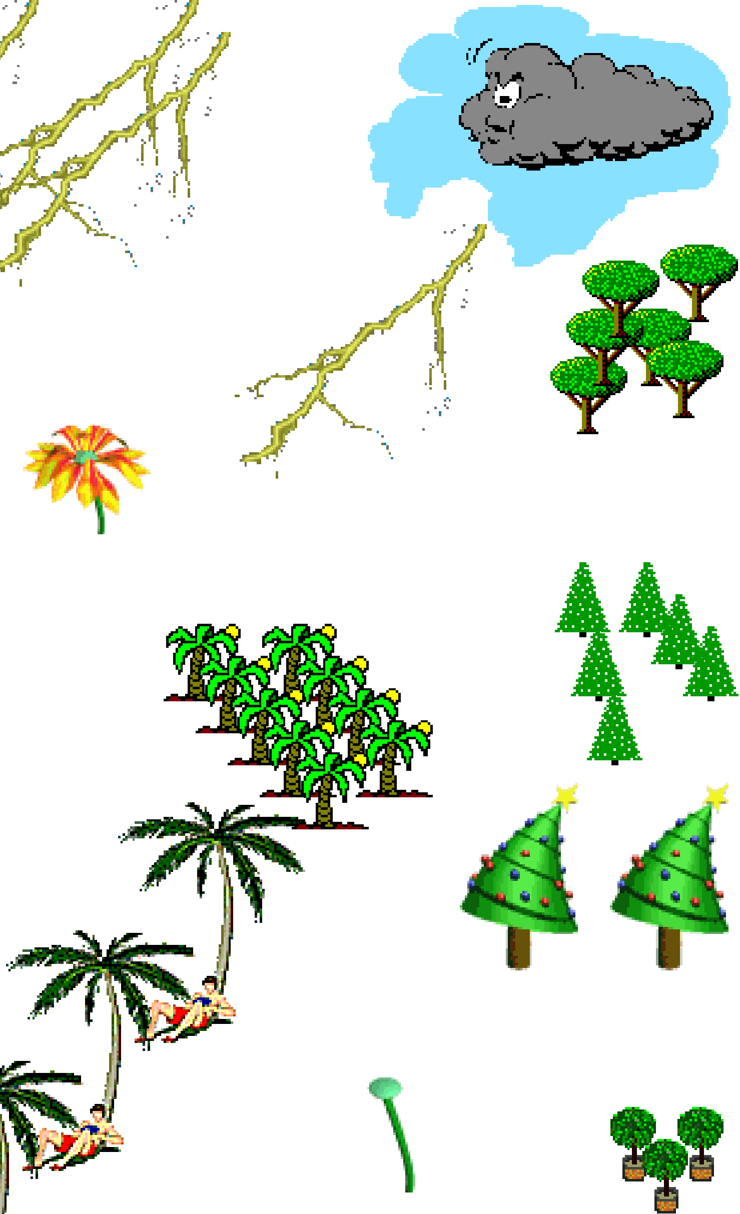
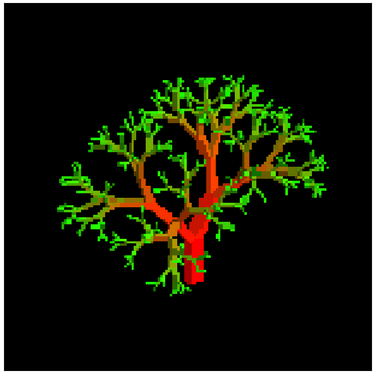
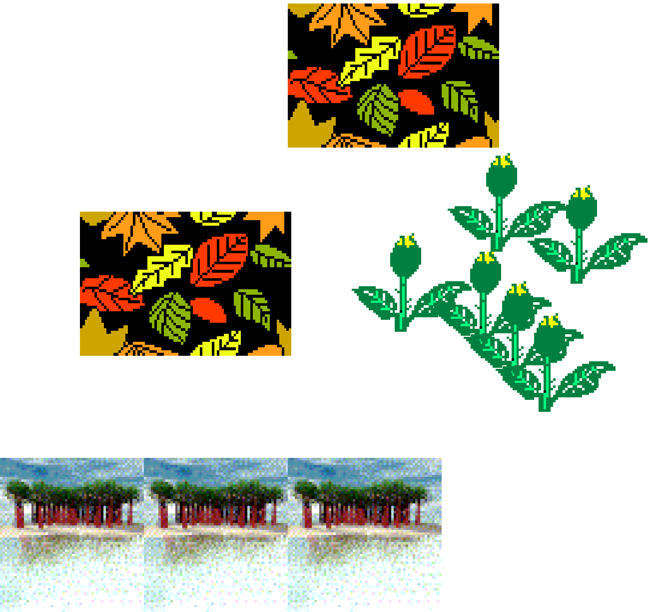
Chapter Six

Trees

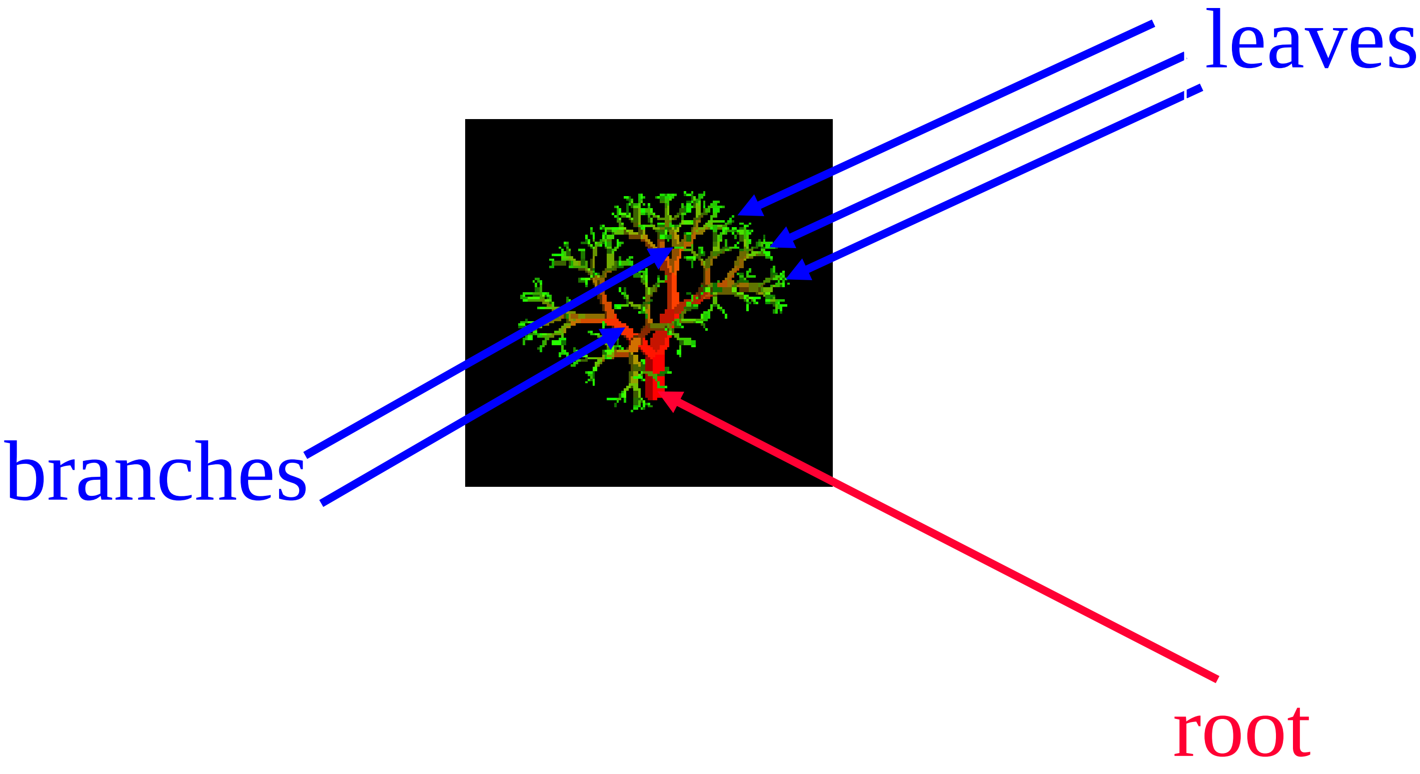
Trees



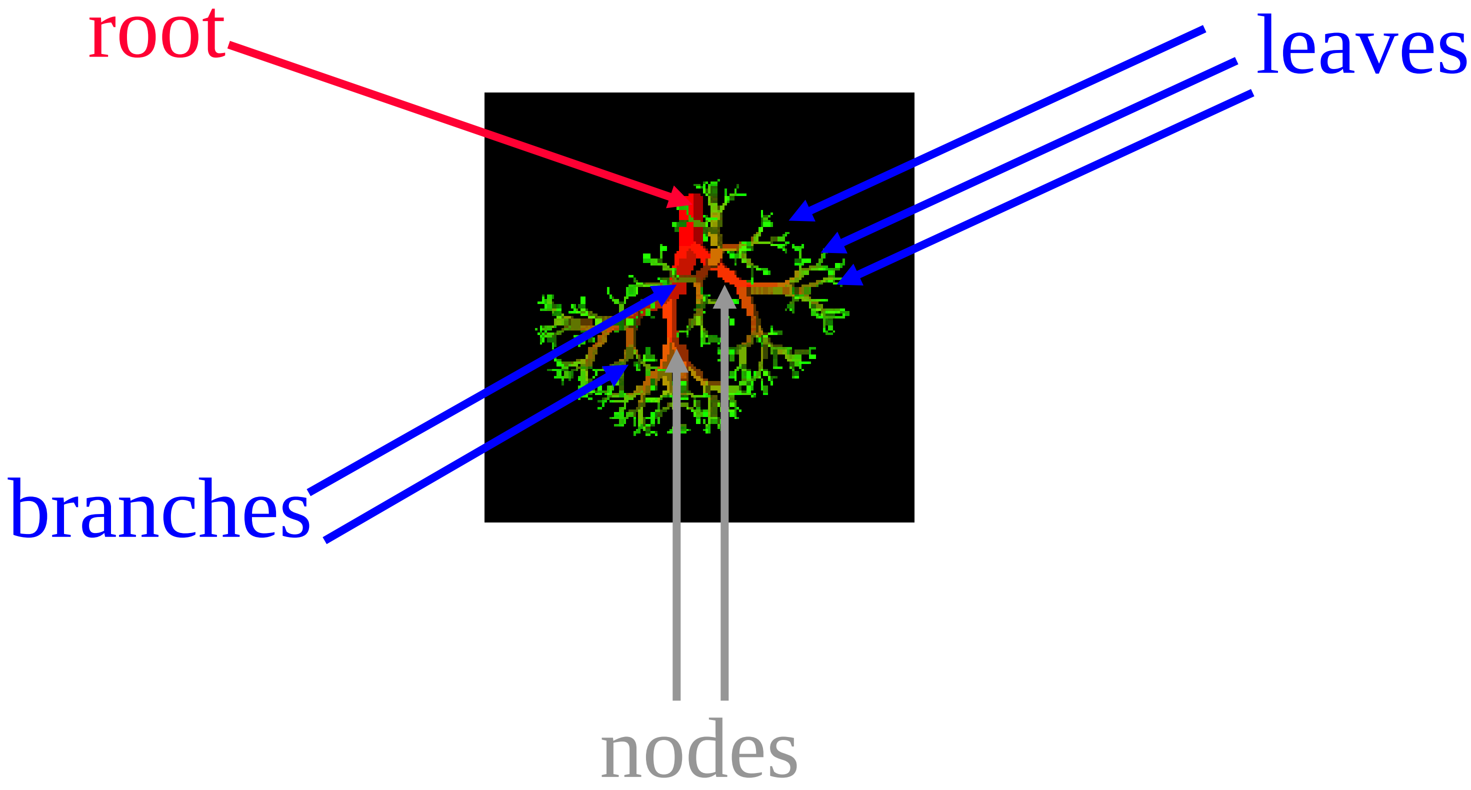




Nature Lover's View Of A Tree



Computer Scientist's View



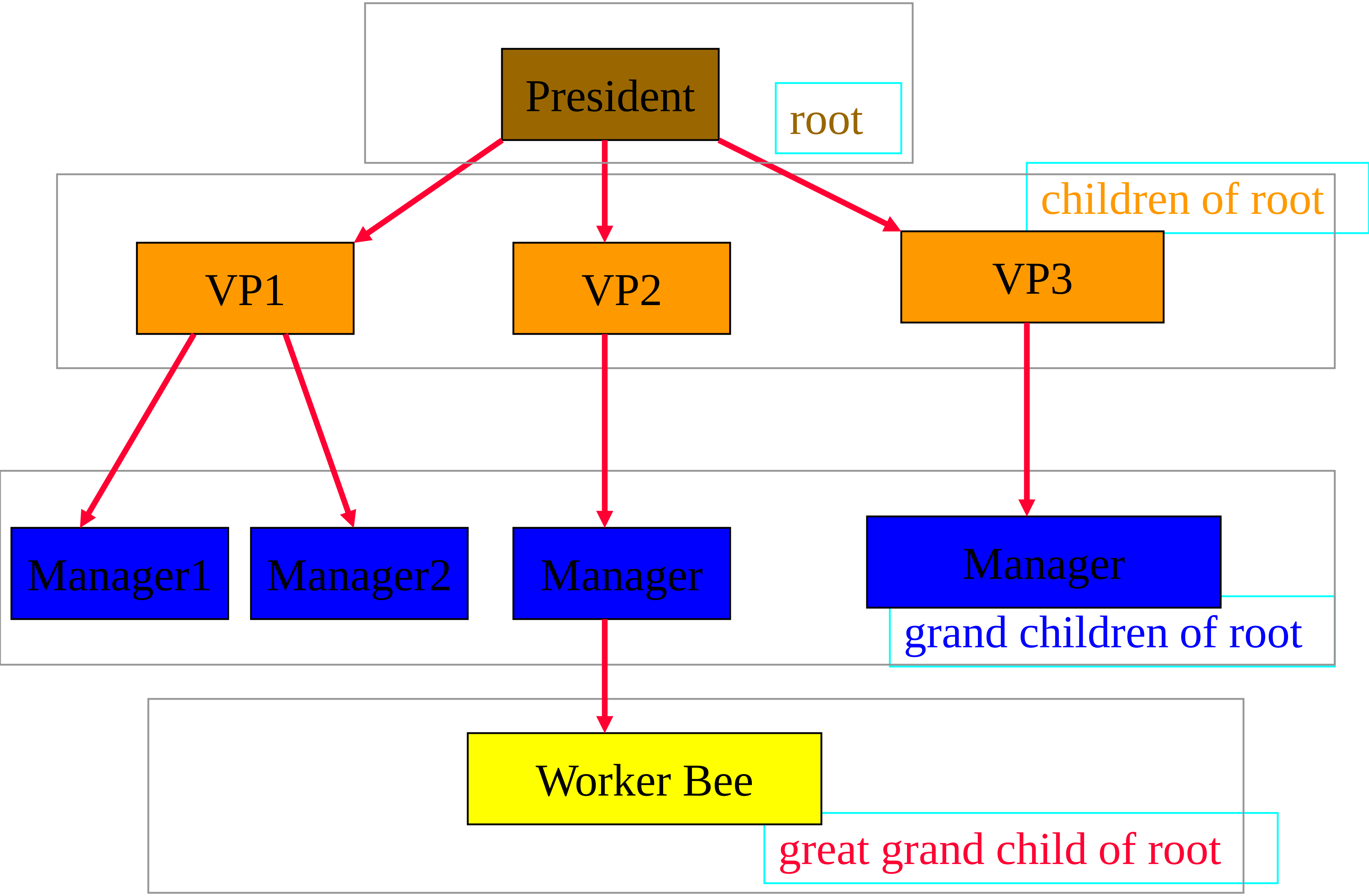
Linear Lists And Trees

- 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.

