Chapter 10 Binary Trees
---二叉树部分

信息科学与技术学院　　　黄方军

data_structures@163.com

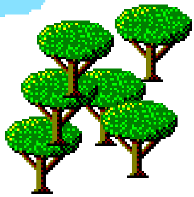
东校区实验中心B502

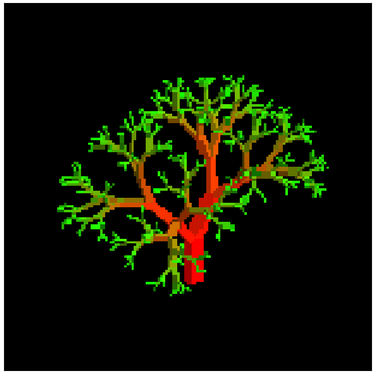
10.1 Binary Trees

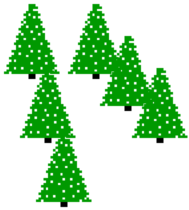
- **General trees (or simply trees)**
- **Binary trees**

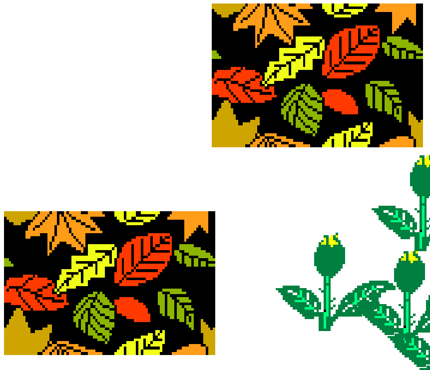
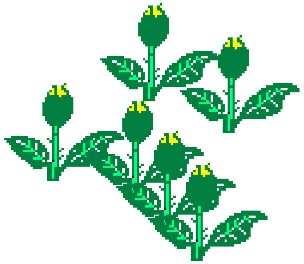
Trees

