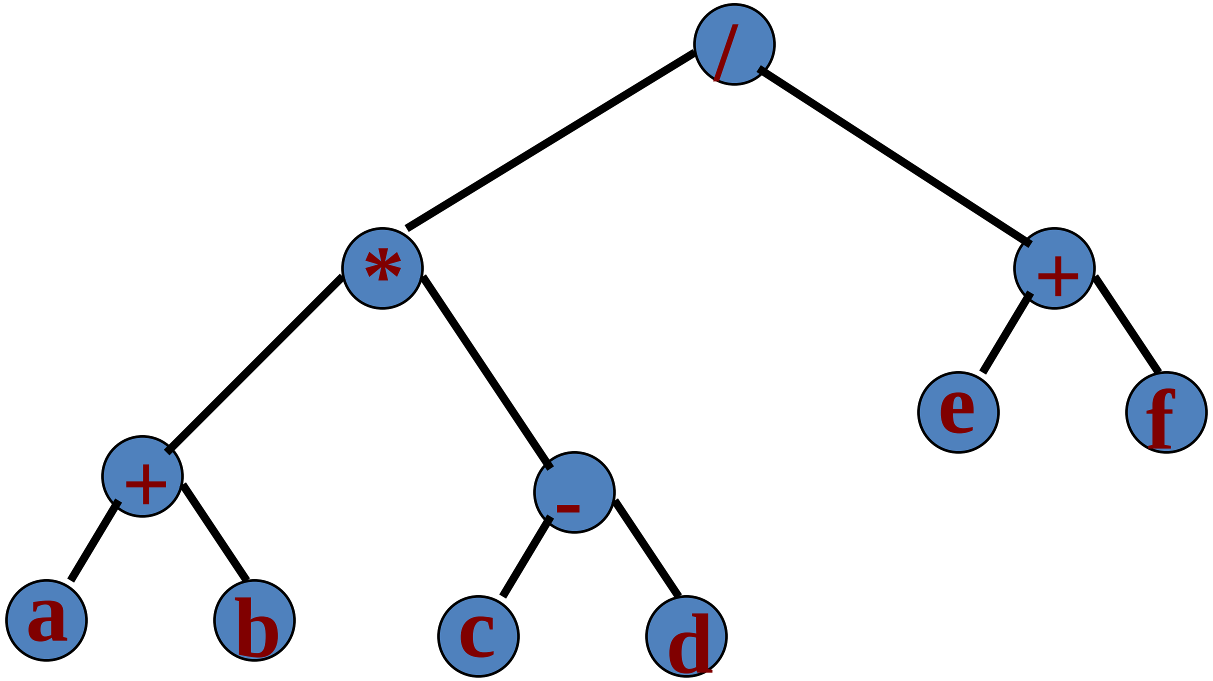
Preorder Example (visit = print)



a b d g h e i c f j

Preorder: Process root R
Traverse the left subtree of R in preorder
Traverse the right subtree of R in preorder

Preorder of Expression Tree



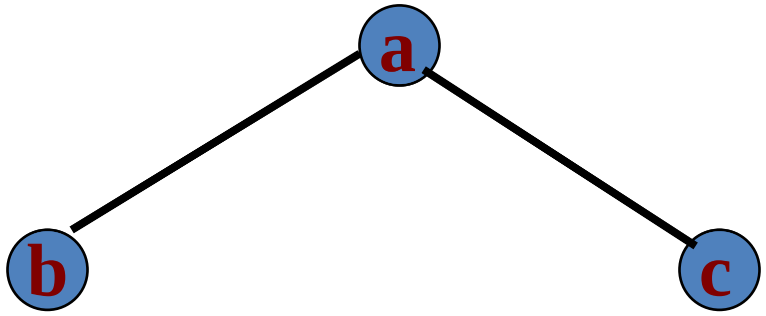
/ * + a b - c d + e f

Gives prefix form of expression!