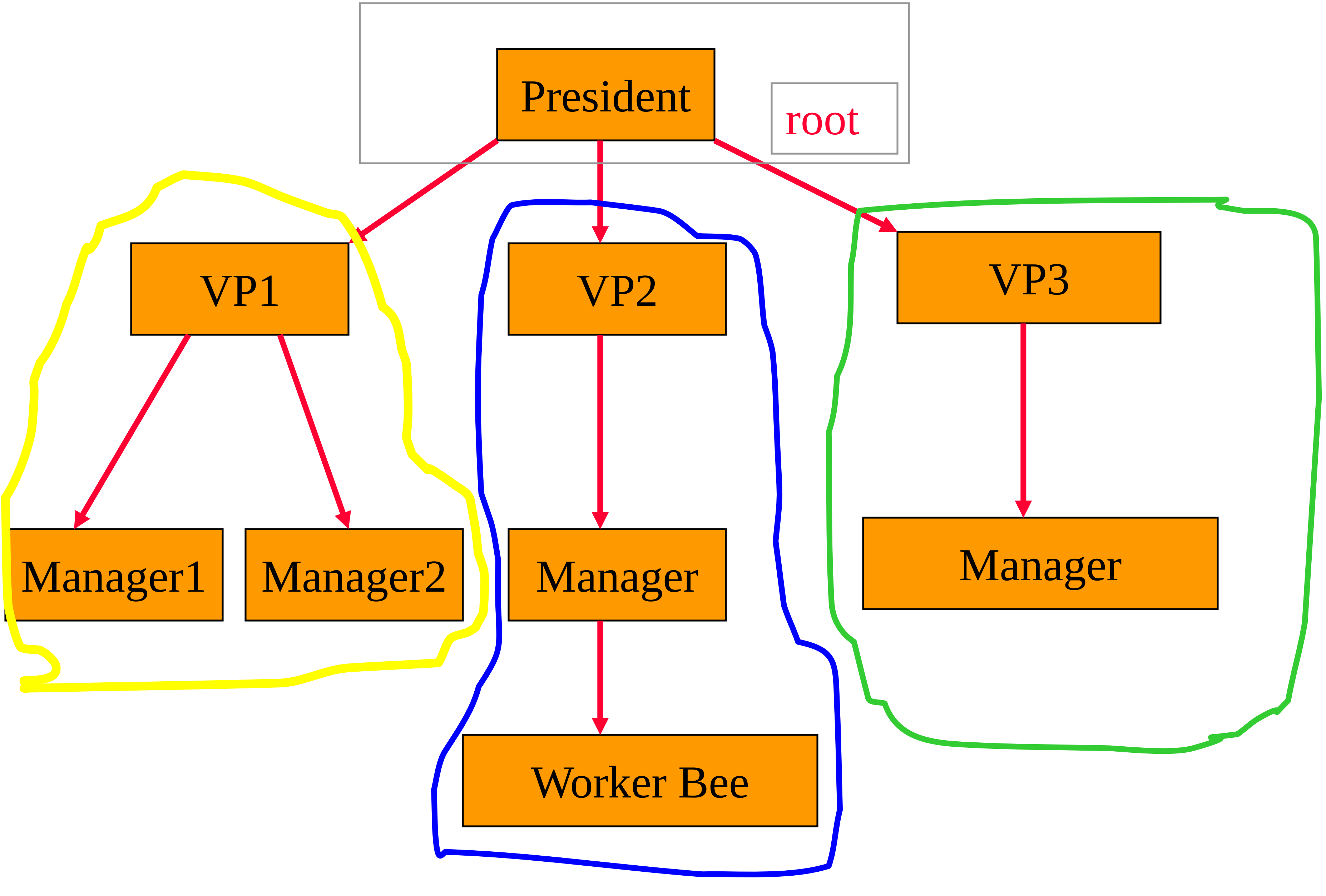
Example Tree



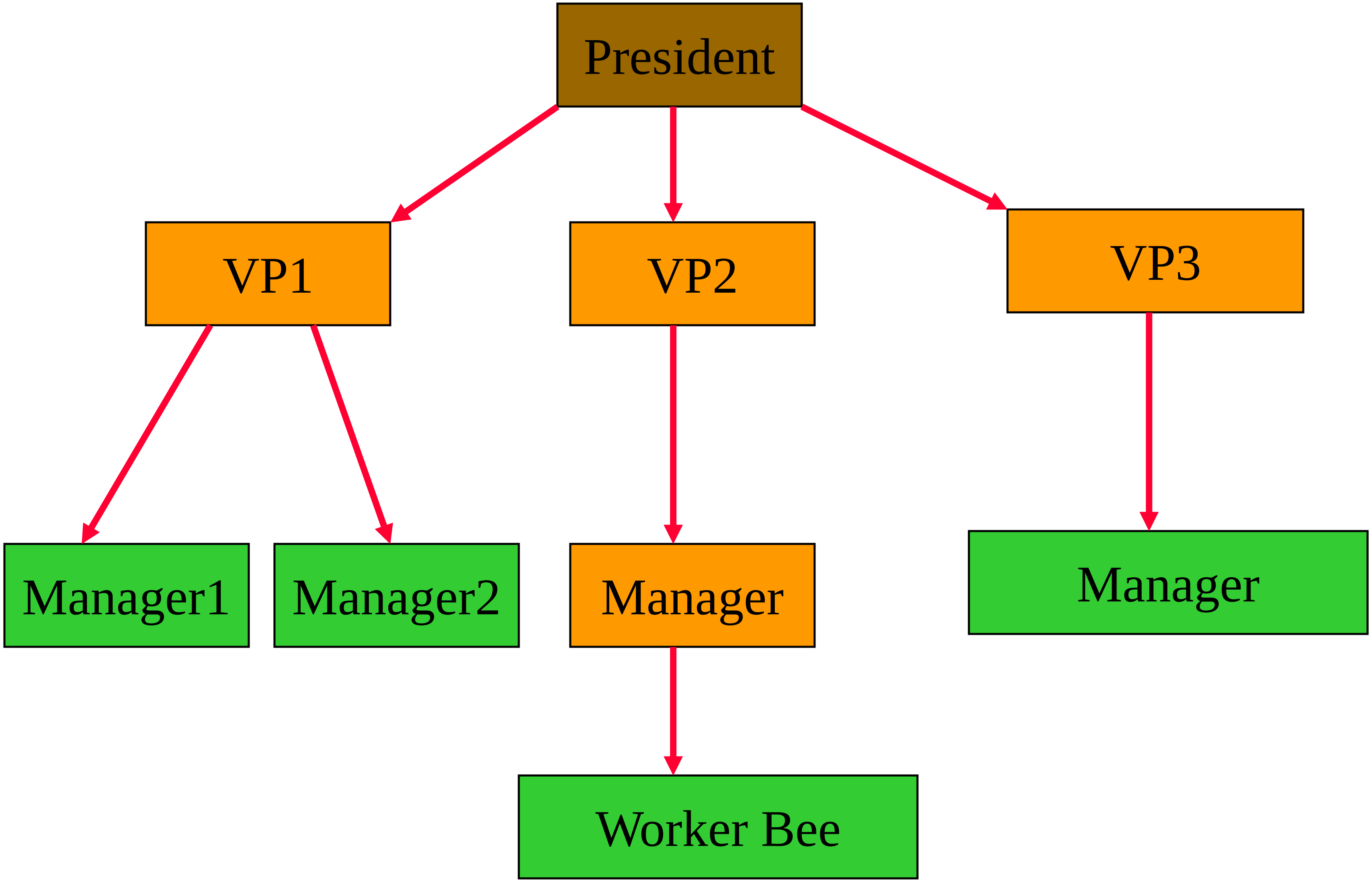
Definition

- A tree t is a finite nonempty set of elements.
- One of these elements is called the root.
- The remaining elements, if any, are partitioned into trees, which are called the subtrees of t.