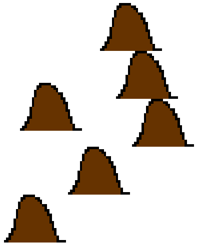


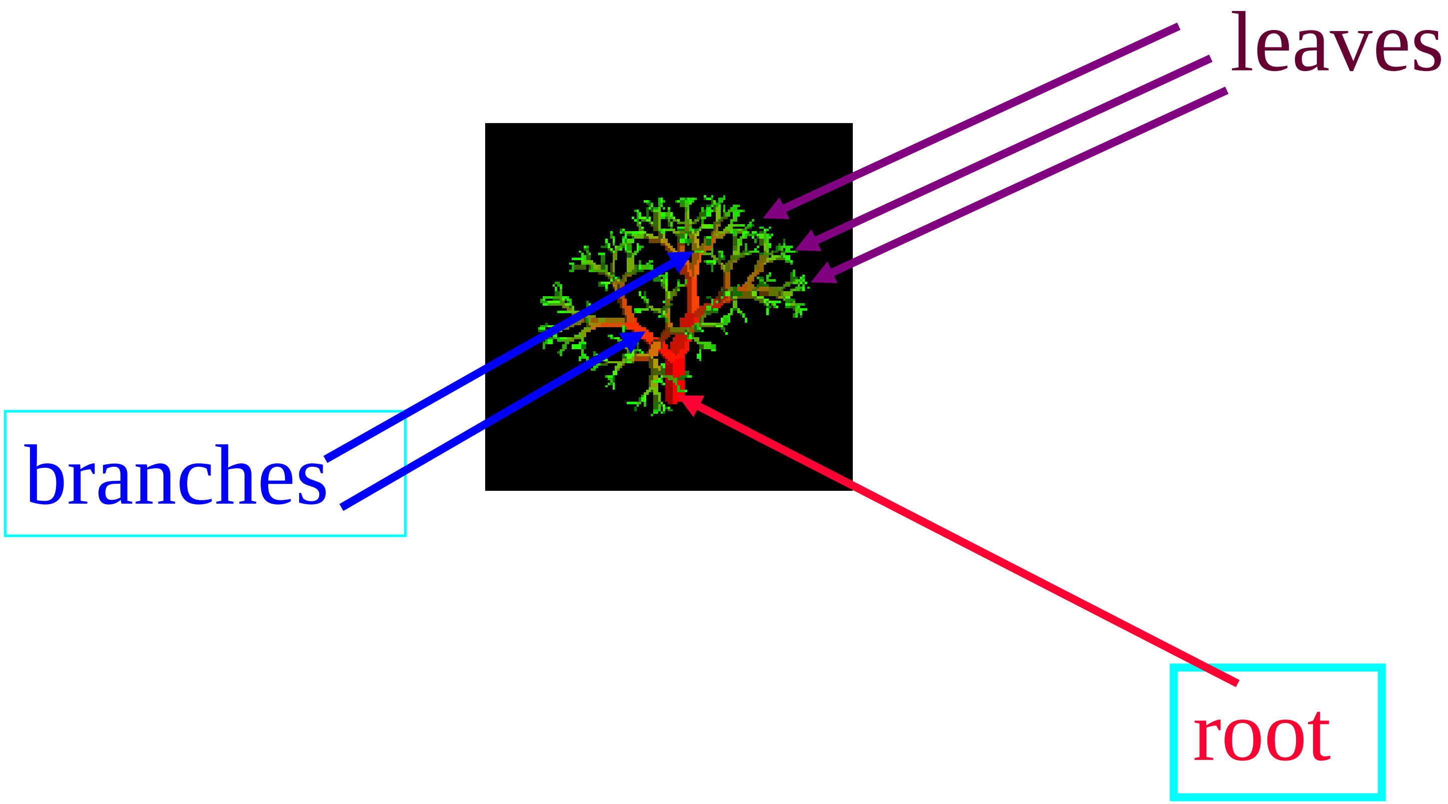





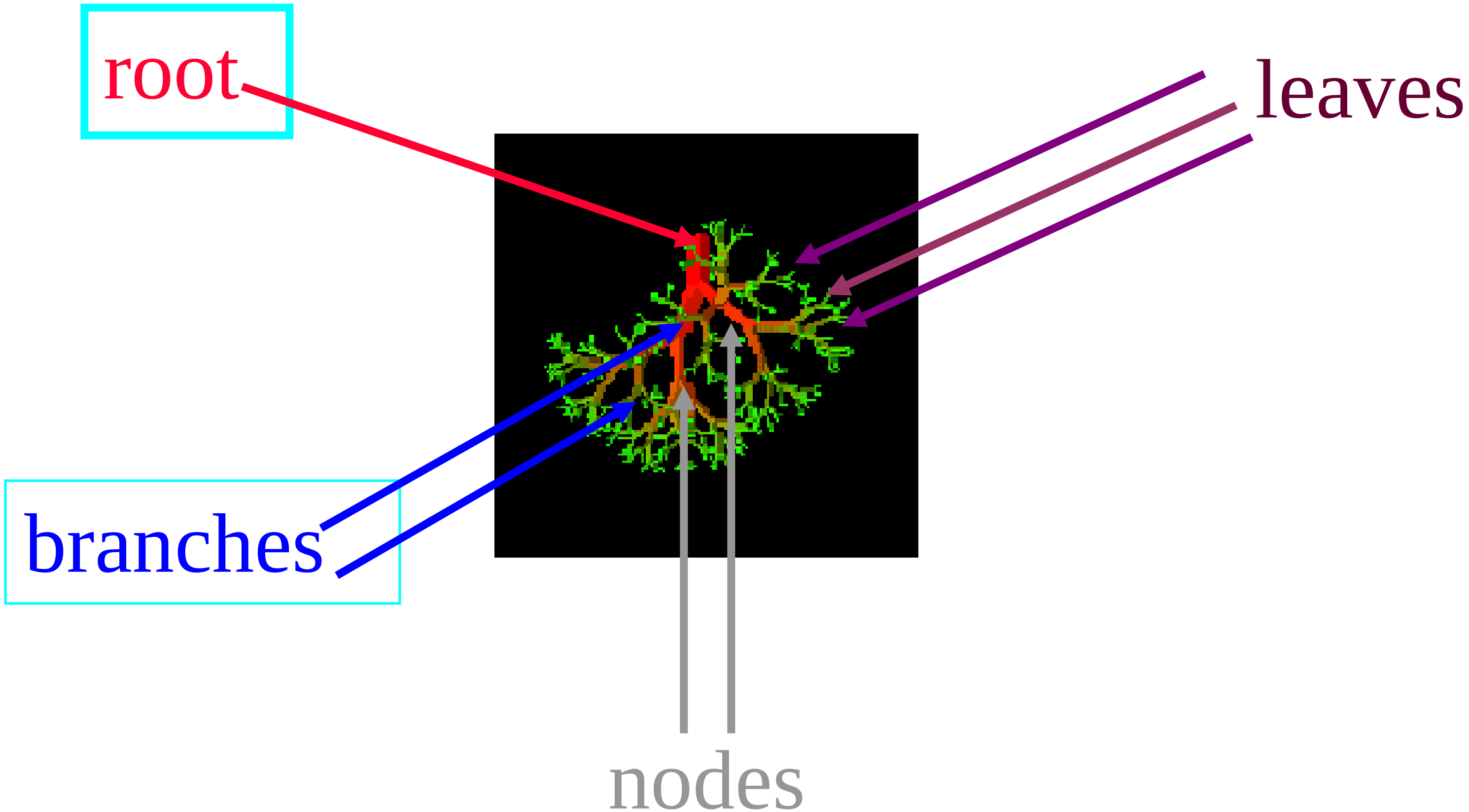






Nature Lover’s View Of A Tree



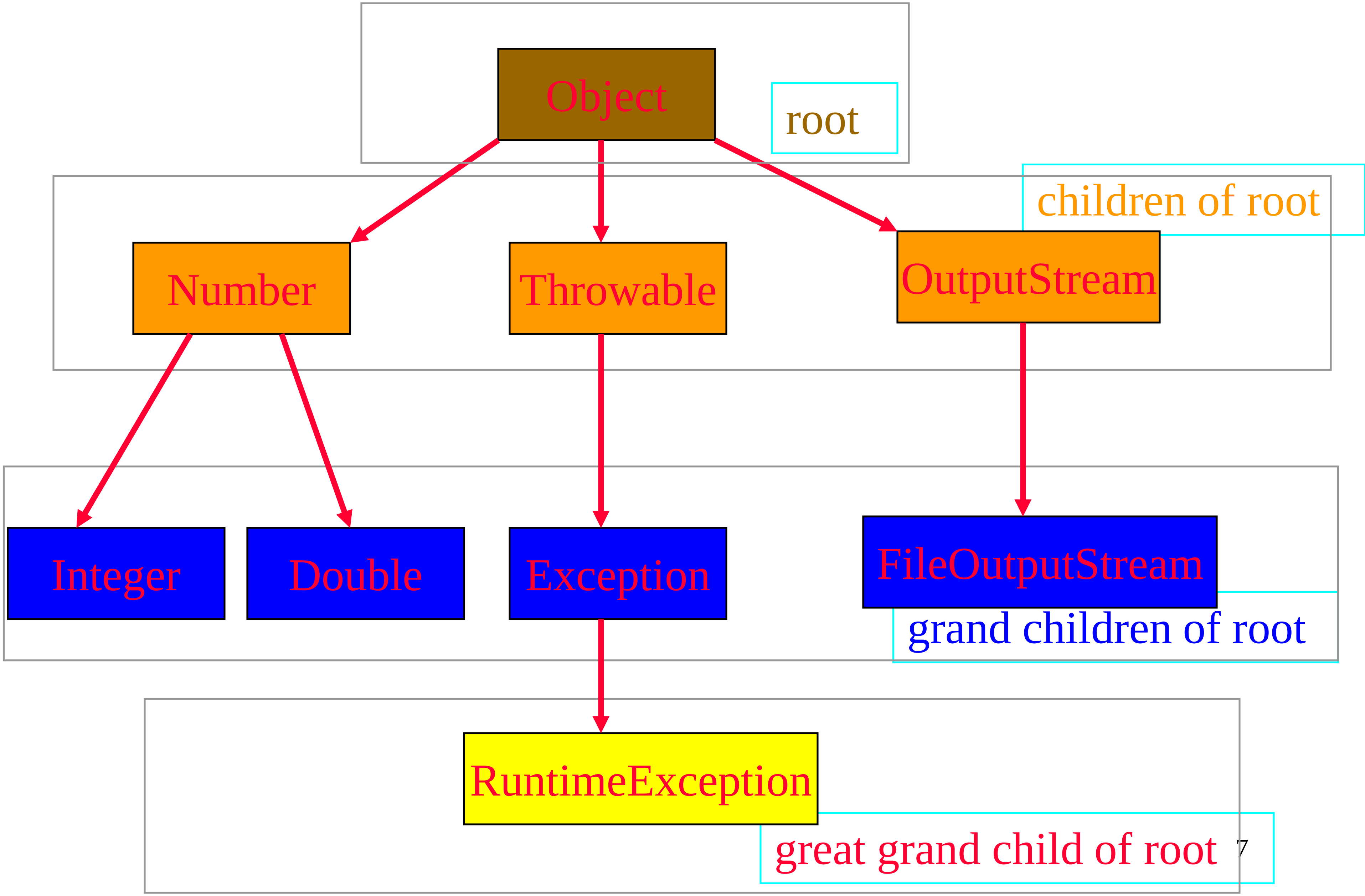