Preorder Traversal

```
Void  preOrder(ptrnode root)
 {
  if (root != NULL)
  {
     visit(root);
     preOrder(root->lchild);
     preOrder(root->rchild);
  }
}
```

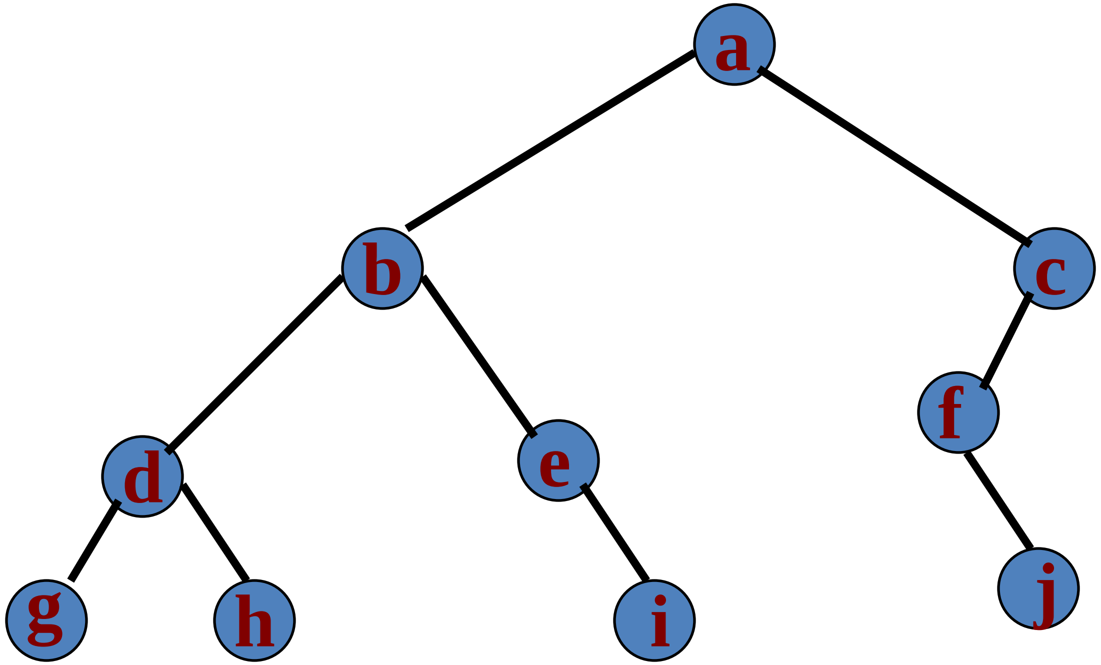
Inorder Example (visit = print)



b a c

Inorder: Traverse the left subtree of R in inorder
Process root R
Traverse the right subtree of R in inorder

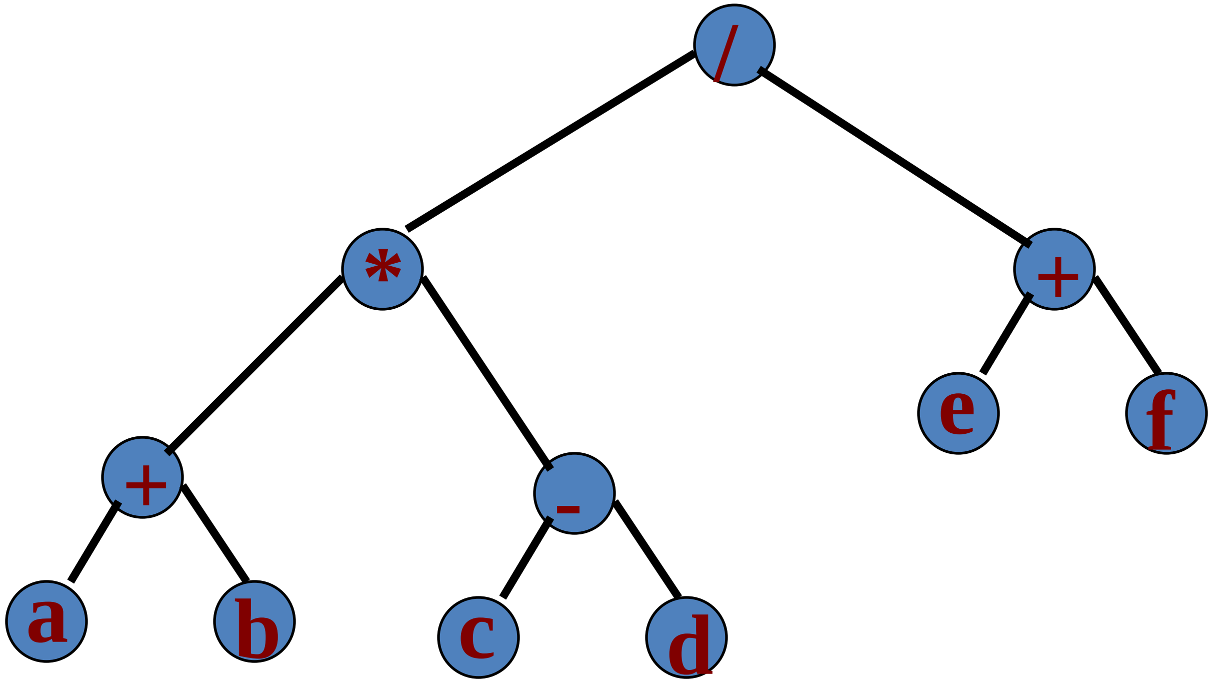
Inorder Example (visit = print)