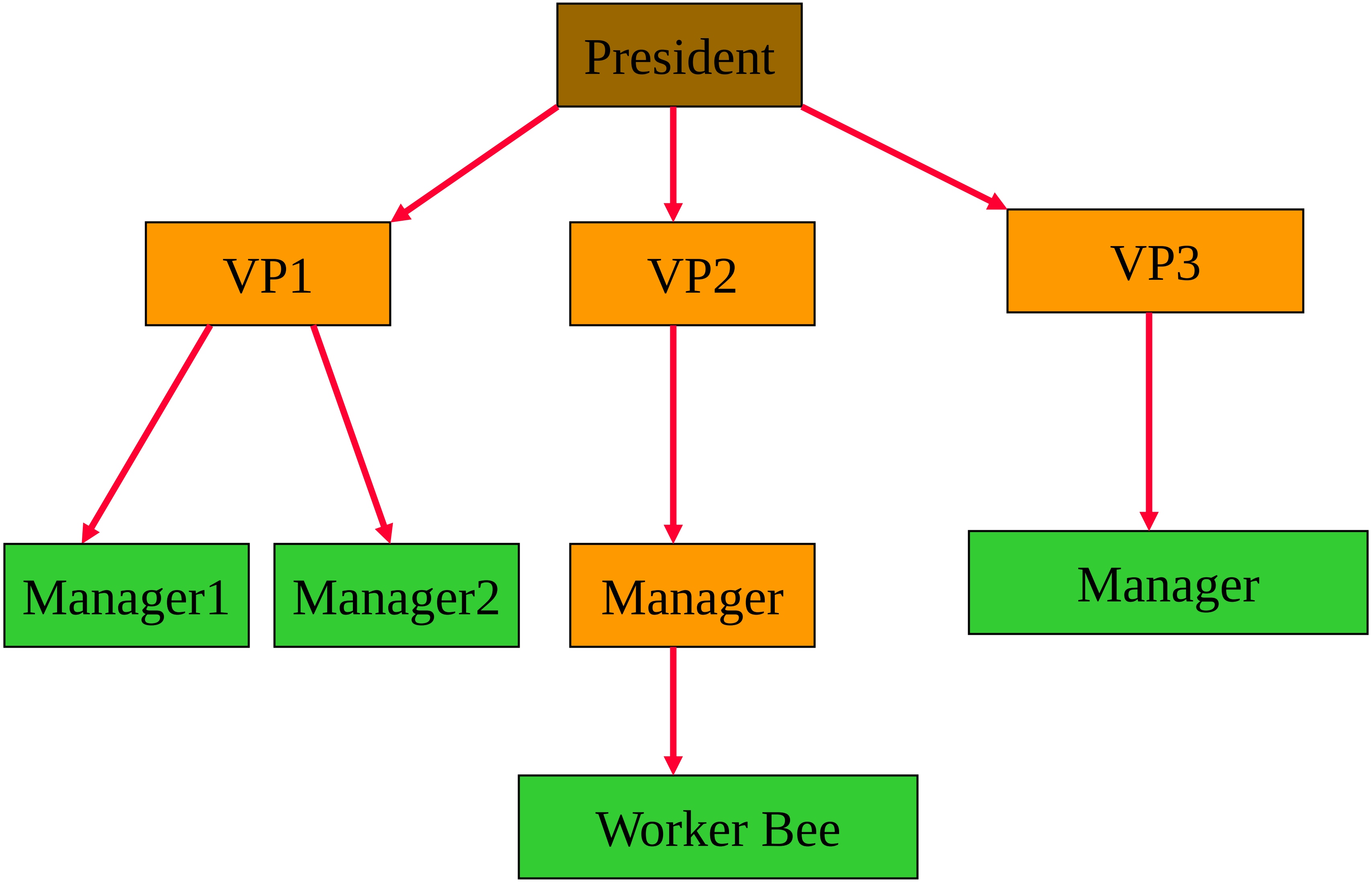
Subtrees


President
root
VP1
VP2
VP3
Manager1
Manager2
Manager
Manager
Worker Bee

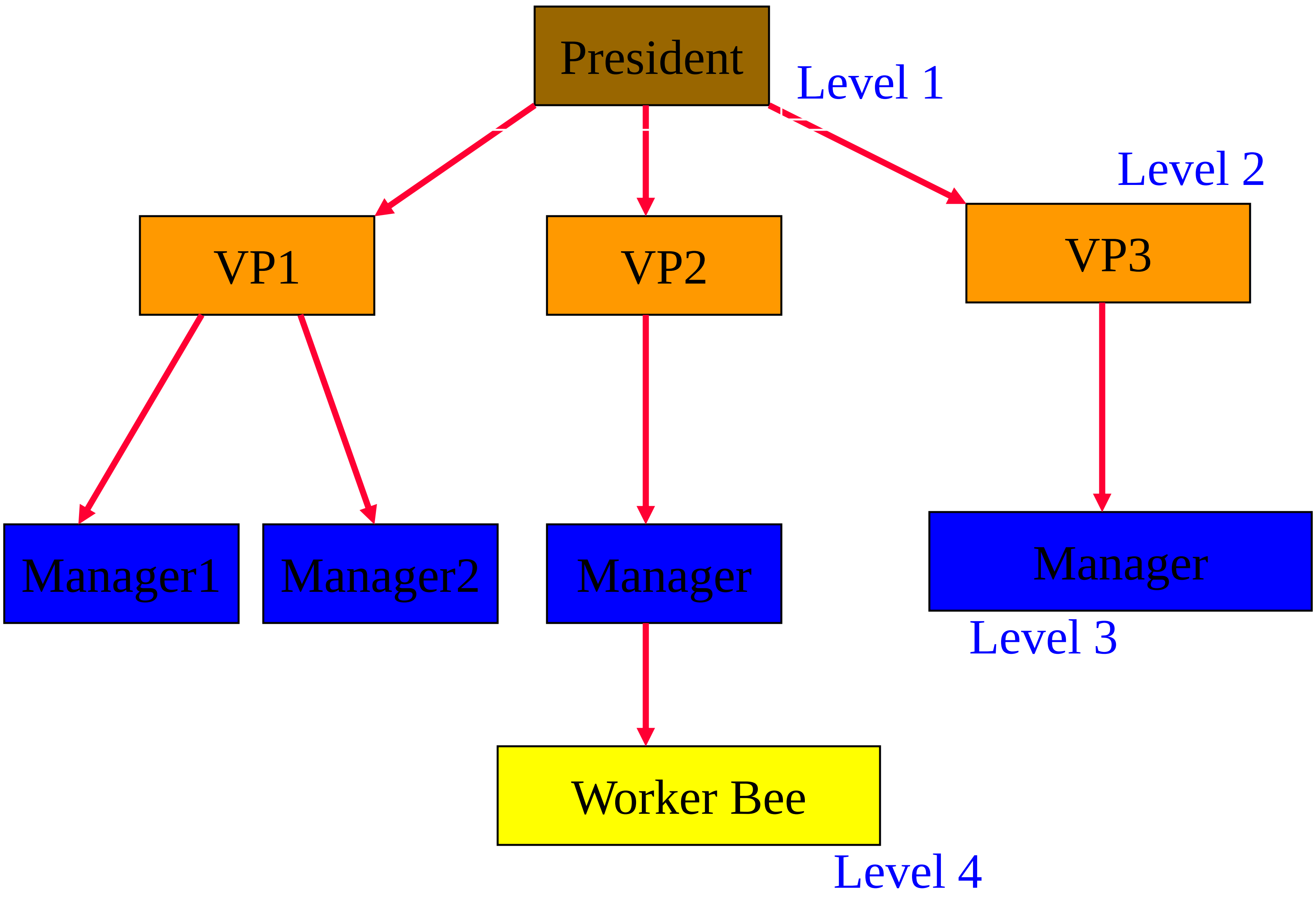
Leaves


President
VP1
VP2
VP3
Manager1
Manager2
Manager
Manager
Worker Bee

Parent, Grandparent, Siblings, Ancestors, Descendants



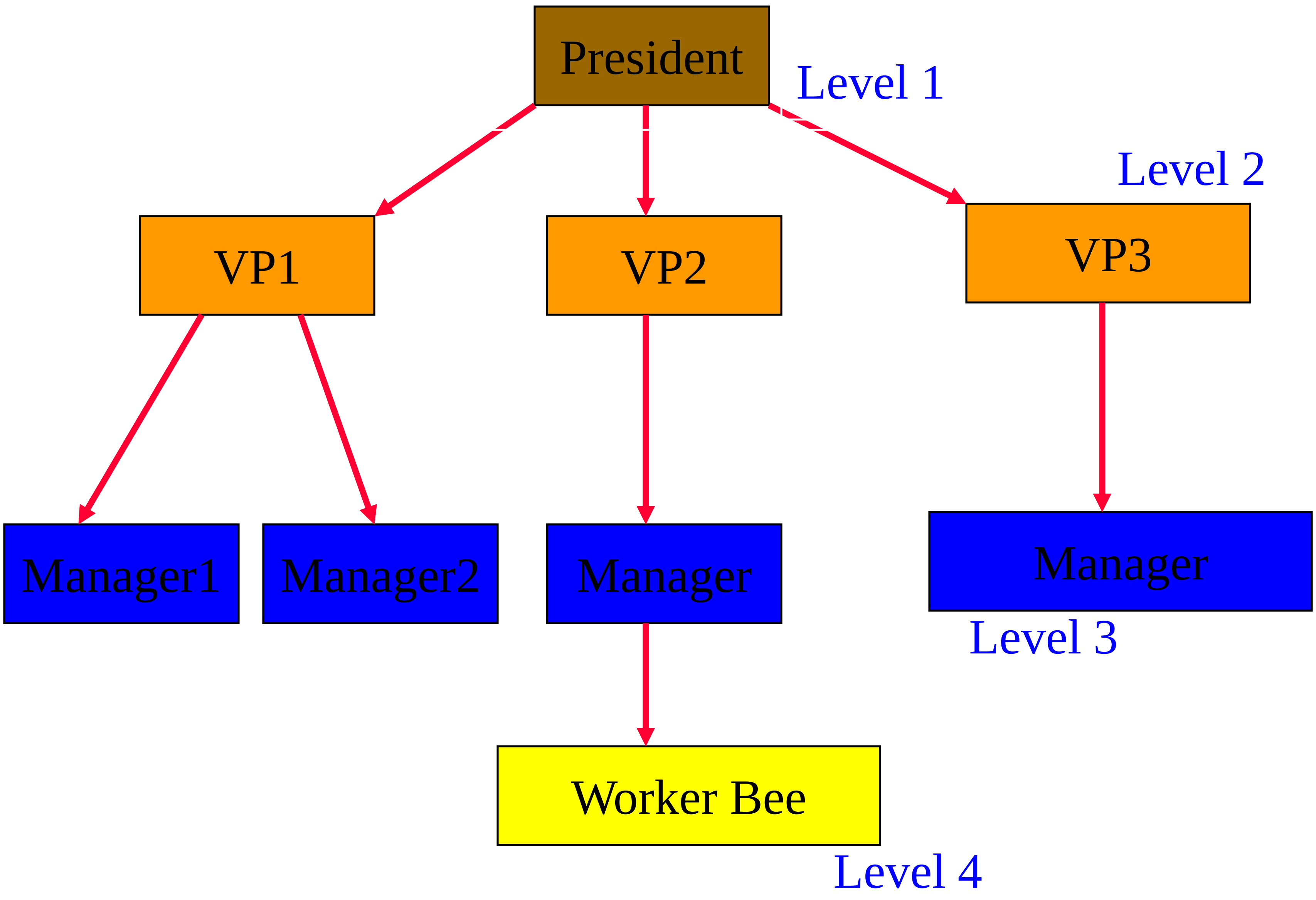
Levels


President
Level 1
Level 2
VP1
VP2
VP3
Manager1
Manager2
Manager
Manager
Level 3
Worker Bee
Level 4

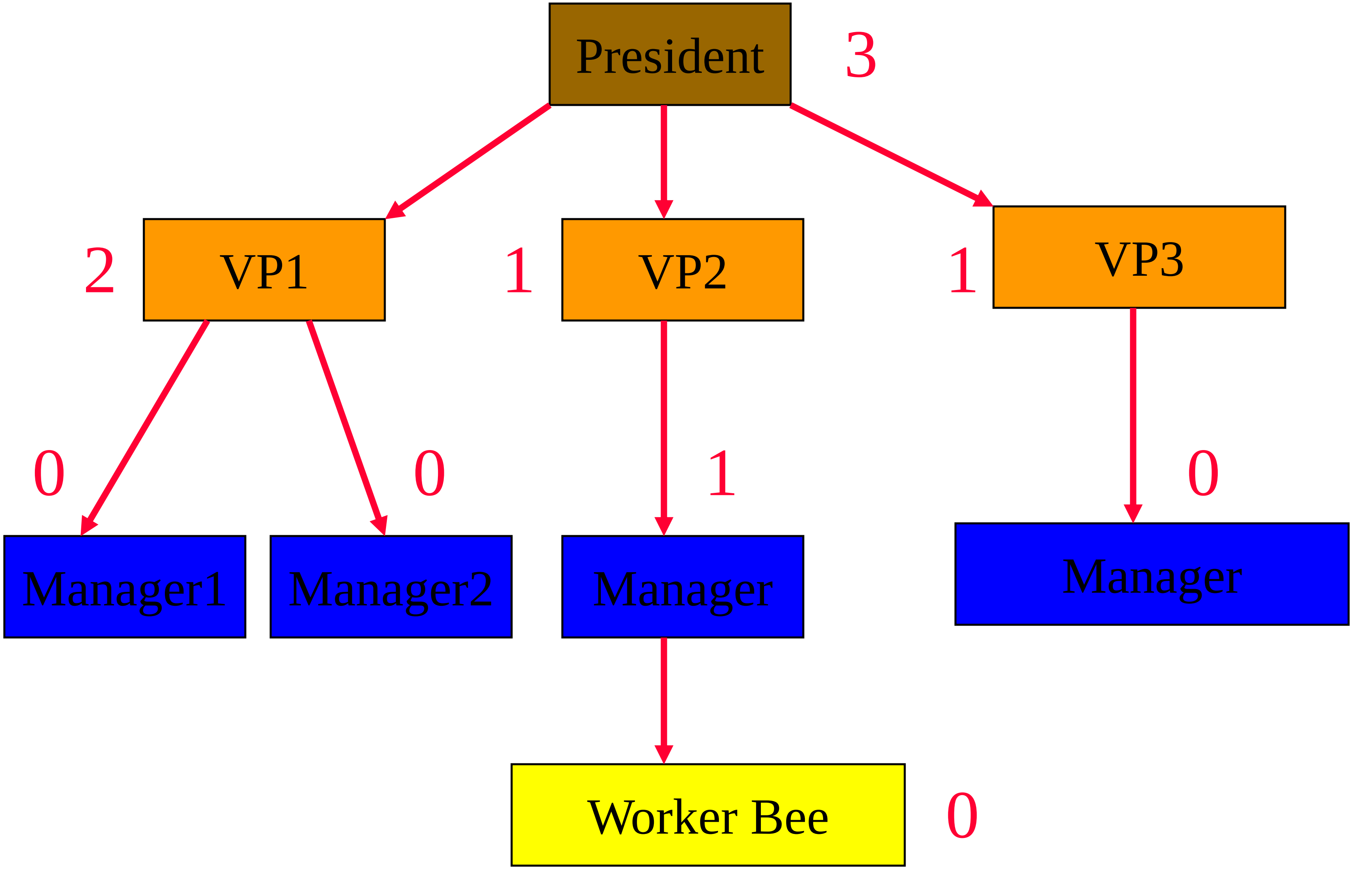
Caution

- Some texts start level numbers at 0 rather than at 1.
- Root is at level 0.
- Its children are at level 1.
- The grand children of the root are at level 2.
- And so on.
- We shall number levels with the root at level 1.

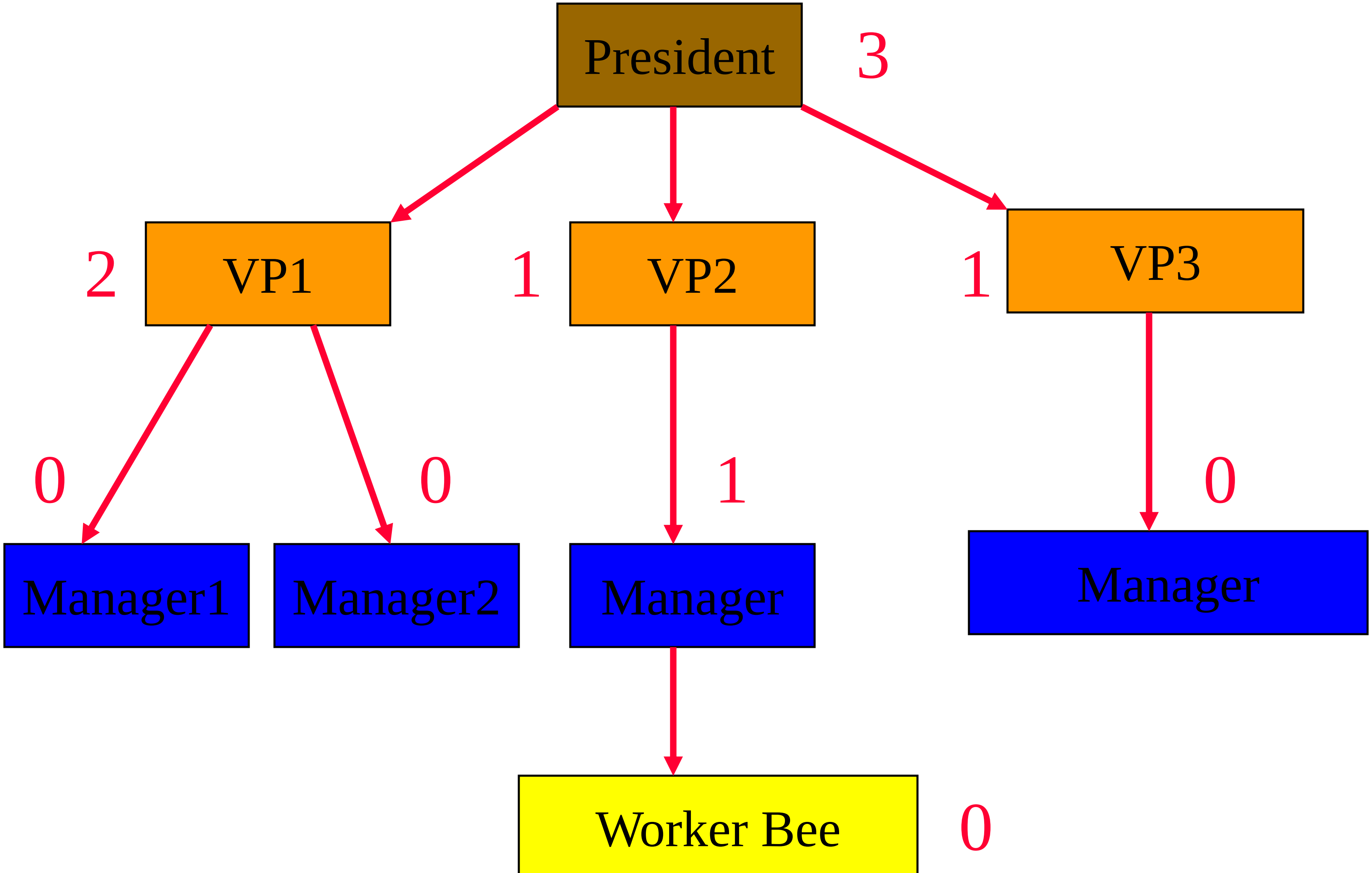
height = depth = number of levels



Node Degree = Number Of Children



Tree Degree = Max Node Degree



Degree of tree = 3.

Binary Tree

- Finite (possibly empty) collection of elements.
- A nonempty binary tree has a root element.
- The remaining elements (if any) are partitioned into two binary trees.
- These are called the left and right subtrees of the binary tree.

Differences Between A Tree & A 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.
- A binary tree may be empty; a tree cannot be empty.

Differences Between A Tree & A Binary Tree

- The subtrees of a binary tree are ordered; those of a tree are not ordered.

- Are different when viewed as binary trees.
- Are the same when viewed as trees.

Arithmetic Expressions

- (a + b) * (c + d) + e – f/g*h + 3.25
- Expressions comprise three kinds of entities.
  - Operators (+, -, /, *).
  - Operands (a, b, c, d, e, f, g, h, 3.25, (a + b), (c + d), etc.).
  - Delimiters ((,)).

Operator Degree

- Number of operands that the operator requires.
- Binary operator requires two operands.
  - a + b
  - c / d
  - e - f
- Unary operator requires one operand.
  - + g
  - - h

Infix Form

- Normal way to write an expression.
- Binary operators come in between their left and right operands.
  - a * b
  - a + b * c
  - a * b / c
  - (a + b) * (c + d) + e – f/g*h + 3.25

Operator Priorities

- How do you figure out the operands of an operator?
  - a + b * c
  - a * b + c / d
- This is done by assigning operator priorities.
  - priority(*) = priority(/) > priority(+) = priority(-)
- When an operand lies between two operators, the operand associates with the operator that has higher priority.

Tie Breaker

- When an operand lies between two operators that have the same priority, the operand associates with the operator on the left.
  - a + b - c
  - a * b / c / d

Delimiters

- Subexpression within delimiters is treated as a single operand, independent from the remainder of the expression.
  - (a + b) * (c – d) / (e – f)

Infix Expression Is Hard To Parse

- Need operator priorities, tie breaker, and delimiters.
- This makes computer evaluation more difficult than is necessary.
- Postfix and prefix expression forms do not rely on operator priorities, a tie breaker, or delimiters.
- So it is easier for a computer to evaluate expressions that are in these forms.

Postfix Form

- The postfix form of a variable or constant is the same as its infix form.
  - a, b, 3.25
- The relative order of operands is the same in infix and postfix forms.
- Operators come immediately after the postfix form of their operands.
  - Infix = a + b
  - Postfix = ab+

Postfix Examples

- Infix = a + b * c
  - Postfix = a b c * +
- Infix = a * b + c
  - Postfix = a b * c +
- Infix = (a + b) * (c – d) / (e + f)
  - Postfix = a b + c d - * e f + /

Unary Operators

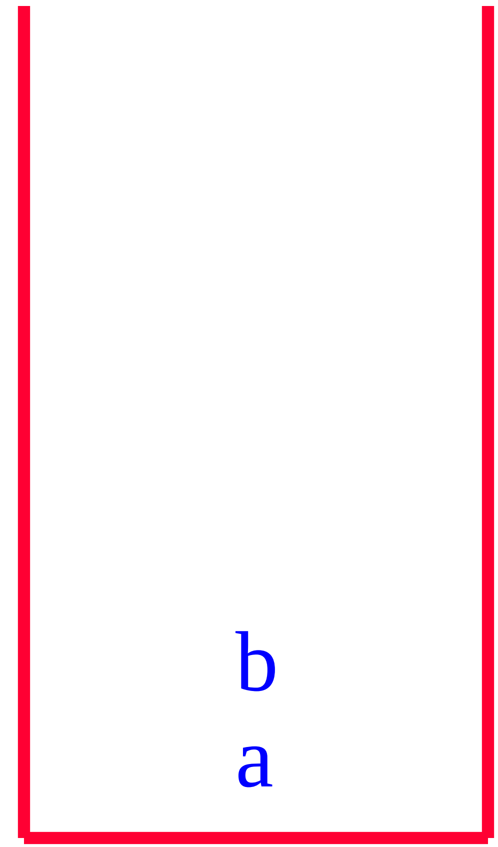
- Replace with new symbols.
  - + a => a @
  - + a + b => a @ b +
  - - a => a ?
  - - a-b => a ? b -

Postfix Evaluation

- Scan postfix expression from left to right pushing operands on to a stack.
- When an operator is encountered, pop as many operands as this operator needs; evaluate the operator; push the result on to the stack.
- This works because, in postfix, operators come immediately after their operands.