Computer Scientist's View



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: Java's Classes

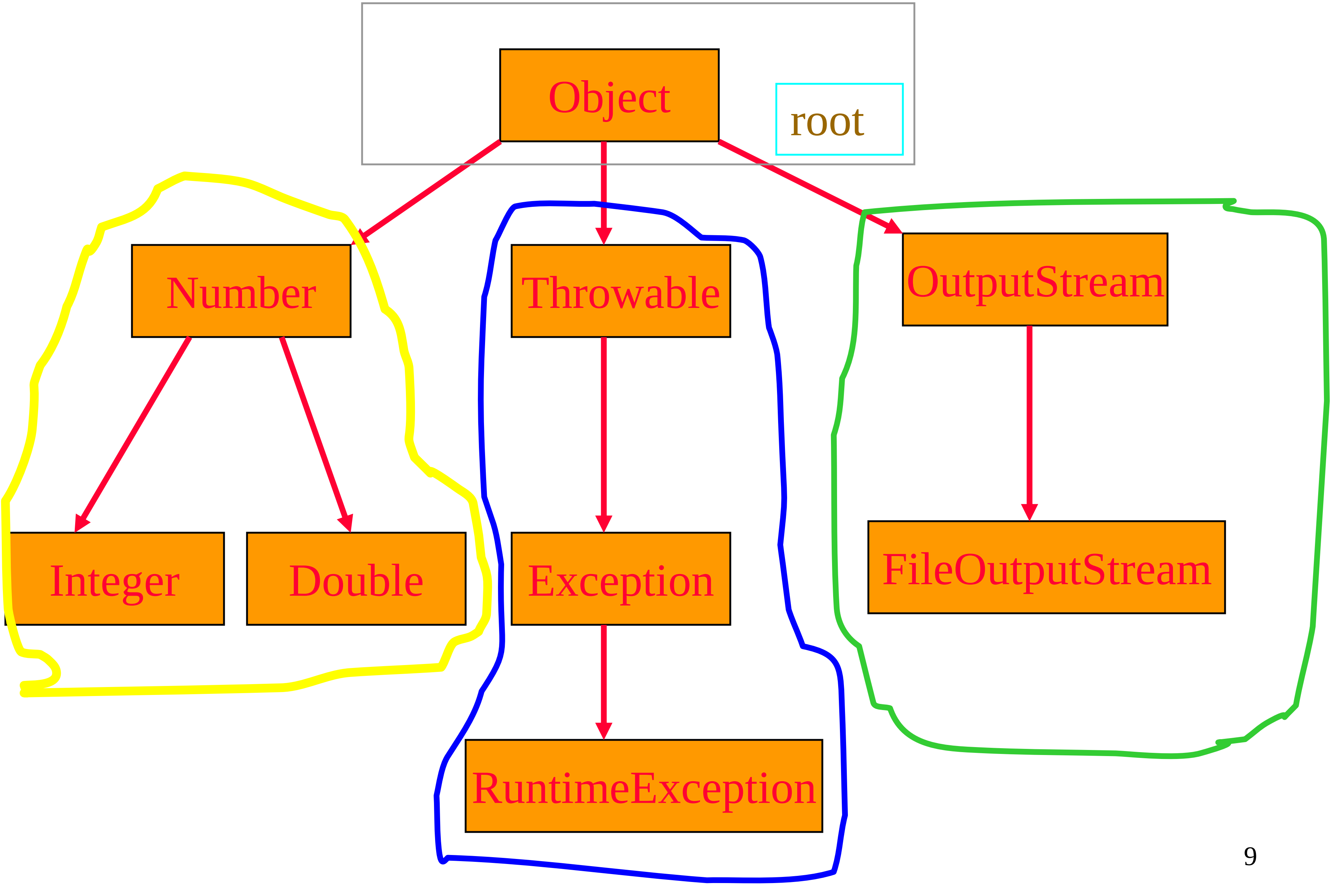


Definition

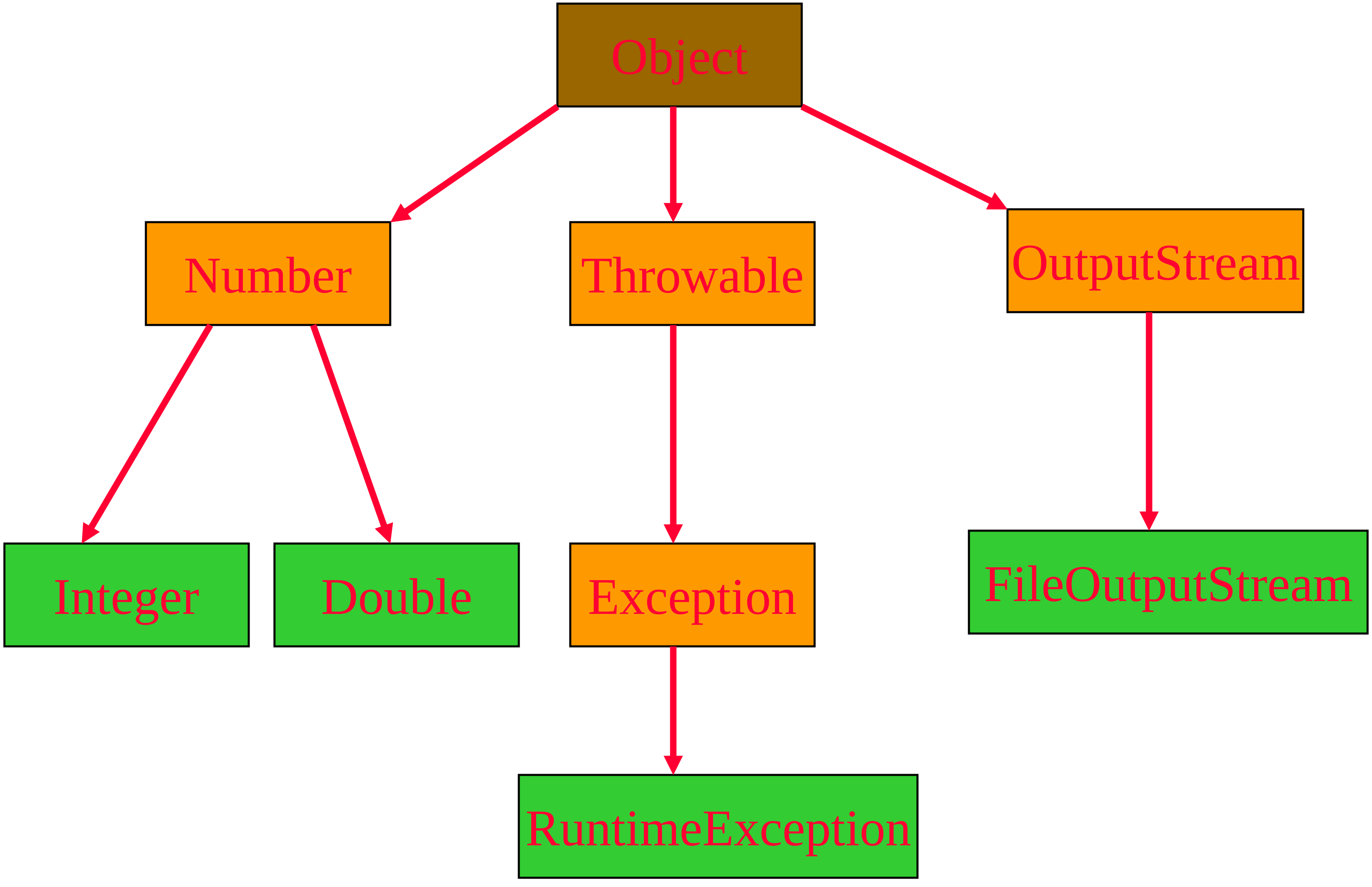
- A tree *t* is a finite nonempty set of elements.
- One of these elements is called the root.
- The remaining elements, if any, are partitioned into trees, which are called the subtrees of *t*.