g d h b e i a f j c

Inorder: Traverse the left subtree of R in inorder
Process root R
Traverse the right subtree of R in inorder

Inorder of Expression Tree



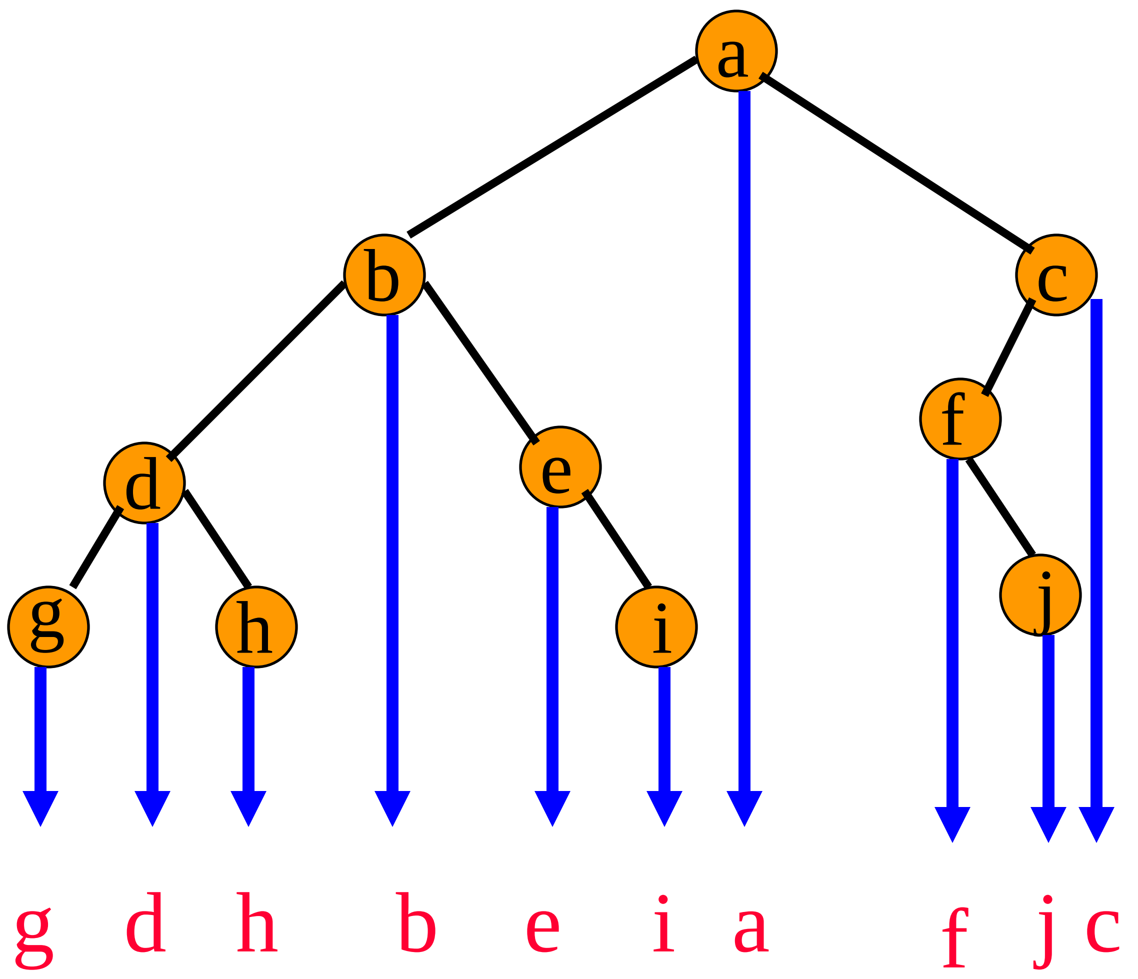
a + b * c - d / e + f

Gives infix form of expression!