Postfix Evaluation

- (a + b) * (c – d) / (e + f)
- a b + c d - * e f + /
- a b + c d - * e f + /
- a b + c d - * e f + /
- a b + c d - * e f + /



stack

Postfix Evaluation

- (a + b) * (c – d) / (e + f)
- a b + c d - * e f + /
- a b + c d - * e f + /
- a b + c d - * e f + /
- a b + c d - * e f + /
- a b + c d - * e f + /
- a b + c d - * e f + /
- a b + c d - * e f + /

d

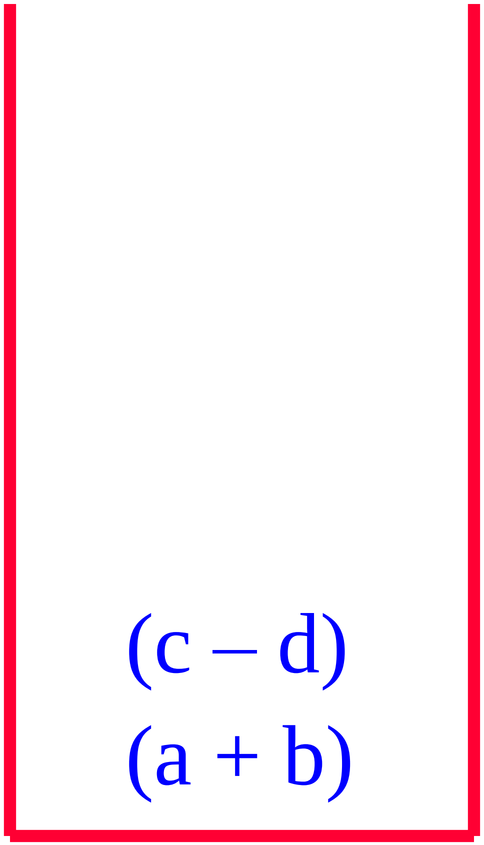
c

(a + b)

stack

Postfix Evaluation

- (a + b) * (c – d) / (e + f)
- a b + c d - * e f + /
- a b + c d - * e f + /



stack

Postfix Evaluation

- (a + b) * (c – d) / (e + f)
- a b + c d - * e f + /
- a b + c d - * e f + /
- a b + c d - * e f + /
- a b + c d - * e f + /
- a b + c d - * e f + /

f

e

(a + b)*(c – d)

stack

Postfix Evaluation

- (a + b) * (c – d) / (e + f)
- a b + c d - * e f + /
- a b + c d - * e f + /
- a b + c d - * e f + /
- a b + c d - * e f + /
- a b + c d - * e f + /
- a b + c d - * e f + /

(e + f)

(a + b)*(c – d)

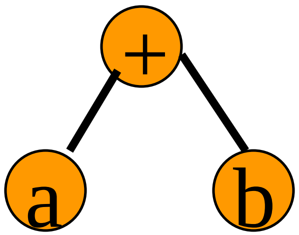
stack

Prefix Form

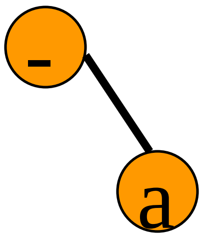
- The prefix form of a variable or constant is the same as its infix form.
  - a, b, 3.25
- The relative order of operands is the same in infix and prefix forms.
- Operators come immediately before the prefix form of their operands.
  - Infix = a + b
  - Postfix = ab+
  - Prefix = +ab

Binary Tree Form

- a + b

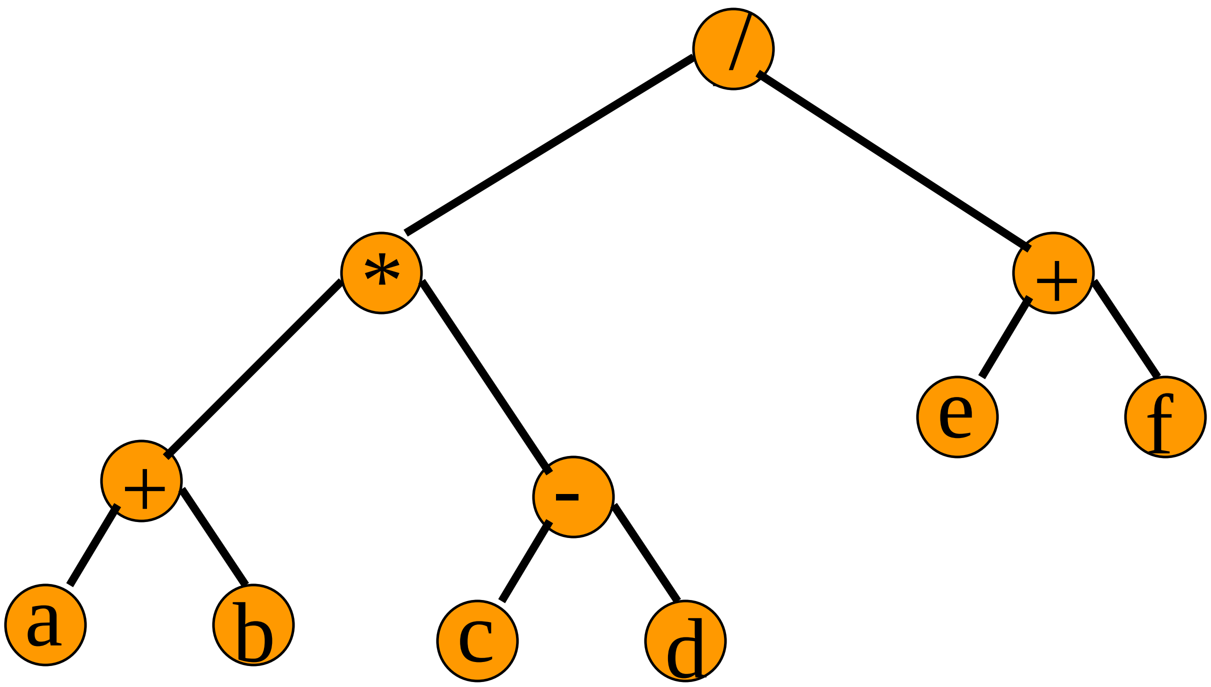


- \- a



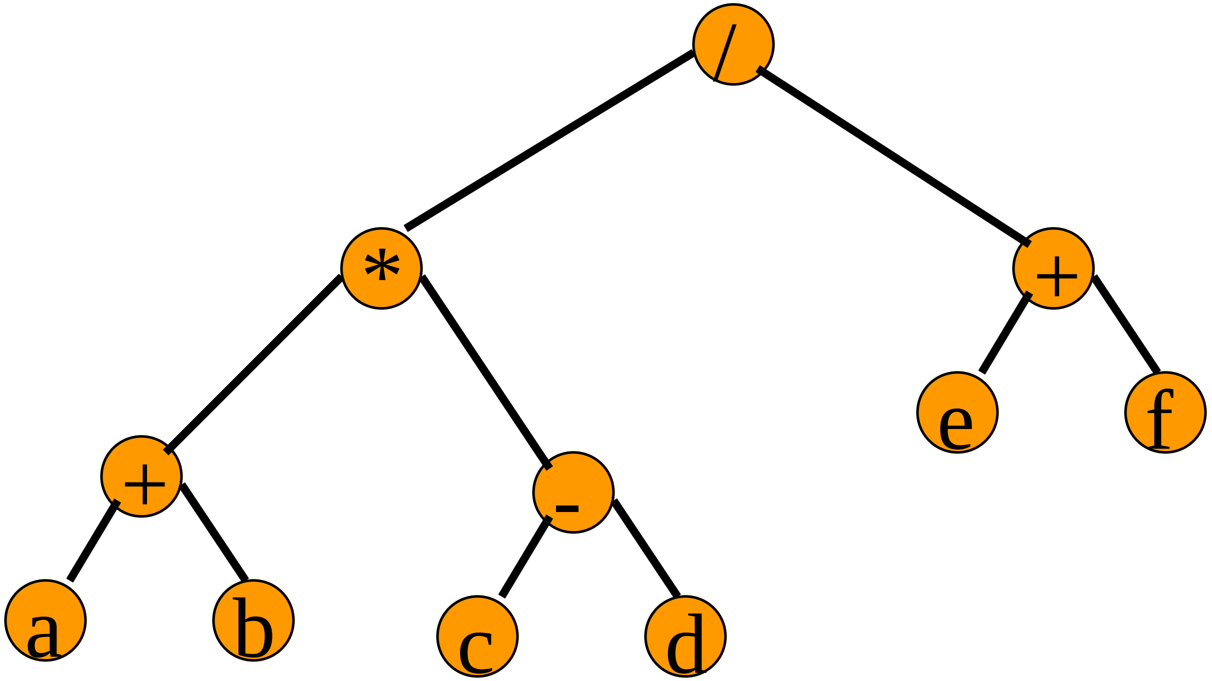
Binary Tree Form

- (a + b) * (c – d) / (e + f)

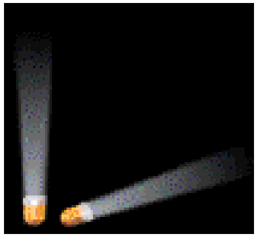


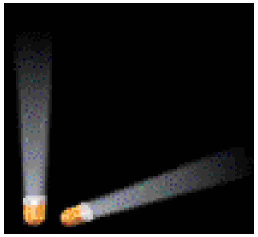
Merits Of Binary Tree Form

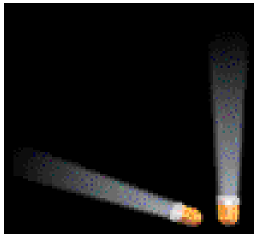
- Left and right operands are easy to visualize.
- Code optimization algorithms work with the binary tree form of an expression.
- Simple recursive evaluation of expression.

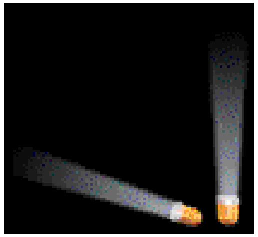


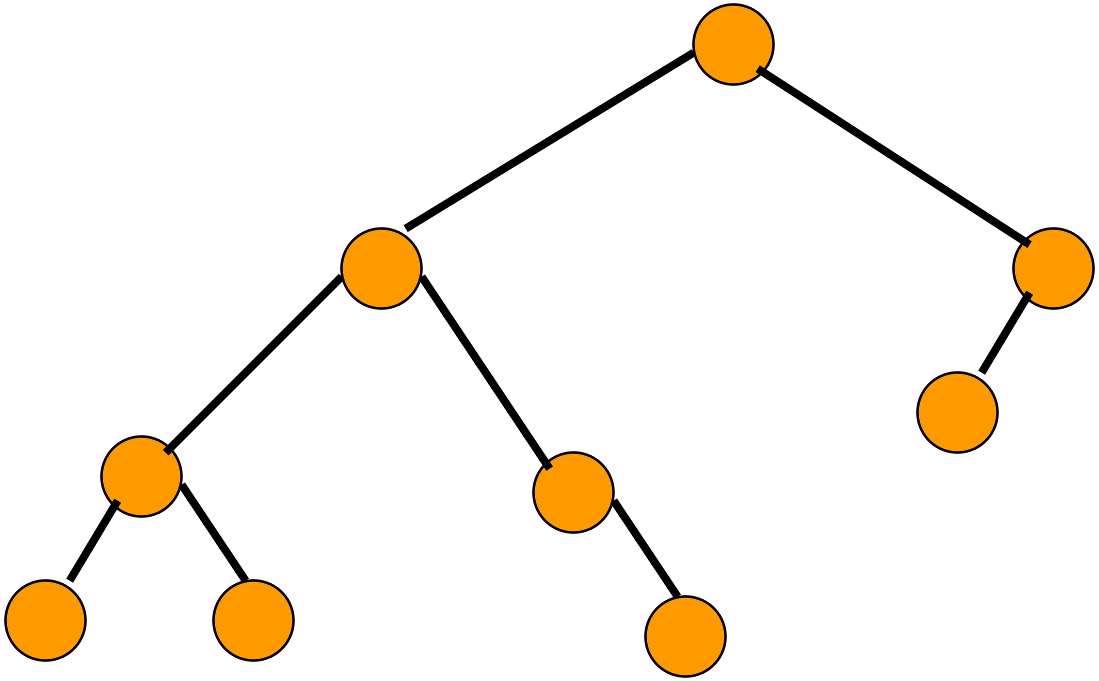
Binary Tree Properties & Representation





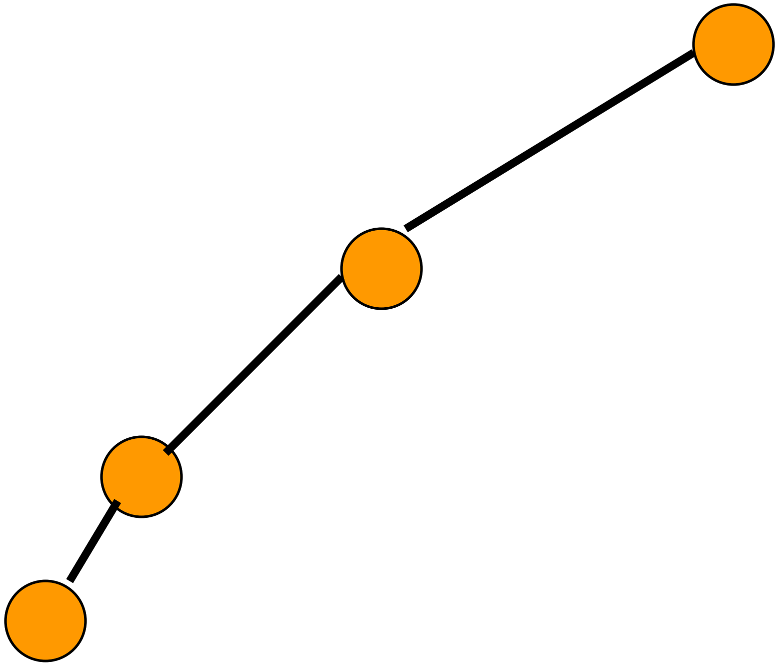






Minimum Number Of Nodes

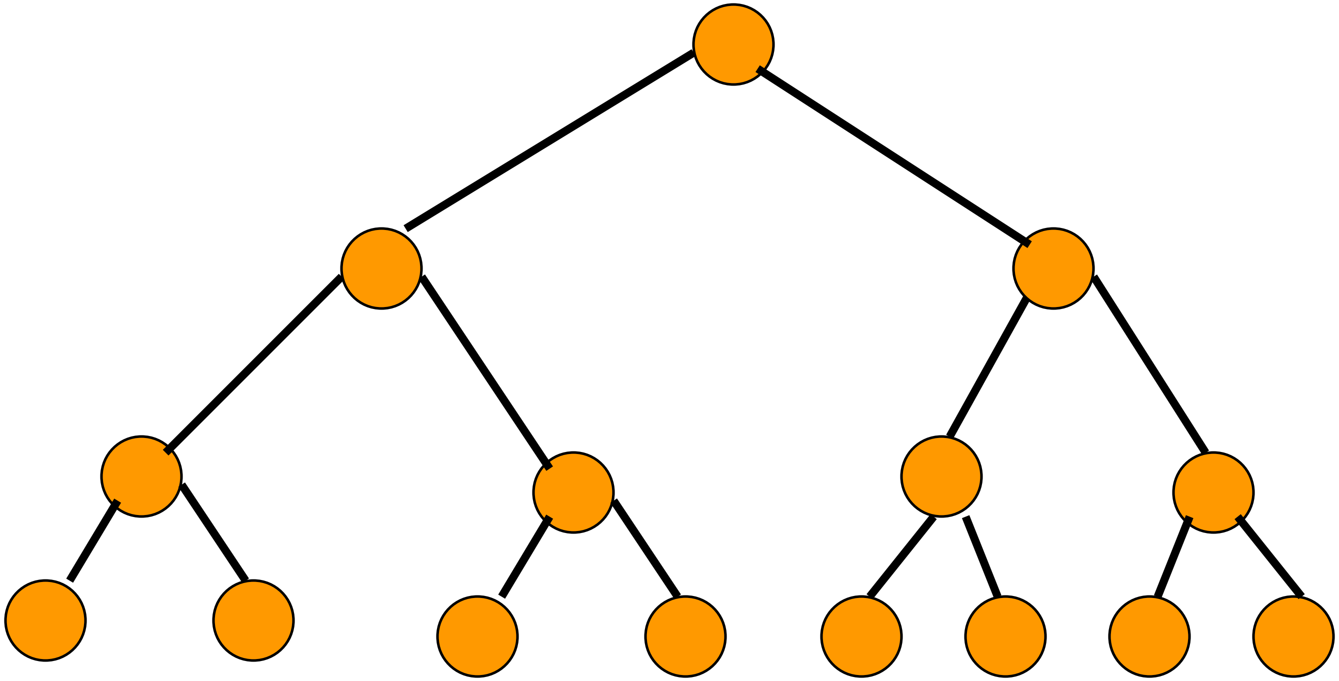
- Minimum number of nodes in a binary tree whose height is h.
- At least one node at each of first h levels.



minimum number of nodes is h

Maximum Number Of Nodes

- All possible nodes at first h levels are present.