注：树不能为空

Subtrees

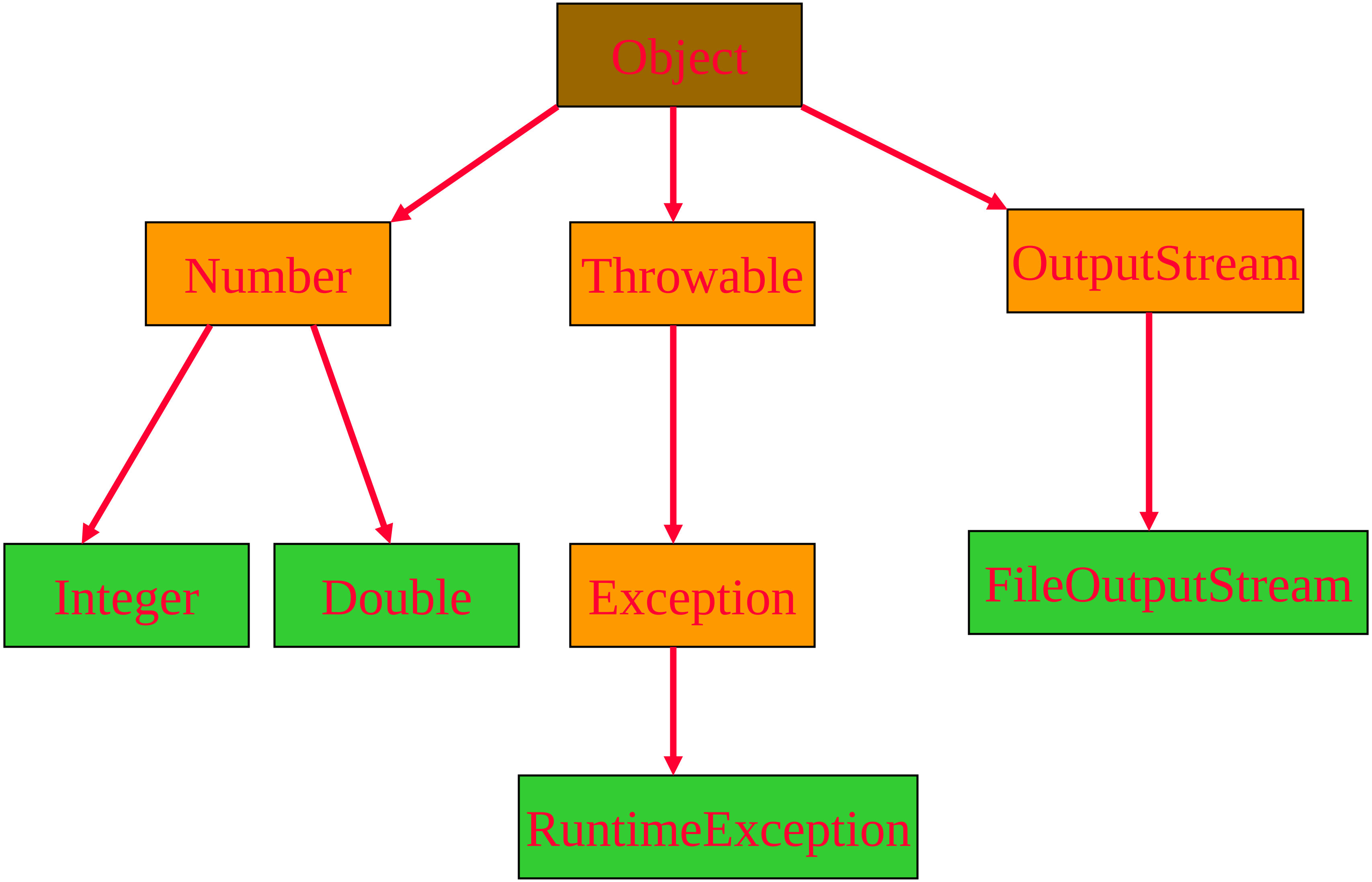

Object
root
Number
Throwable
OutputStream
Integer
Double
Exception
FileOutputStream
RuntimeException