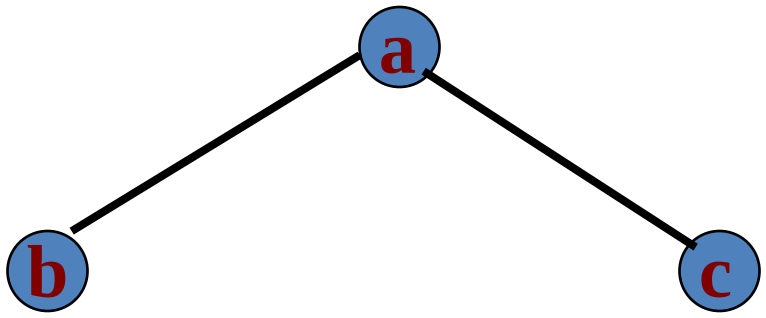
Inorder Traversal

```
void inOrder(ptrnode root)
{
  if (root != NULL)
  {
    inOrder(root->lchild);
    visit(root);
    inOrder(root->rchild);
  }
}
```

Inorder By Projection



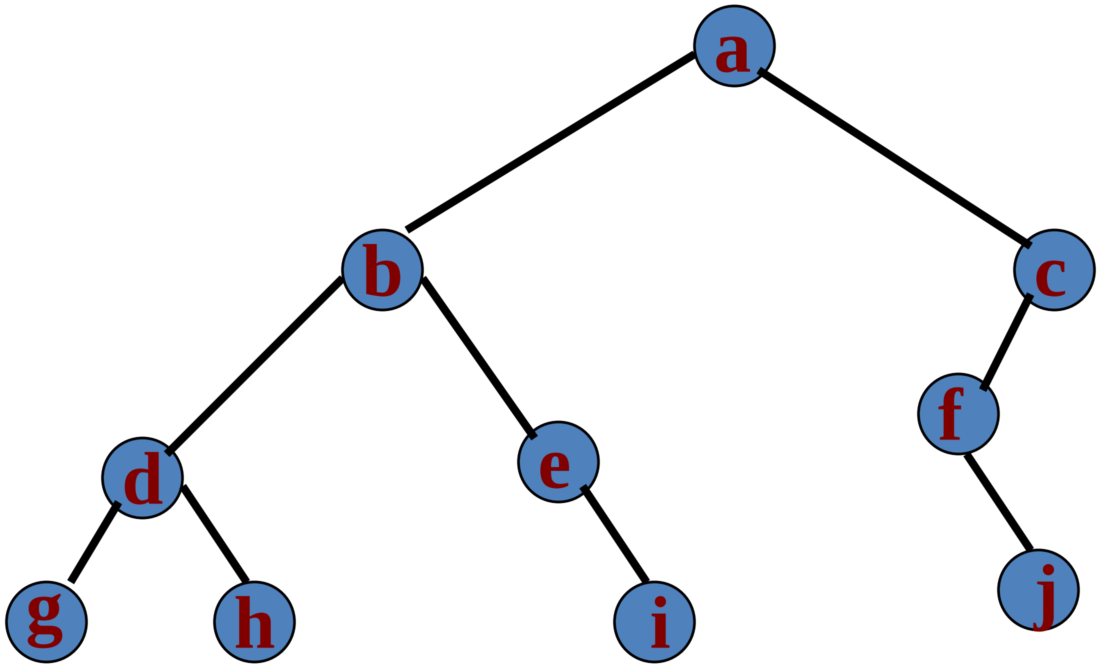
Postorder Example (visit = print)



b c a

Postorder: Traverse the left subtree of R in postorder
Traverse the right subtree of R in postorder
Process root R