Maximum number of nodes

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

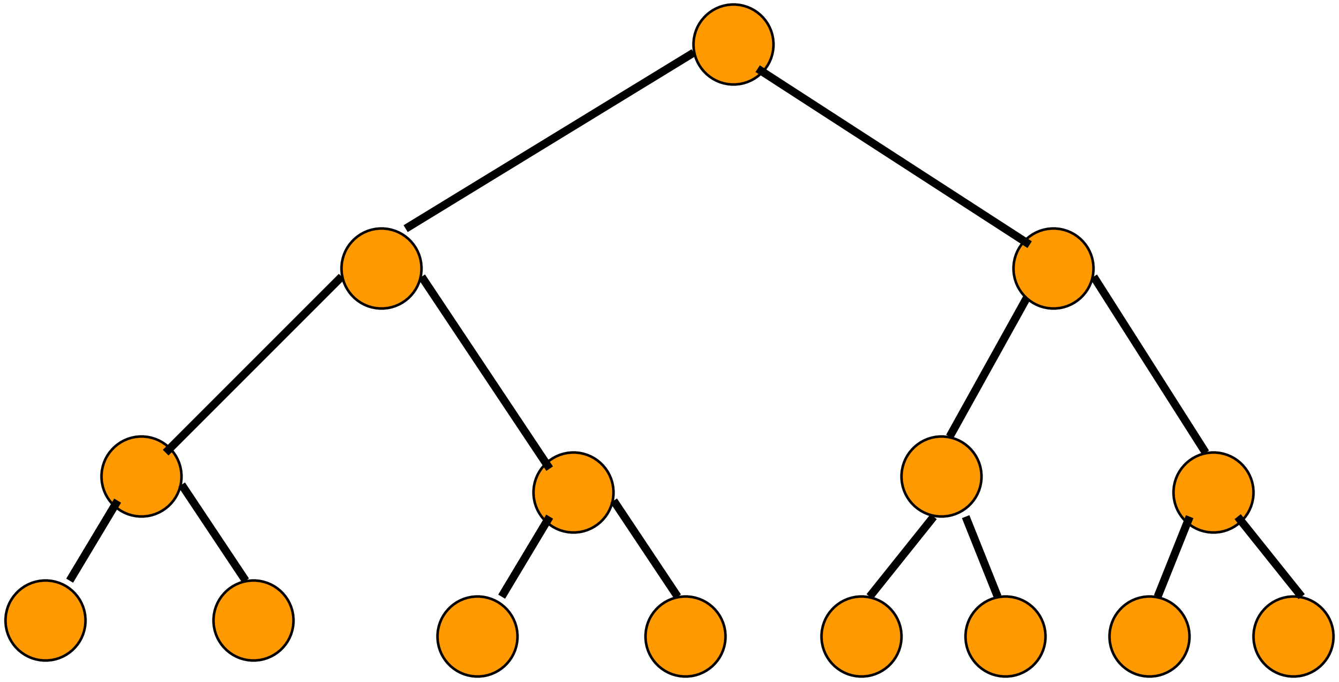
$= 2^h - 1$

Number Of Nodes & Height

- Let n be the number of nodes in a binary tree whose height is h.
- $h <= n <= 2^h - 1$
- $\log_2(n+1) <= h <= n$

Full Binary Tree

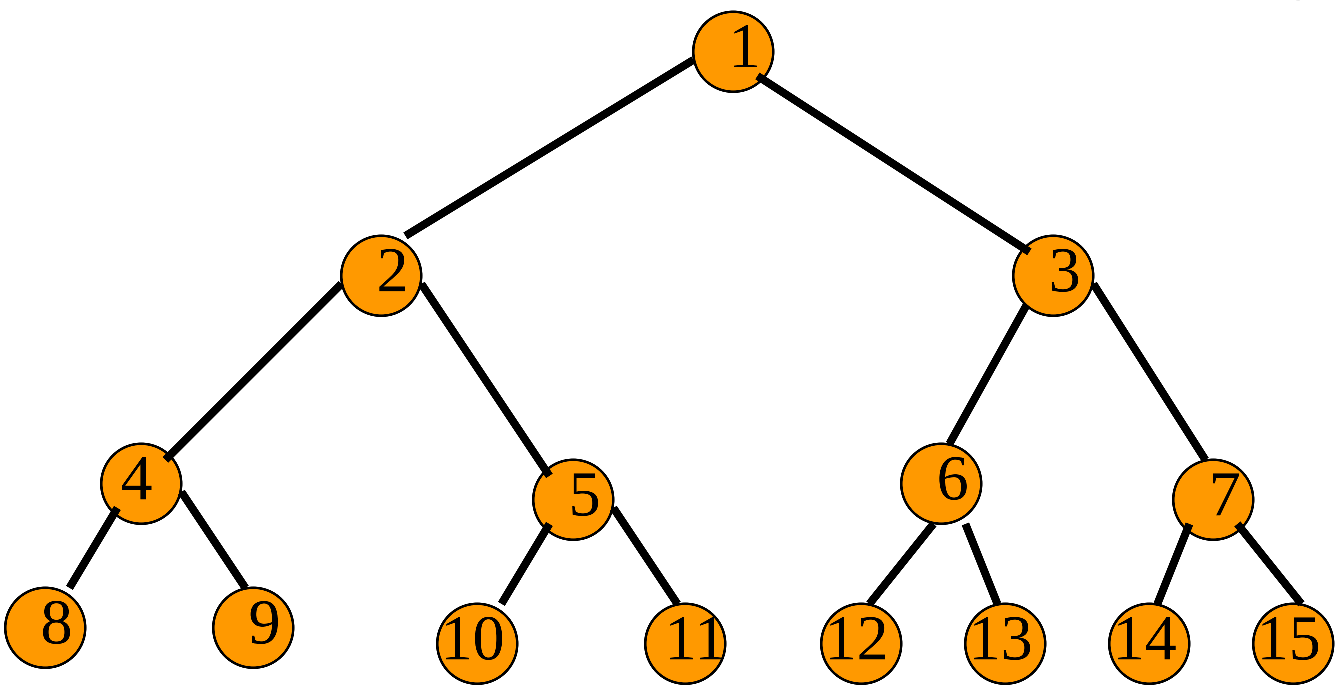
- A full binary tree of a given height h has $2^h - 1$ nodes.



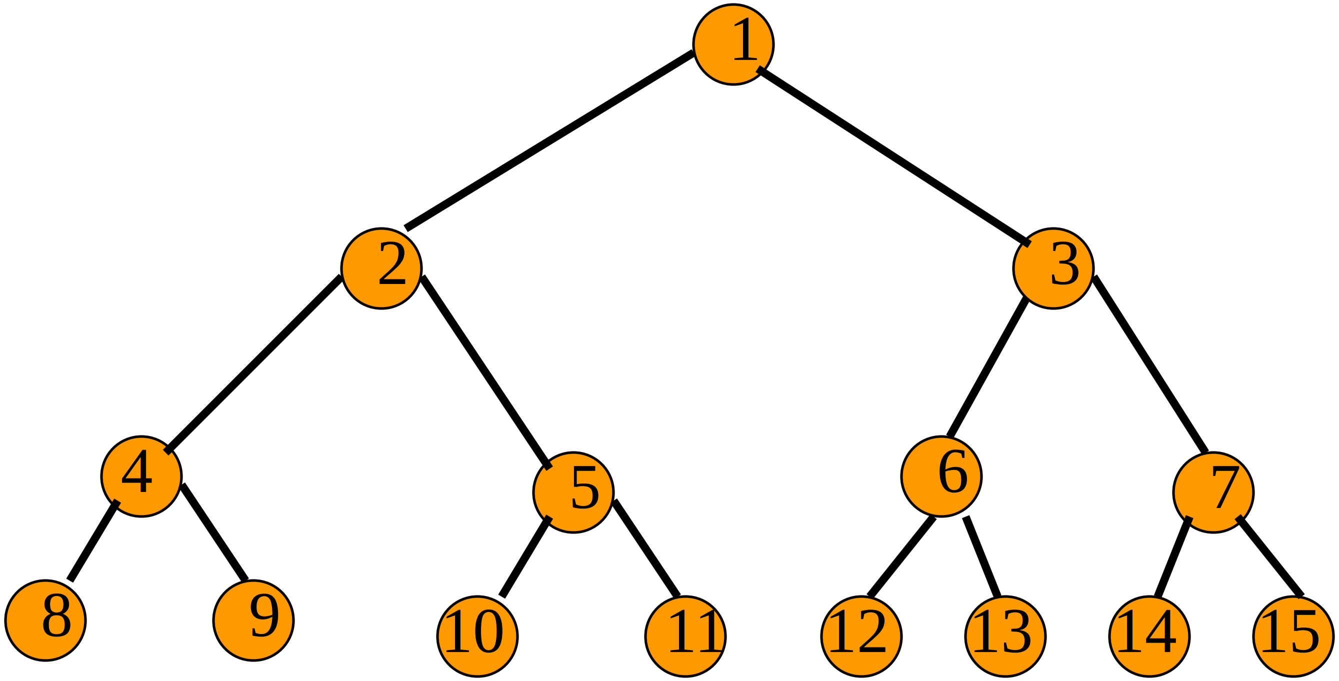
Height 4 full binary tree.

Numbering Nodes In A Full Binary Tree

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

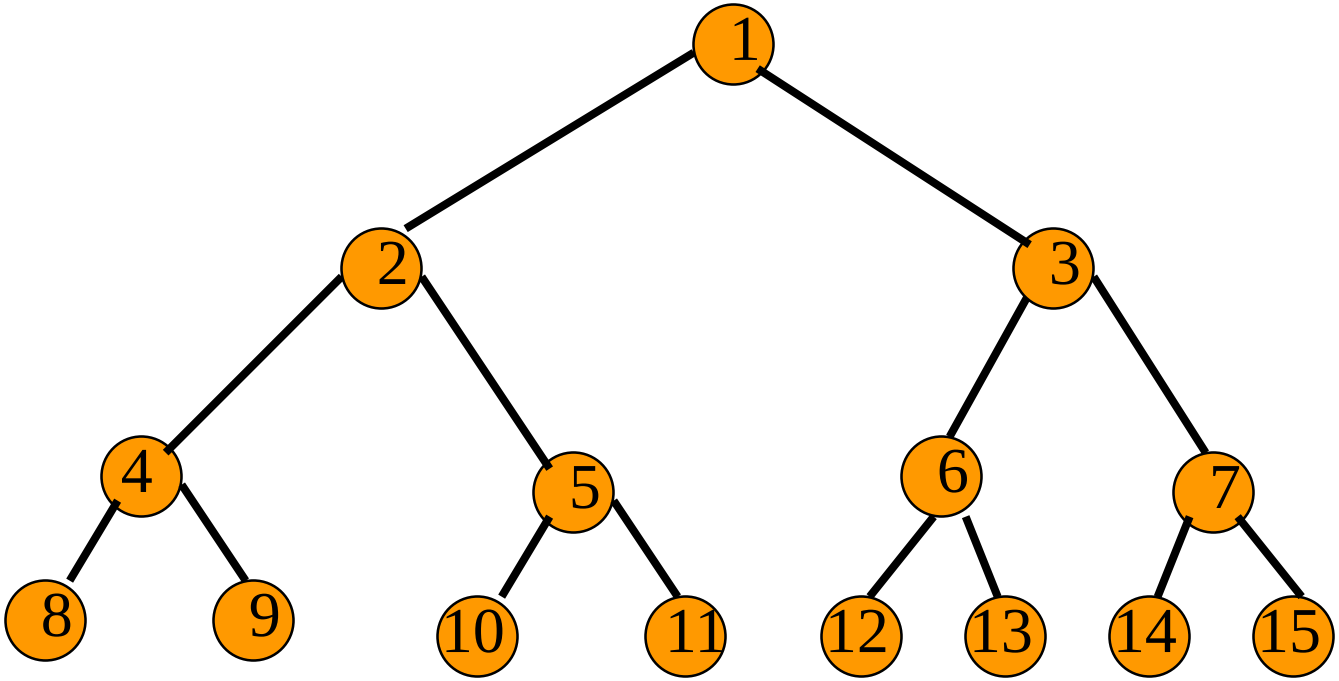


Node Number Properties



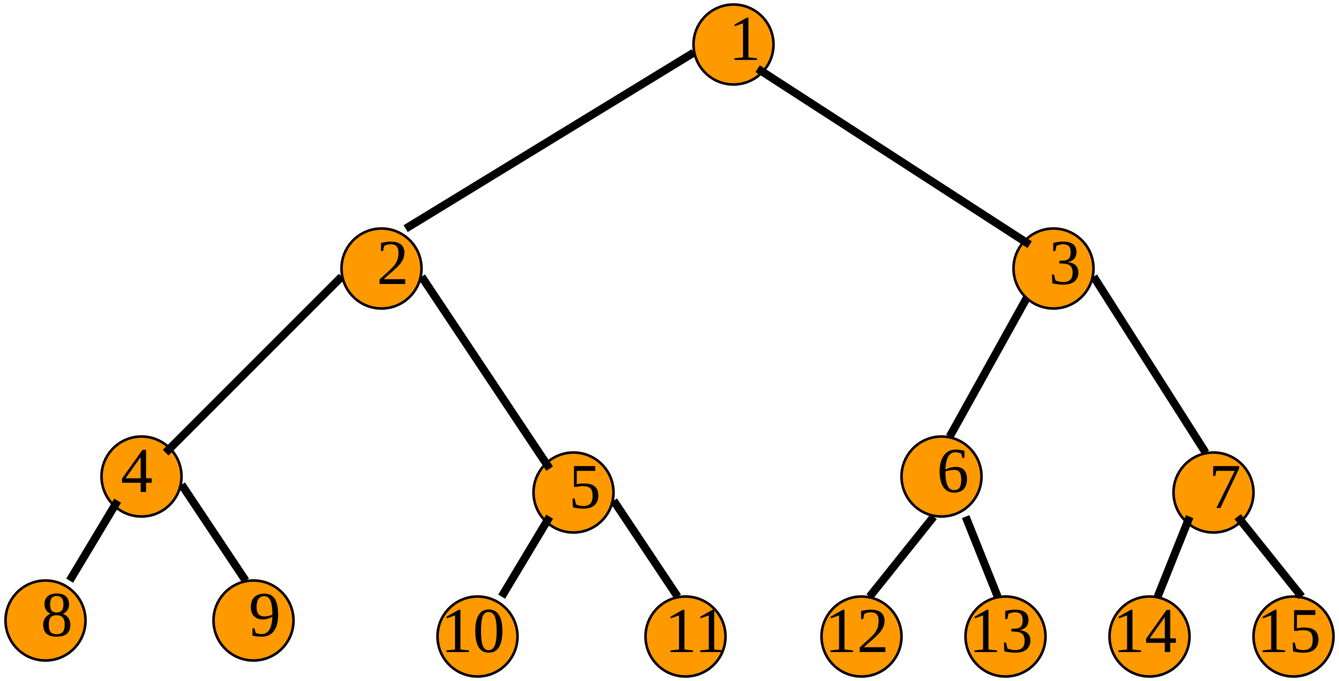
- Parent of node i is node i / 2, unless i = 1.
- Node 1 is the root and has no parent.