Leaves

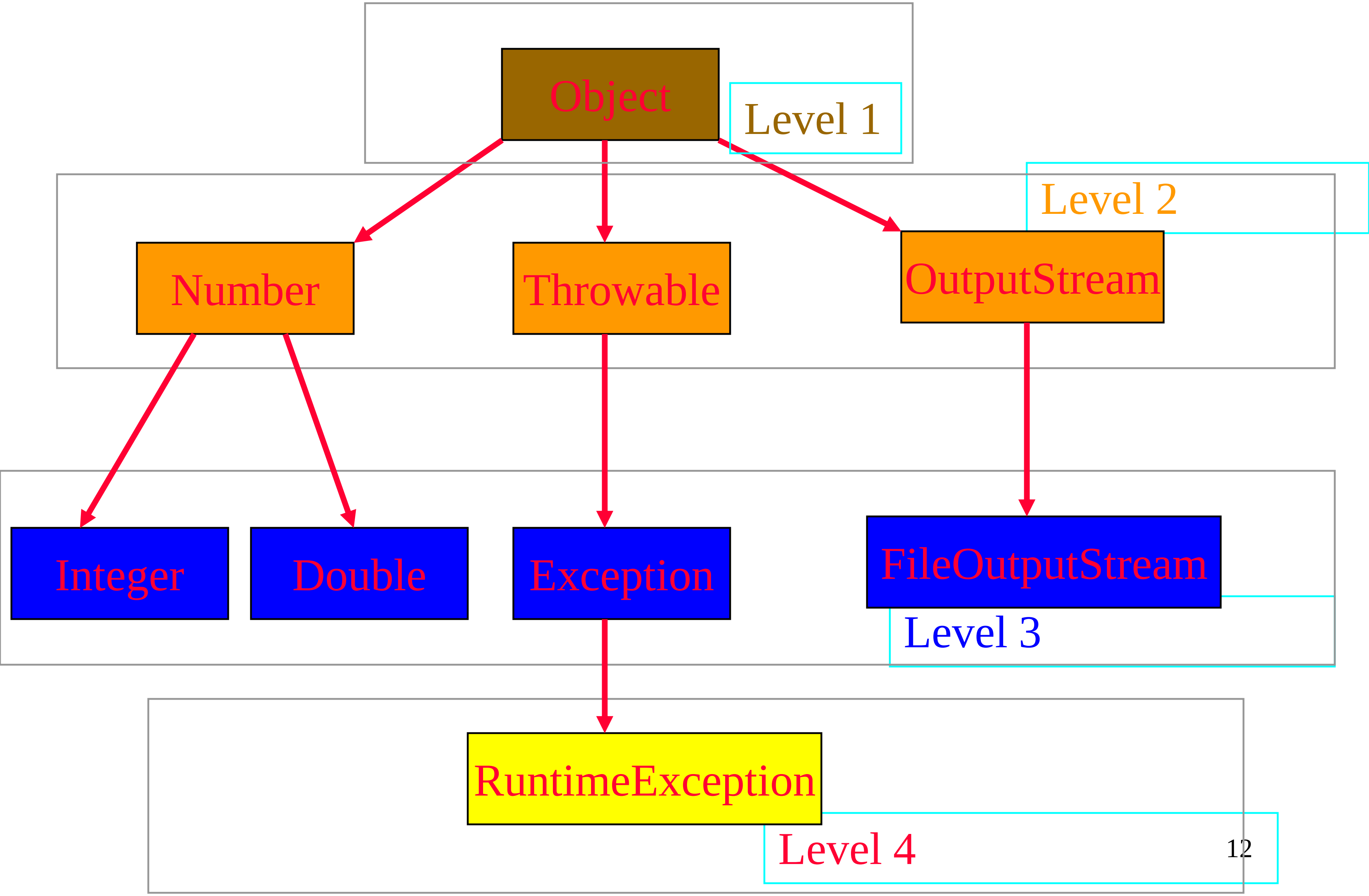


Object
Number
Throwable
OutputStream
Integer
Double
Exception
FileOutputStream
RuntimeException