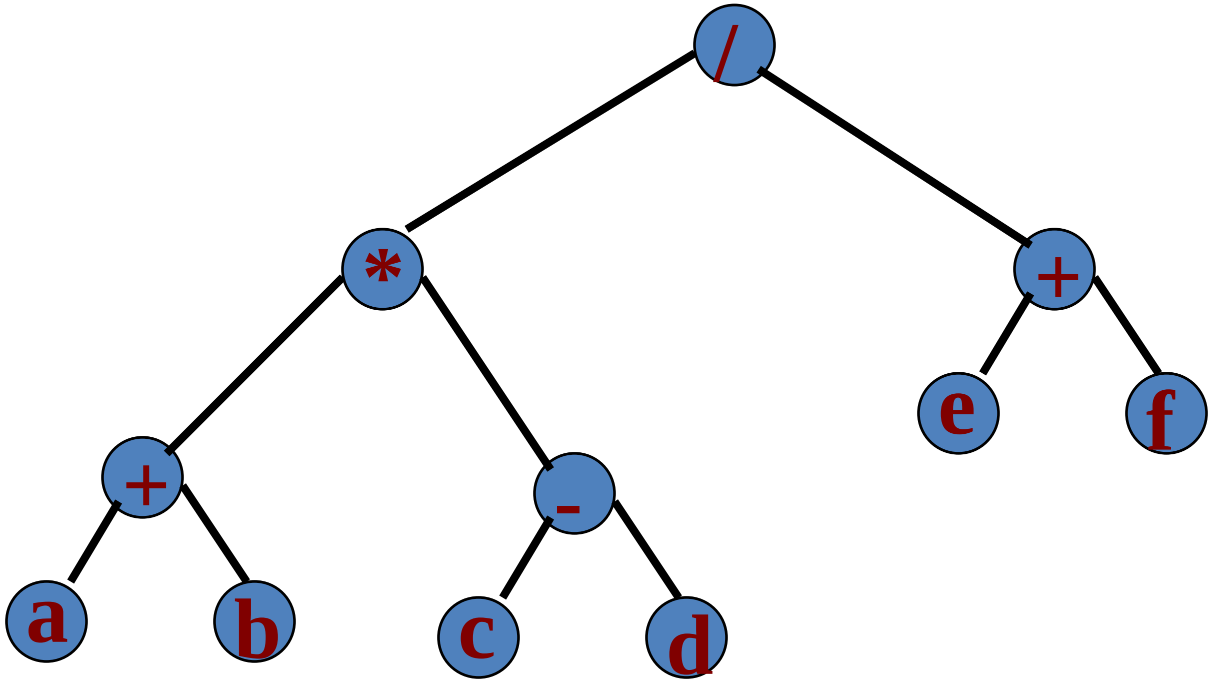
Postorder Example (visit = print)



g h d i e b j f c a

Postorder: Traverse the left subtree of R in postorder
Traverse the right subtree of R in postorder
Process root R

Postorder of Expression Tree



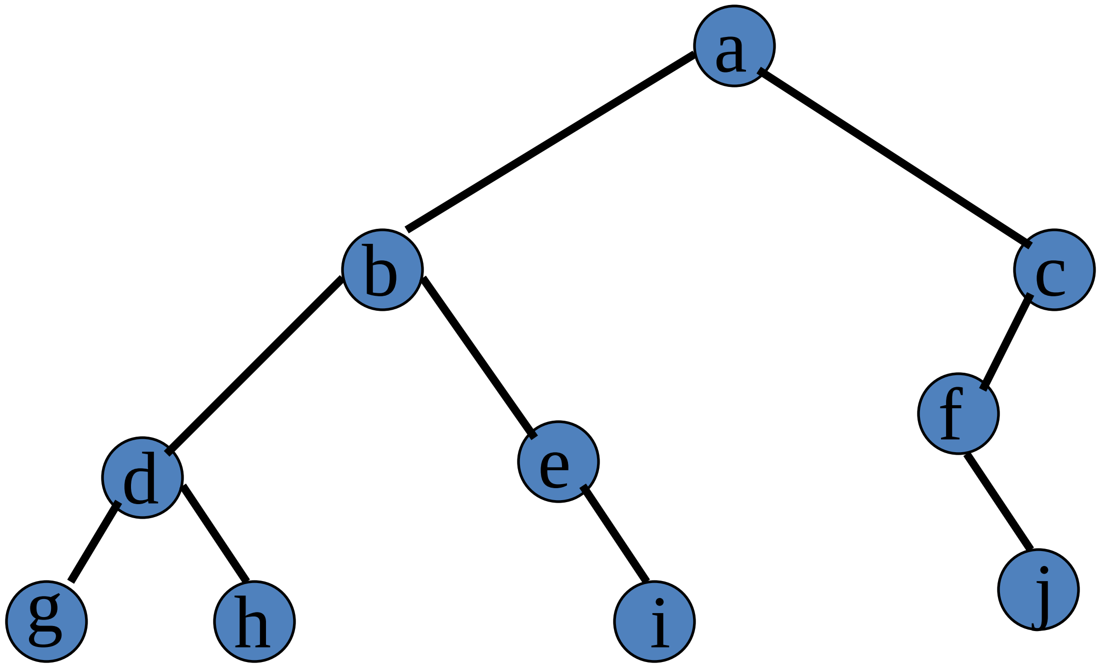
a b + c d - * e f + /

Gives postfix form of expression!