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

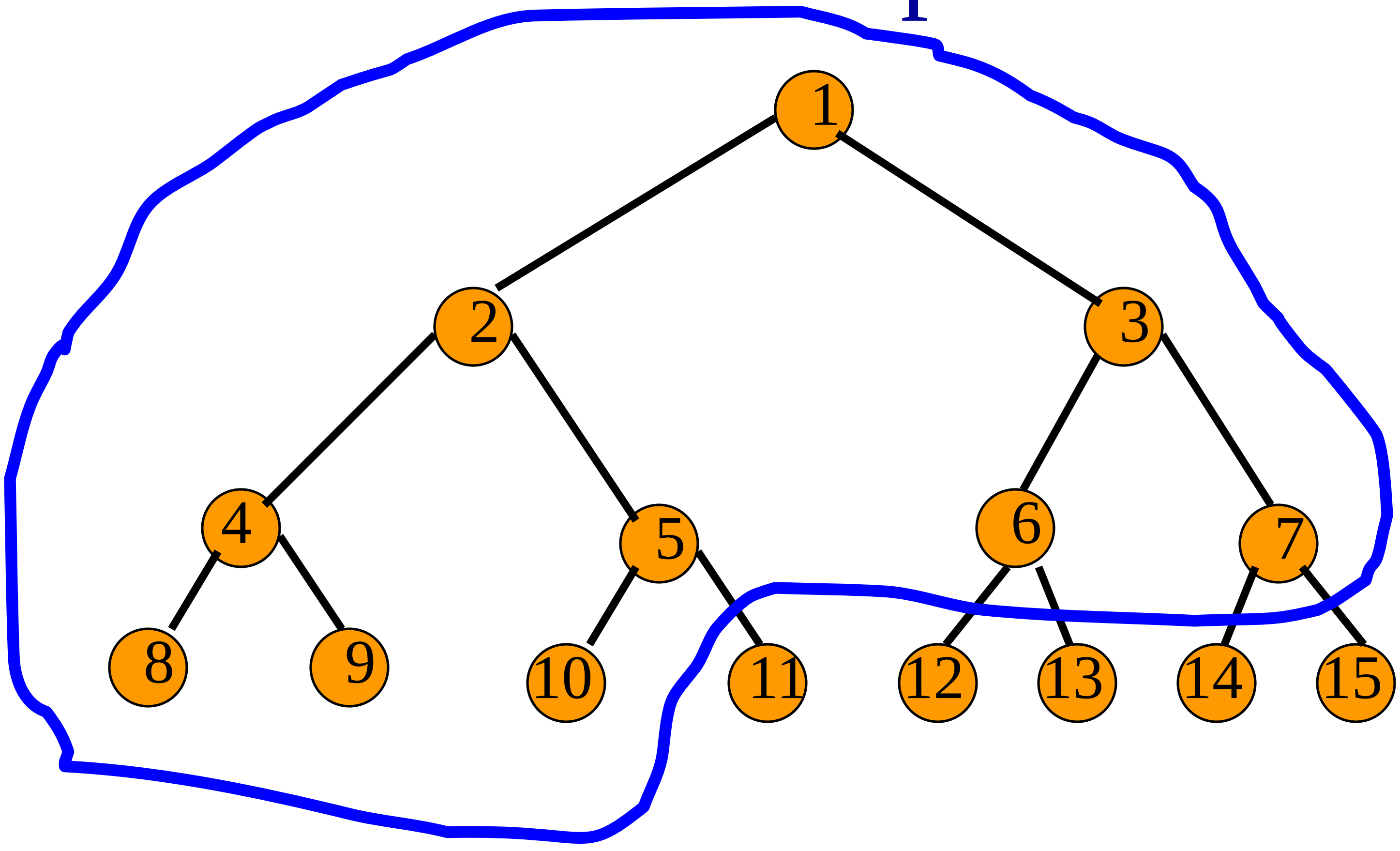


- Right child of node i is node 2i+1, unless 2i+1 > n, where n is the number of nodes.
- If 2i+1 > n, node i has no right child.

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.

Example



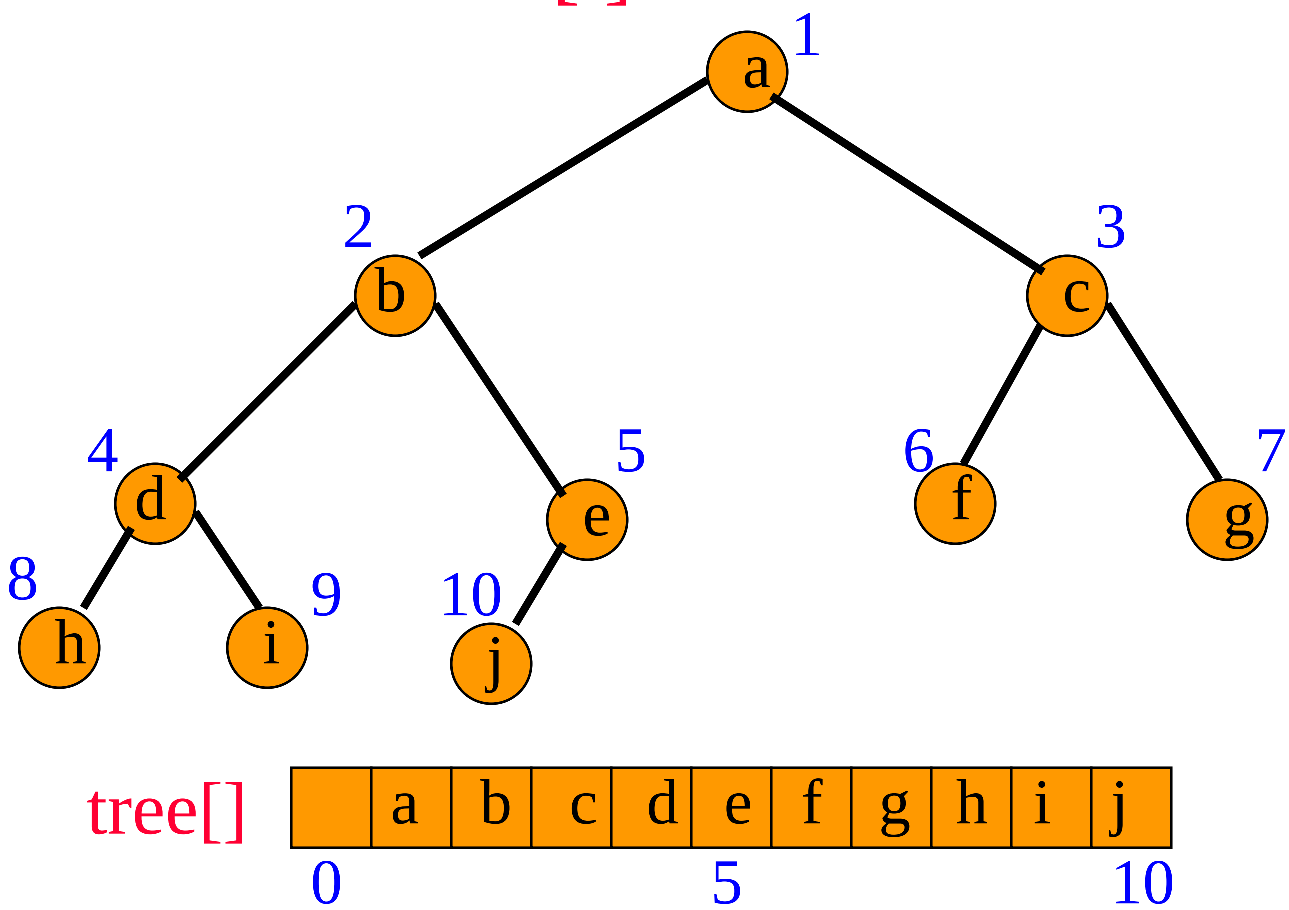
- Complete binary tree with 10 nodes.

Binary Tree Representation

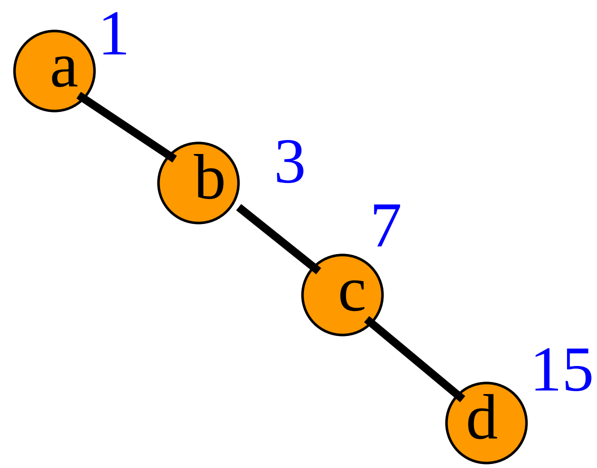
- Array representation.
- Linked representation.

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].



Right-Skewed Binary Tree



tree[]		a	-	b	-	-	-	c	-	-	-	-	-	-	-	d
	0					5					10					15

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

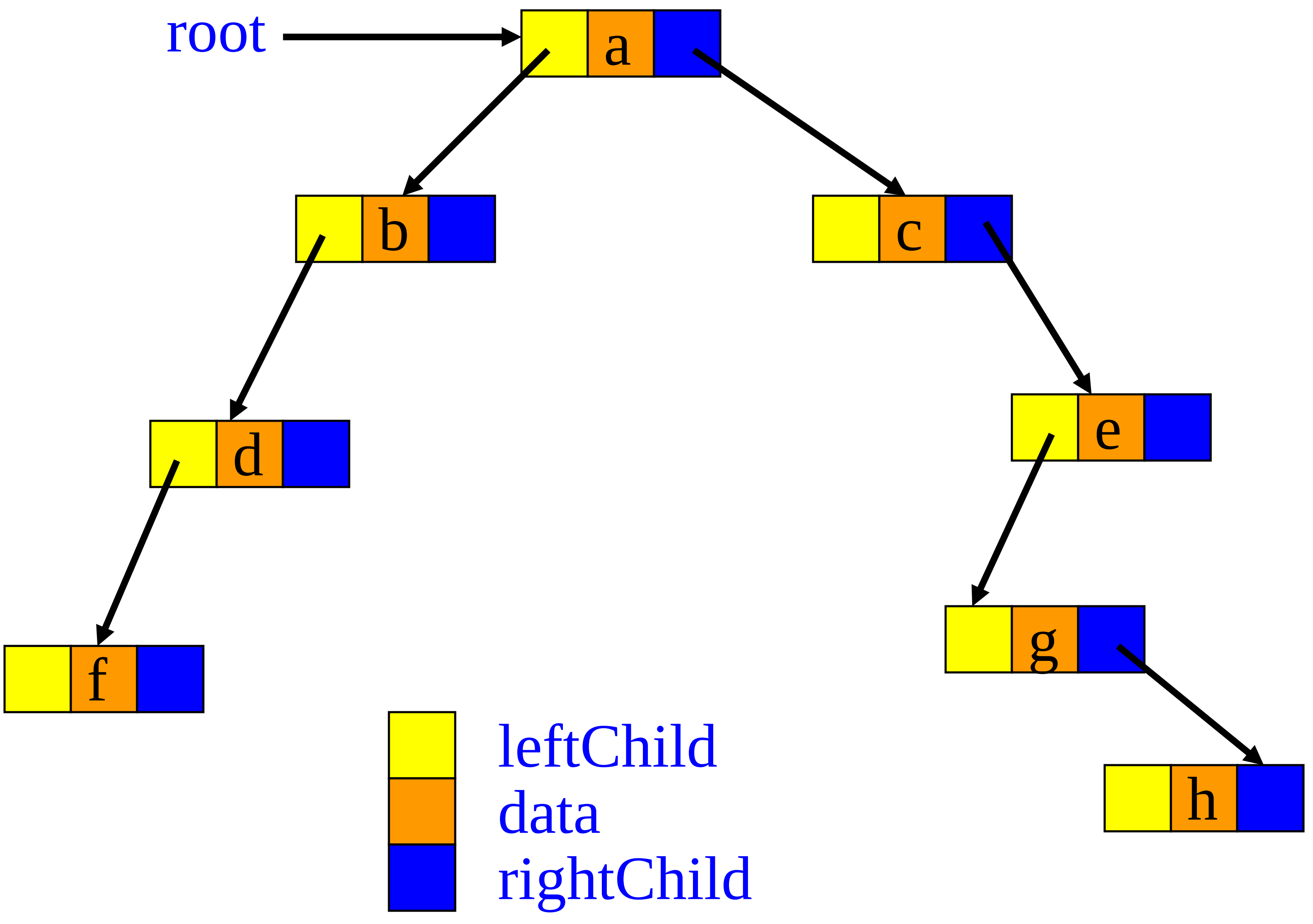
Linked Representation

- Each binary tree node is represented as an object whose data type is TreeNode.
- The space required by an n node binary tree is n * (space required by one node).

The Struct binaryTreeNode

```
template <class T>
class TreeNode
{
   T data;
   TreeNode<T> *leftChild,
               *rightChild;
   TreeNode()
      {leftChild = rightChild = NULL;}
   // other constructors come here
};
```

Linked Representation Example



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.

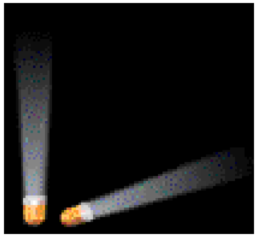
Binary Tree Traversal

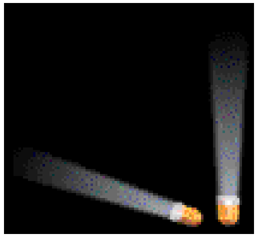
- Many binary tree operations are done by performing a traversal of the binary tree.
- In a traversal, *each element* of the binary tree is *visited exactly once*.
- During the visit of an element, all action (make a clone, display, evaluate the operator, etc.) with respect to this element is taken.

Binary Tree Traversal Methods

- Preorder
- Inorder
- Postorder
- Level order

Binary Tree Traversal Methods





- In a traversal of a binary tree, 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.

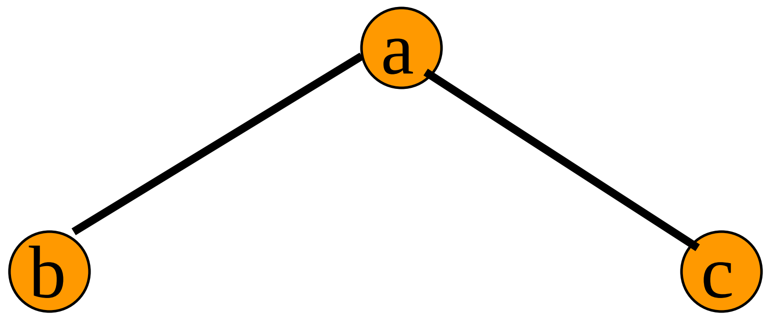
Binary Tree Traversal Methods

- Preorder
- Inorder
- Postorder
- Level order

Preorder Traversal

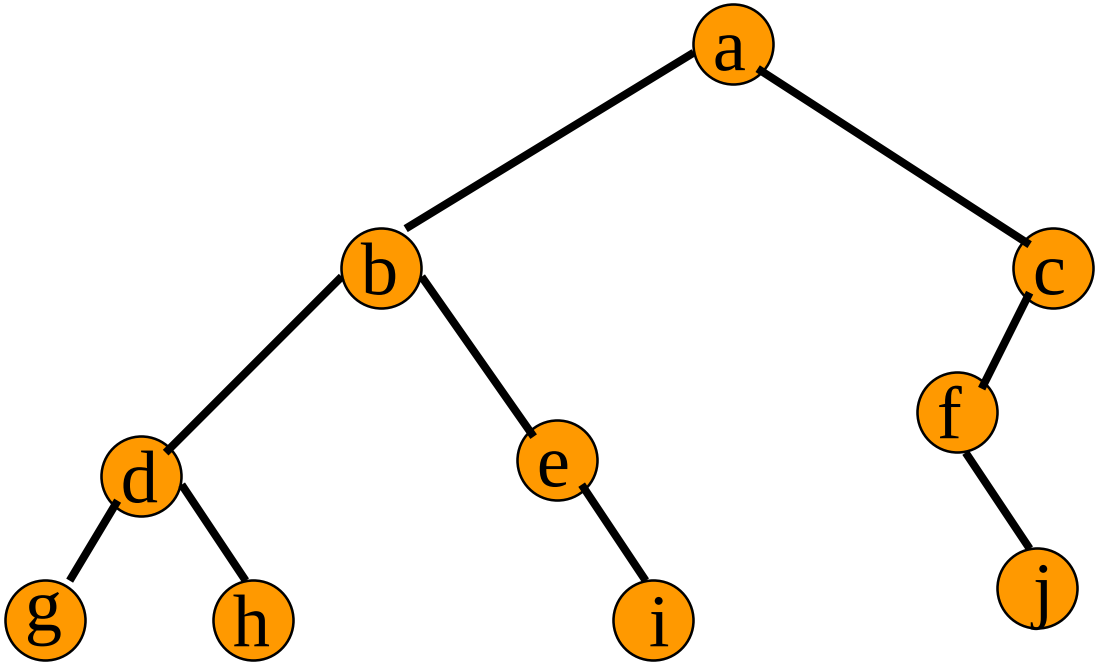
```
template <class T>
void PreOrder(TreeNode<T> *t)
{
    if (t != NULL)
    {
        Visit(t);
        PreOrder(t->leftChild);
        PreOrder(t->rightChild);
    }
}
```

Preorder Example (Visit = print)