Parent, Grandparent, Siblings, Ancestors, Descendants



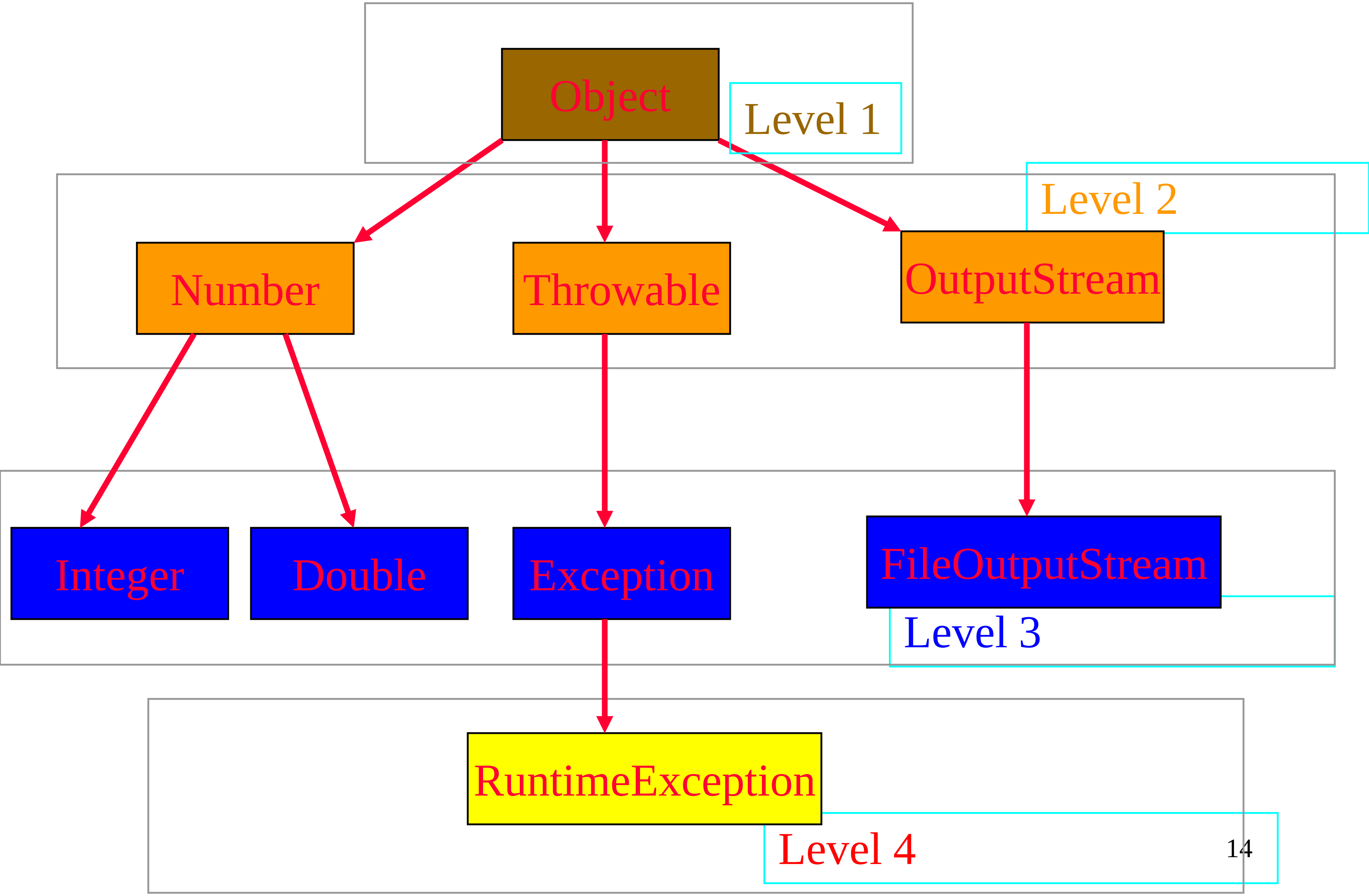
Levels


Object
Level 1
Level 2
Number
Throwable
OutputStream
Integer
Double
Exception
FileOutputStream
Level 3
RuntimeException
Level 4