Postorder Traversal

```
void postOrder(ptrnode root)
{
   if (root != NULL)
   {
      postOrder(root->lchild);
      postOrder(root->rchild);
      visit(root);
   }
}
```

Level-order Example (Visit = print)

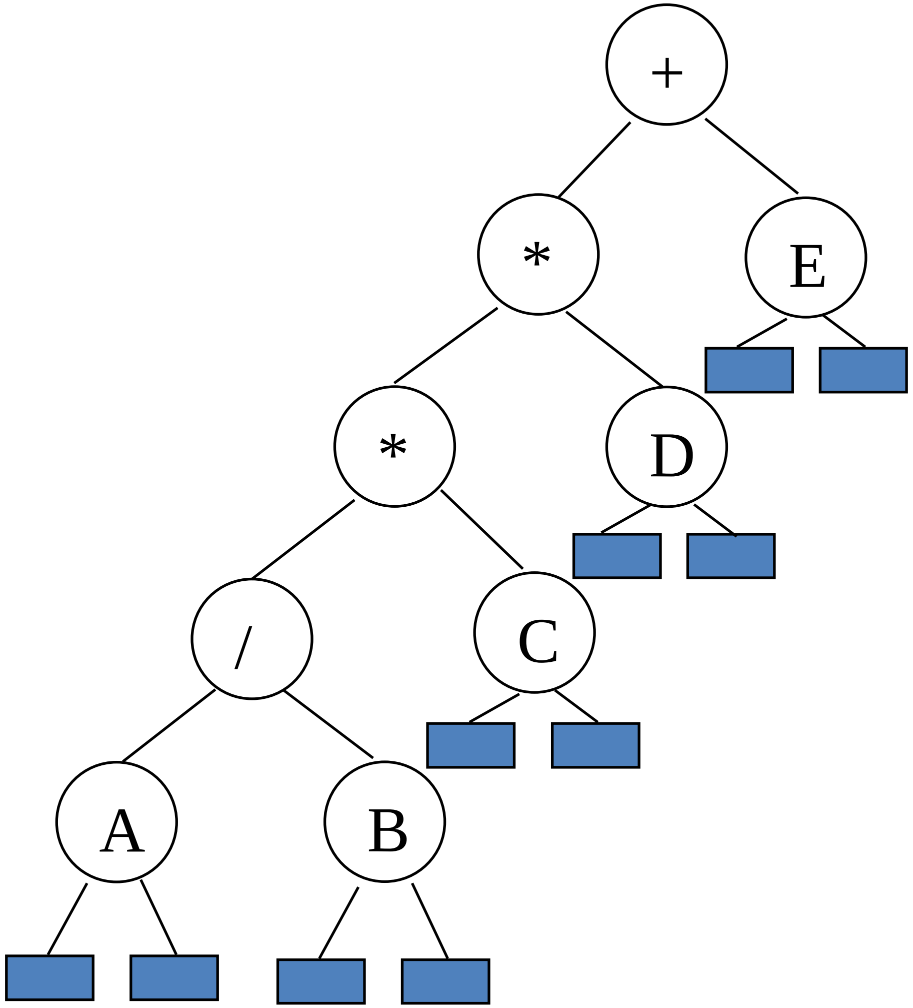


a b c d e f g h i j

Level Order

while (root != NULL)
{
visit node pointed at by root and put its children on a FIFO queue;
if FIFO queue is empty, set root = NULL;
otherwise, delete a node from the FIFO queue and call it root;
}

Another example of Expression Tree Using BT



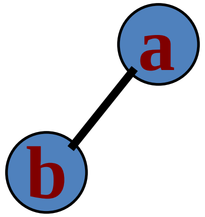
inorder traversal
A / B * C * D + E
infix expression
preorder traversal
+ * * / A B C D E
prefix expression
postorder traversal
A B / C * D * E +
postfix expression
level order traversal
+ * E * D / C A B