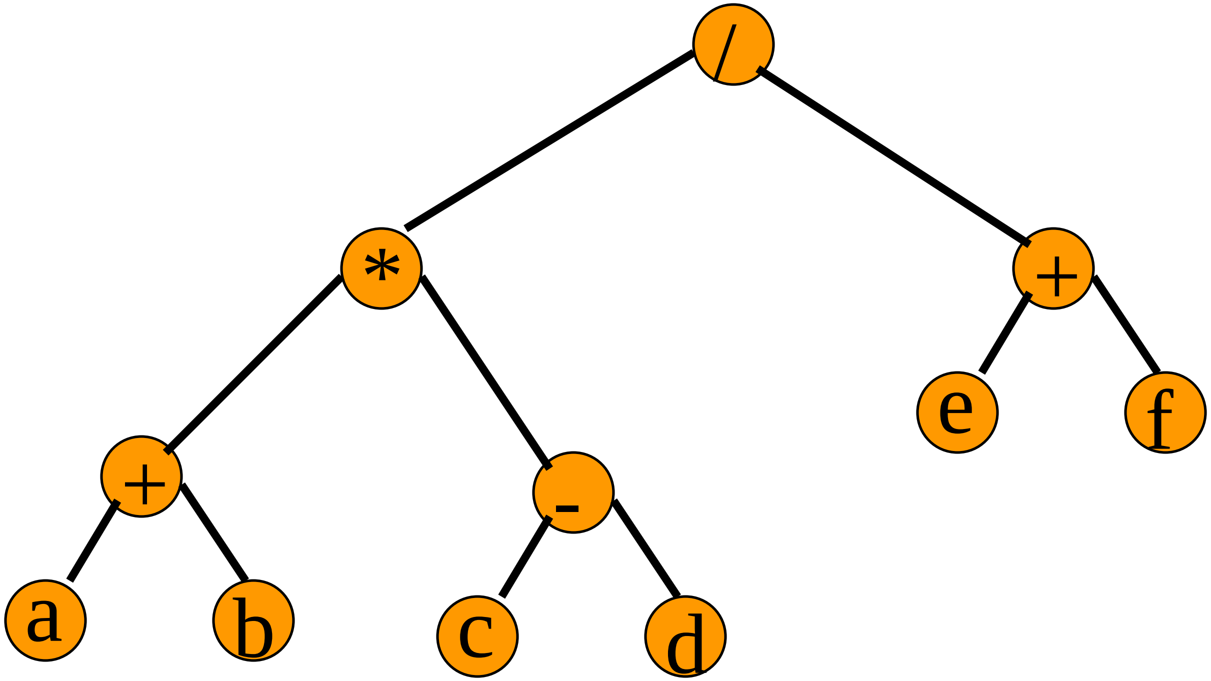
a b c

Preorder Example (Visit = print)



a b d g h e i c f j

Preorder Of Expression Tree



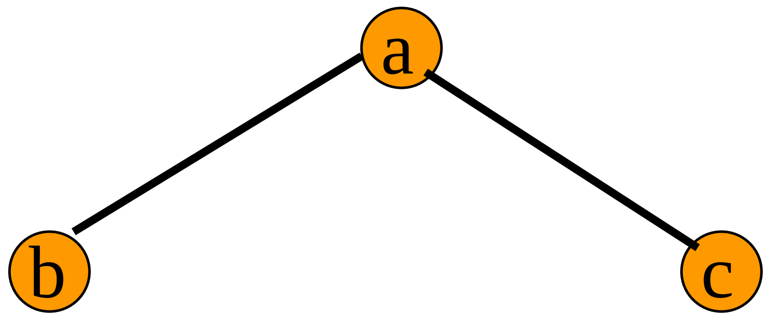
/ * + a b - c d + e f

Gives prefix form of expression!

Inorder Traversal

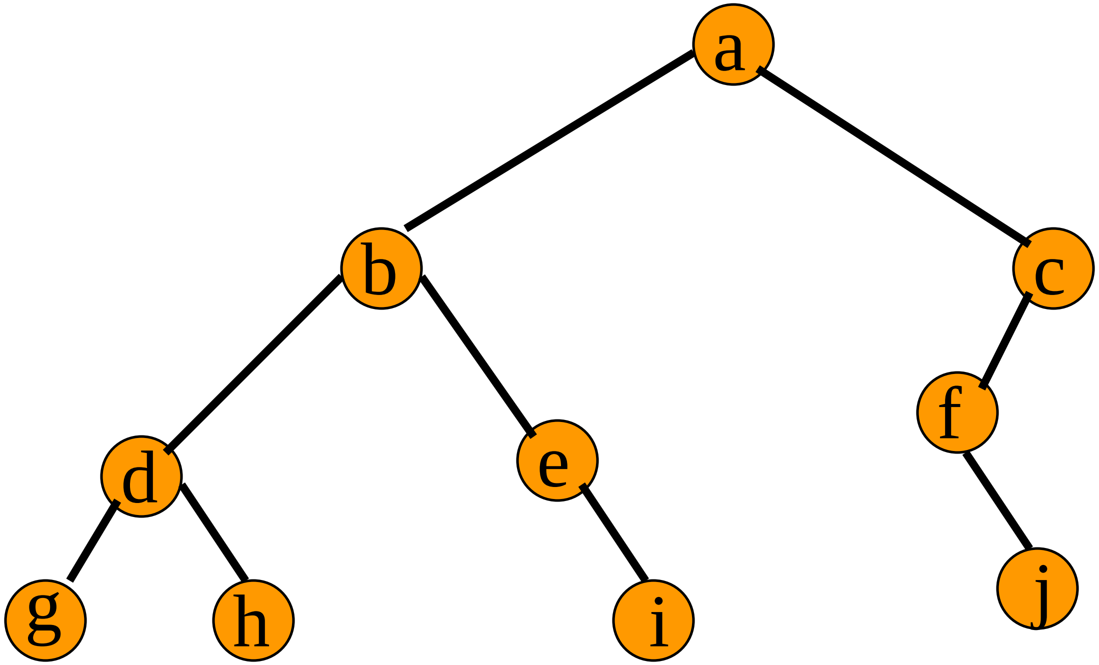
```
template <class T>
void InOrder(TreeNode<T> *t)
{
   if (t != NULL)
   {
      InOrder(t->leftChild);
      Visit(t);
      InOrder(t->rightChild);
   }
}
```

Inorder Example (Visit = print)



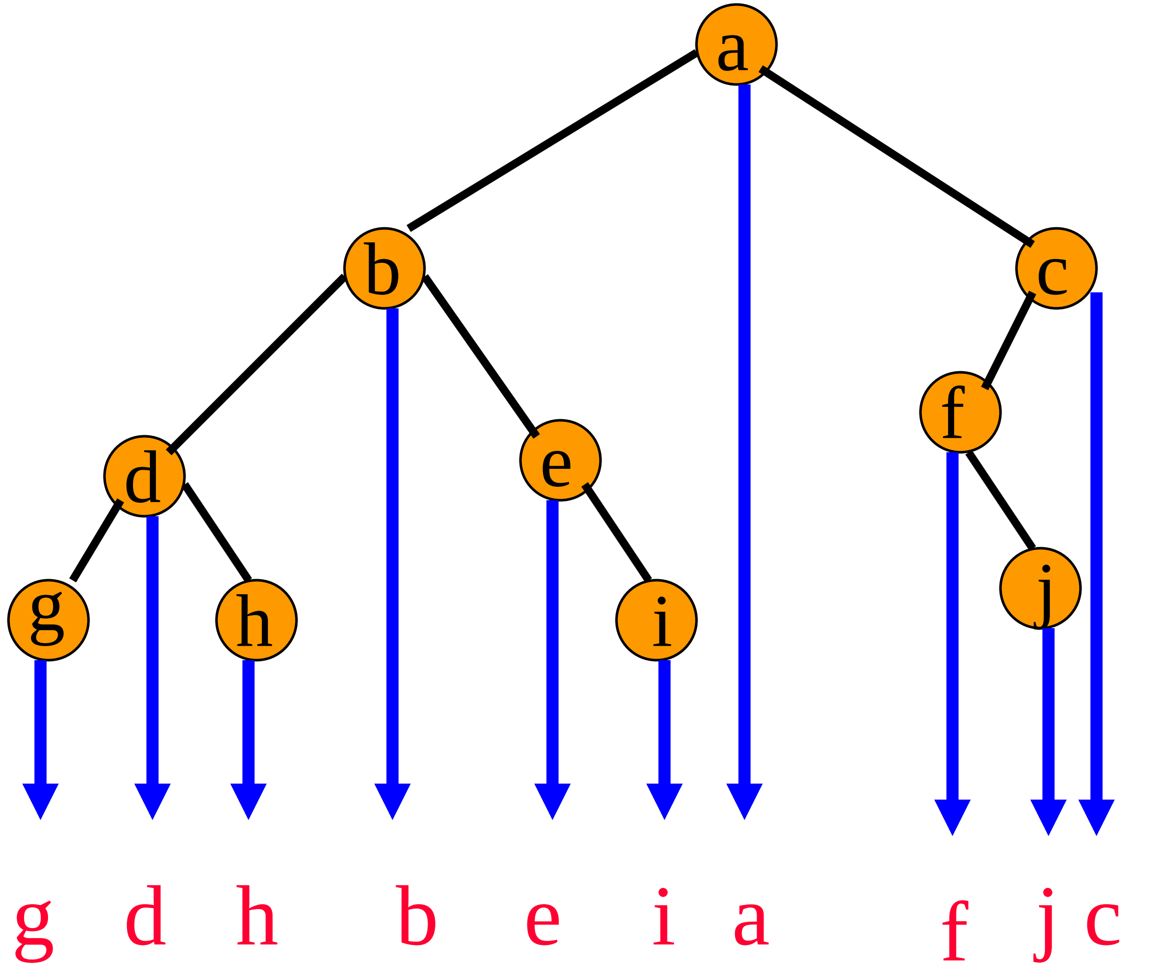
b a c

Inorder Example (Visit = print)

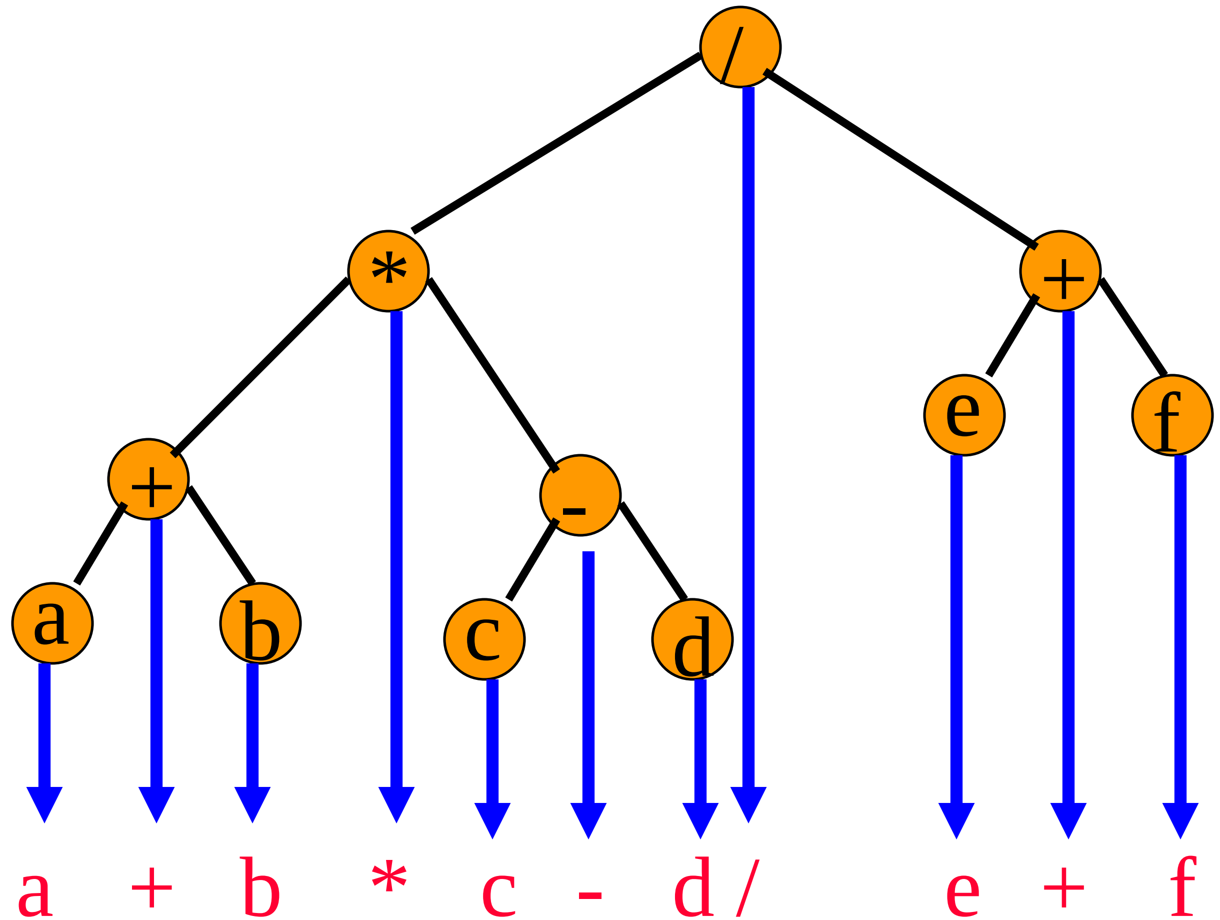


g d h b e i a f j c

Inorder By Projection (Squishing)



Inorder Of Expression Tree

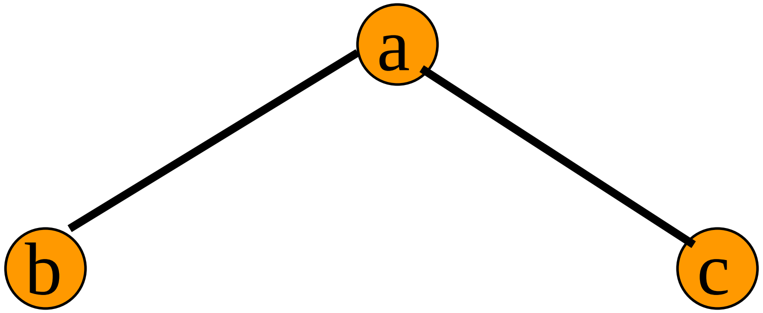


Gives infix form of expression (sans parentheses)!