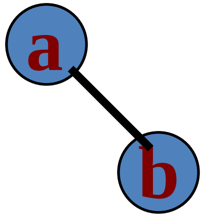
Binary Tree Construction

- Suppose that the elements in a binary tree are distinct
- Can you construct the binary tree from which a given traversal sequence came?
- When a traversal sequence has more than one element, the binary tree is not uniquely defined
- Therefore, the tree from which the sequence was obtained cannot be reconstructed uniquely

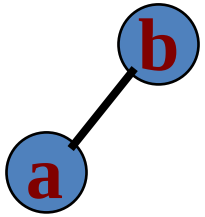
Some Examples

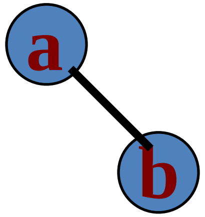
preorder = ab



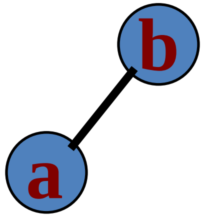


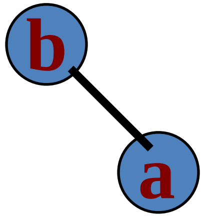
inorder = ab



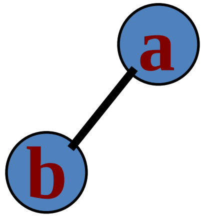


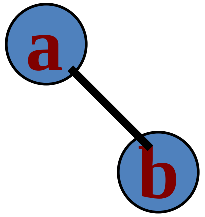
postorder = ab





level order = ab





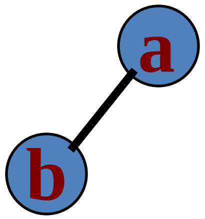
Binary Tree Construction

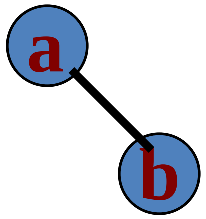
- Can you construct the binary tree, given two traversal sequences?
- Depends on which two sequences are given.

Preorder and Postorder

preorder = ab

postorder = ba

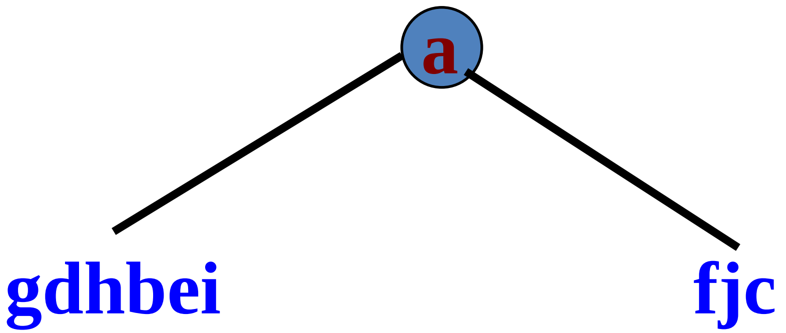




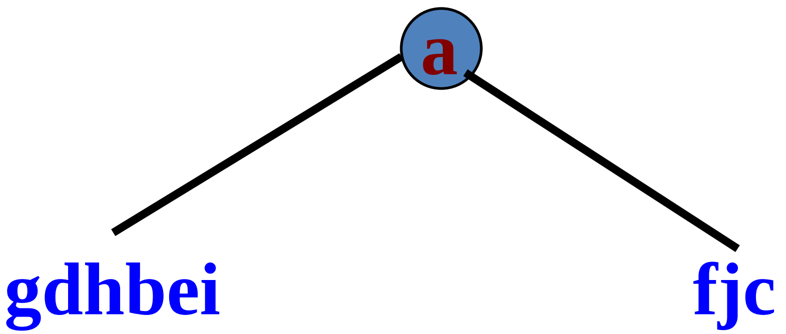
- Preorder and postorder do not uniquely define a binary tree
- Nor do preorder and level order (same example)
- Nor do postorder and level order (same example)

Inorder and Preorder

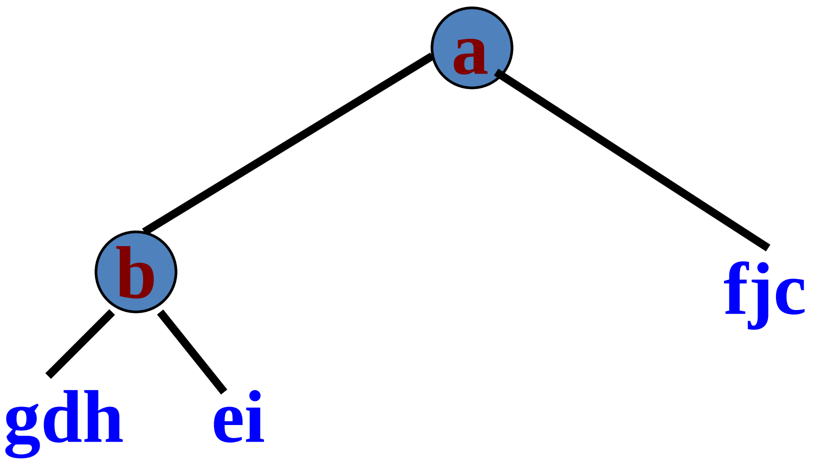
- inorder = g d h b e i a f j c
- preorder = a b d g h e i c f j
- Scan the preorder left to right using the inorder to separate left and right subtrees
- a is the root of the tree; gdhbei are in the left subtree; fjc are in the right subtree



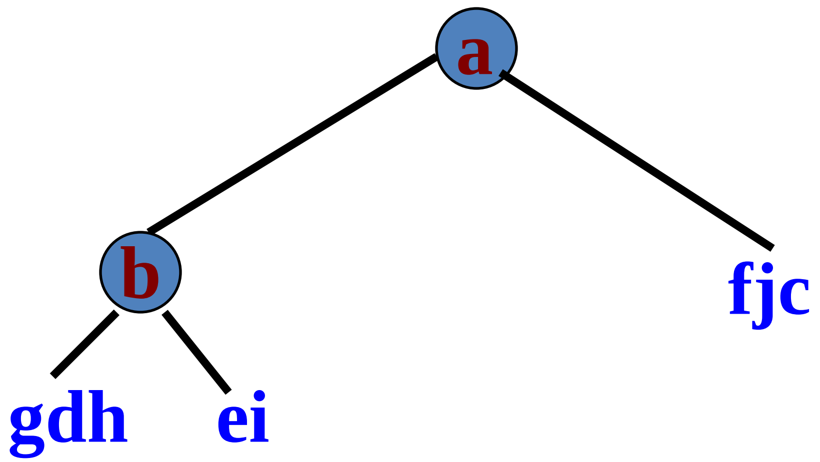
Inorder and Preorder



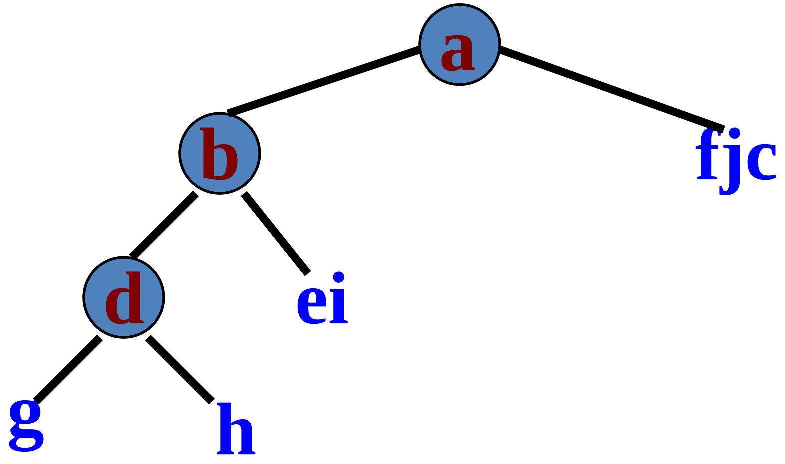
- preorder = b d g h e i c f j
- inorder = g d h b e i a f j c
- b is the next root; gdh are in the left subtree; ei are in the right subtree



Inorder and Preorder



- preorder = d g h e i c f j
- inorder = g d h b e i a f j c
- d is the next root; g is in the left subtree; h is in the right subtree



Inorder and Postorder

- Scan postorder from right to left using inorder to separate left and right subtrees.
- inorder = g d h b e i a f j c
- postorder = g h d i e b j f c a
- Tree root is a; gdhbei are in left subtree; fjc are in right subtree.

Inorder and Level Order

- Scan level order from left to right using inorder to separate left and right subtrees.
- inorder = g d h b e i a f j c
- level order = a b c d e f g h i j
- Tree root is a; gdhbei are in left subtree; fjc are in right subtree.

References

Source: www.programming.im.ncnu.edu.tw/HorowitzC2e/