Postorder Traversal

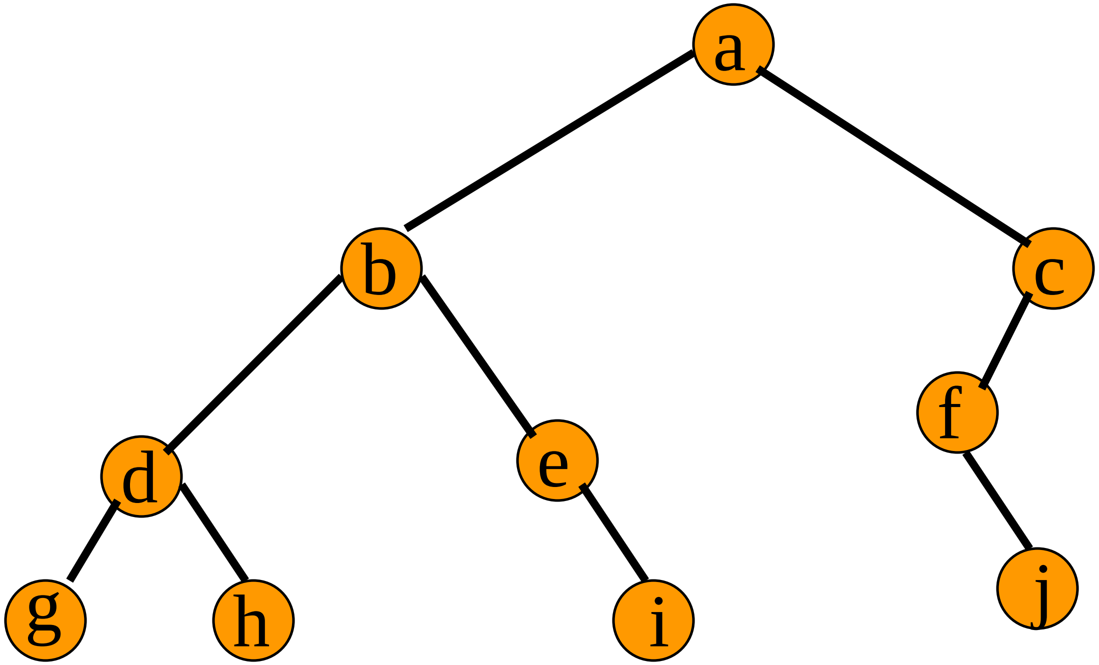
```
template <class T>
void PostOrder(TreeNode<T> *t)
{
    if (t != NULL)
    {
        PostOrder(t->leftChild);
        PostOrder(t->rightChild);
        Visit(t);
    }
}
```

Postorder Example (Visit = print)



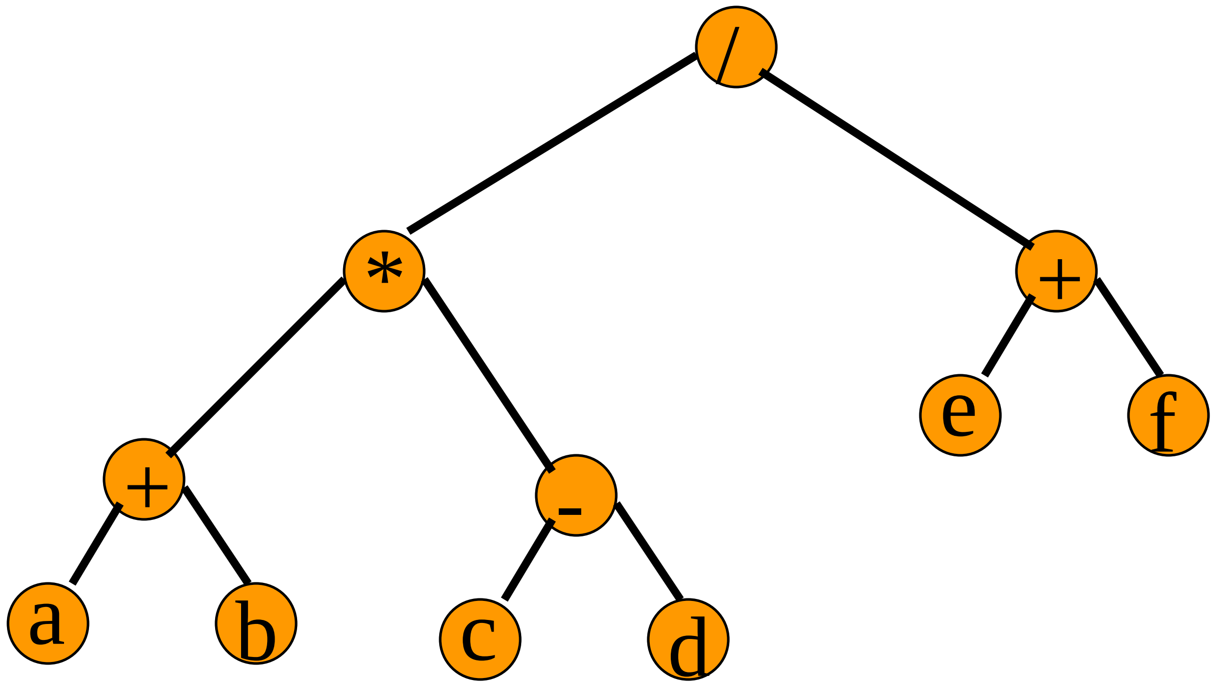
b c a

Postorder Example (Visit = print)



g h d i e b j f c a

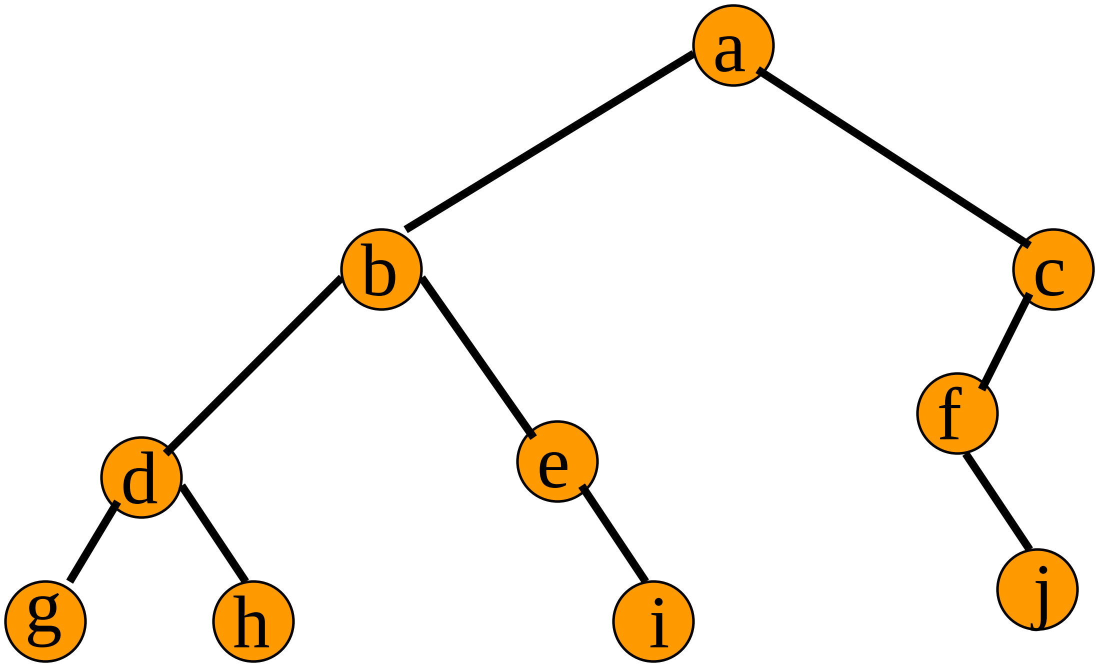
Postorder Of Expression Tree



a b + c d - * e f + /

Gives postfix form of expression!

Traversal Applications



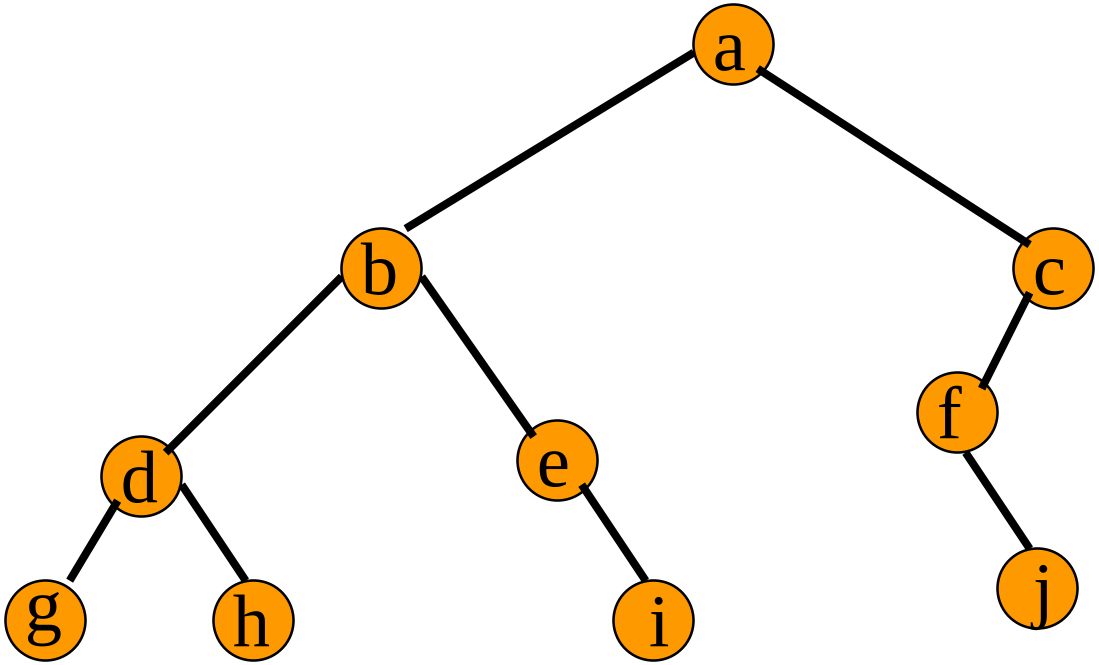
- Make a clone.
- Determine height.
- Determine number of nodes.

Level Order

Let t be the tree root.

```
while (t != NULL)
{
    visit t and put its children on a FIFO queue;
    if FIFO queue is empty, set t = NULL;
    otherwise, pop a node from the FIFO queue
   and call it t;
}
```

Level-Order Example (Visit = print)



a b c d e f g h i j

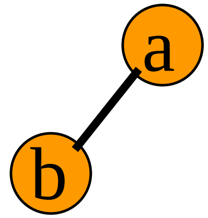
Binary Tree Construction

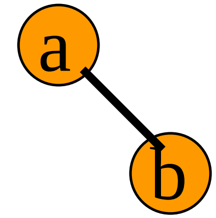
- Suppose that the elements in a binary tree are distinct.
- Can you construct the binary tree from which a given traversal sequence came?
- When a traversal sequence has more than one element, the binary tree is not uniquely defined.
- Therefore, the tree from which the sequence was obtained cannot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

Some Examples

preorder

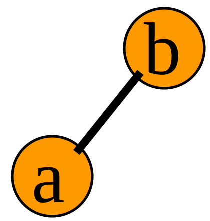
= ab

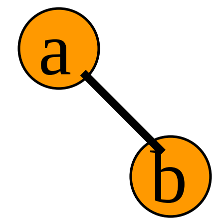




inorder

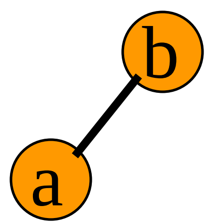
= ab

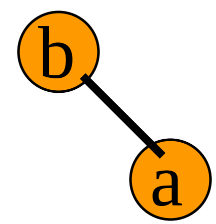




postorder

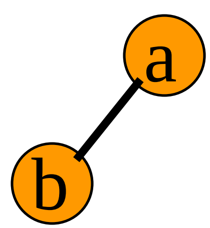
= ab

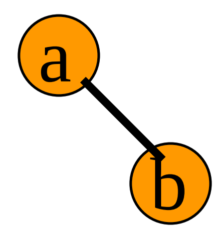




level order

= ab





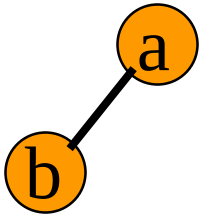
Binary Tree Construction

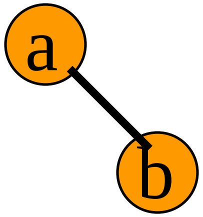
- Can you construct the binary tree, given two traversal sequences?
- Depends on which two sequences are given.

Preorder And Postorder

preorder = ab

postorder = ba





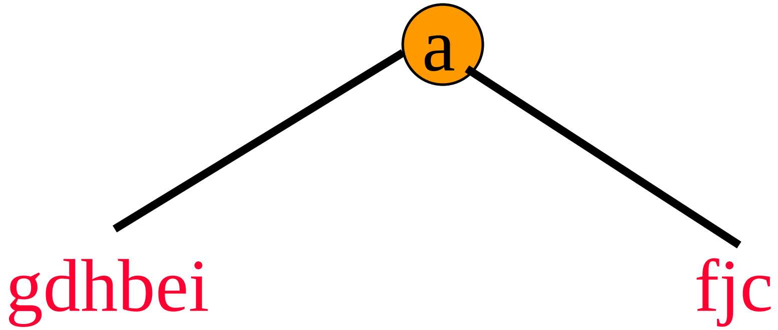
- Preorder and postorder do not uniquely define a binary tree.

- Nor do preorder and level order (same example).

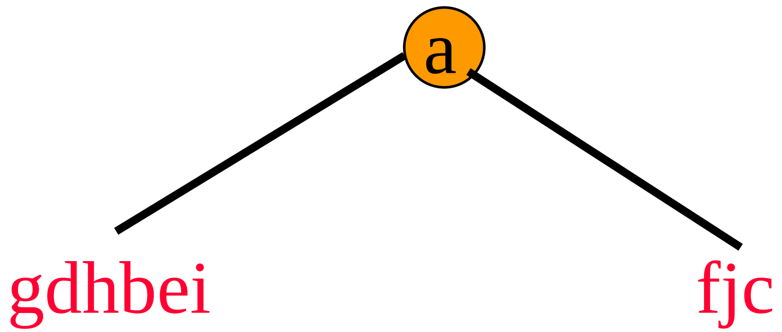
- Nor do postorder and level order (same example).

Inorder And Preorder

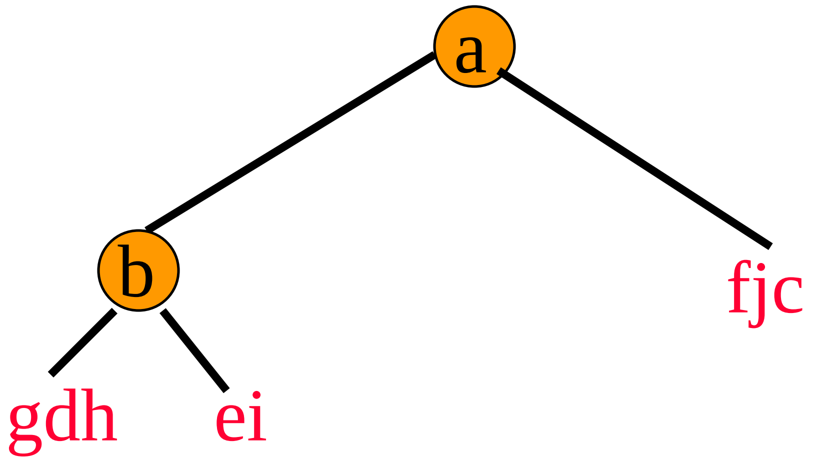
- inorder = g d h b e i a f j c
- preorder = a b d g h e i c f j
- Scan the preorder left to right using the inorder to separate left and right subtrees.
- a is the root of the tree; gdhbei are in the left subtree; fjc are in the right subtree.



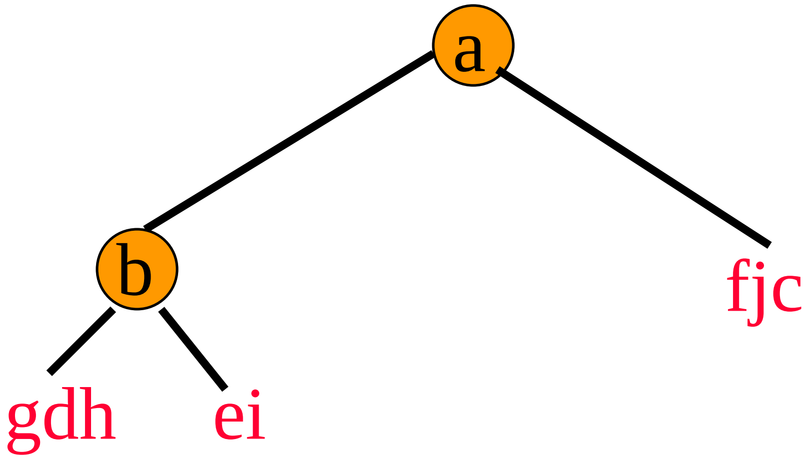
Inorder And Preorder



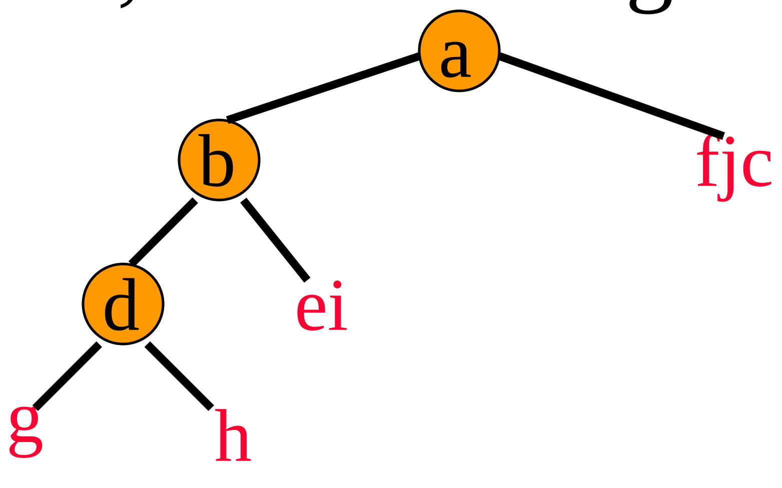
- preorder = a b d g h e i c f j
- b is the next root; gdh are in the left subtree; ei are in the right subtree.



Inorder And Preorder



- preorder = a b d g h e i c f j
- 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.