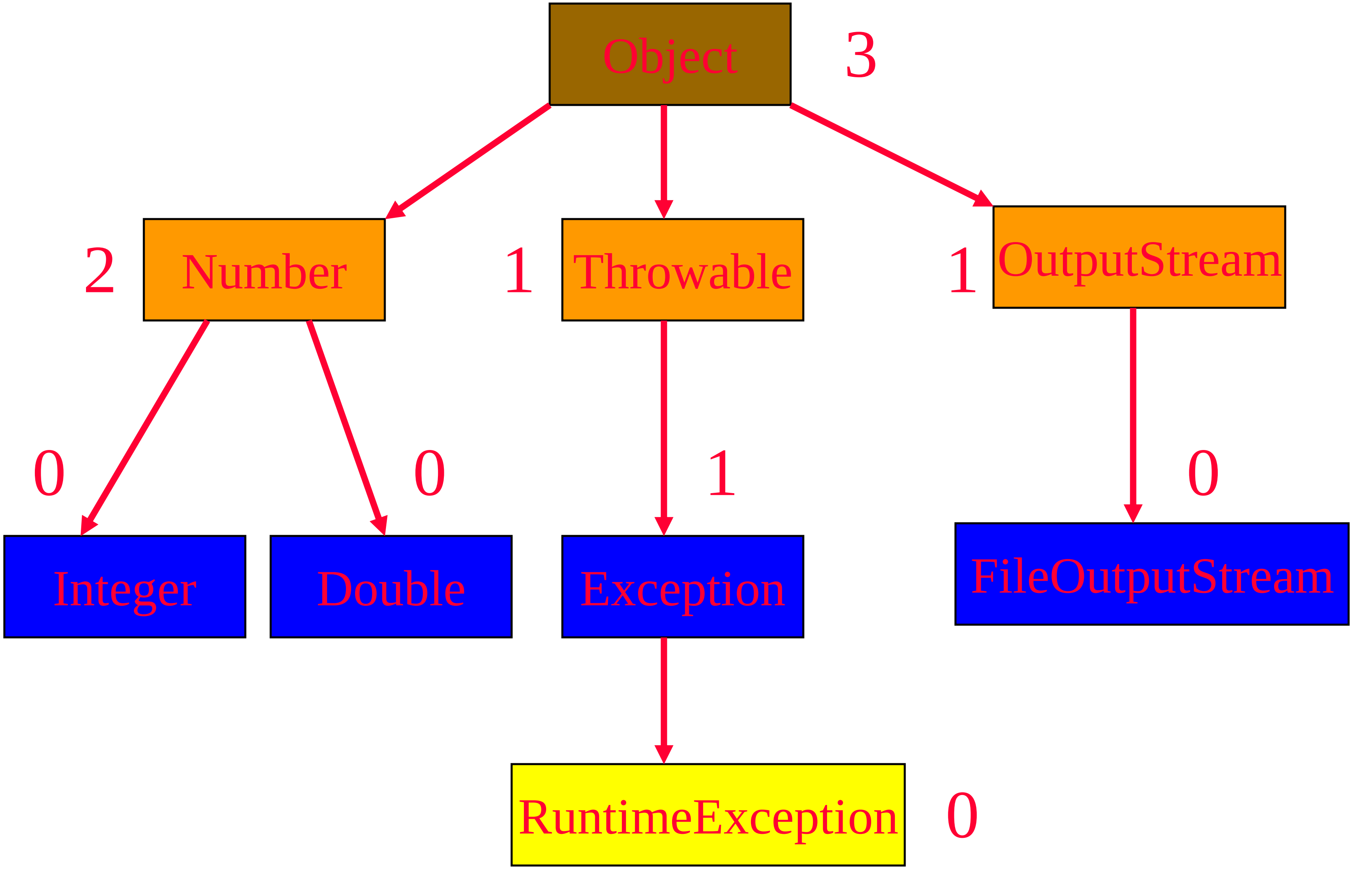
Caution

- 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.

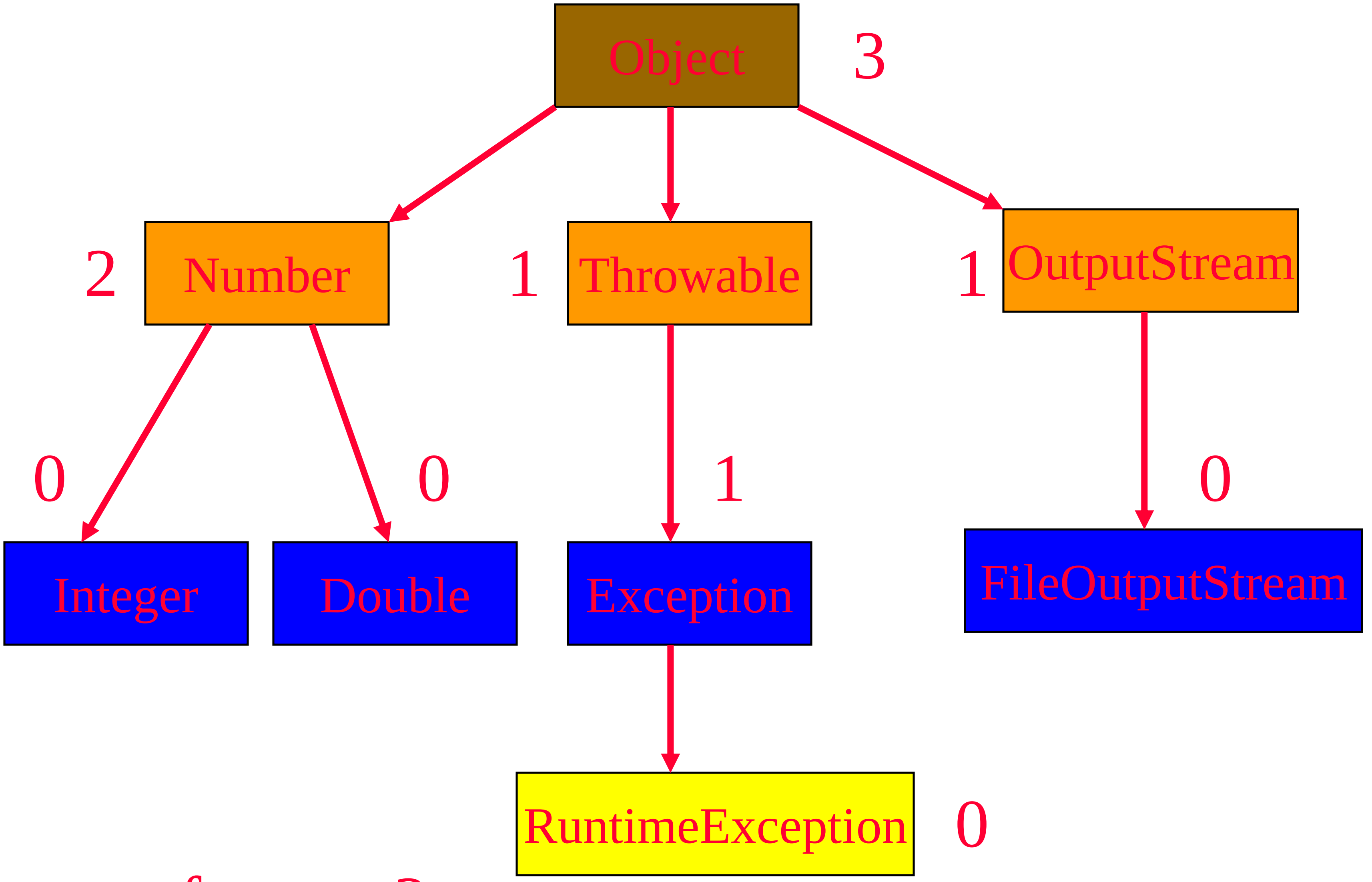
height = depth = number of levels



Node Degree = Number Of Children



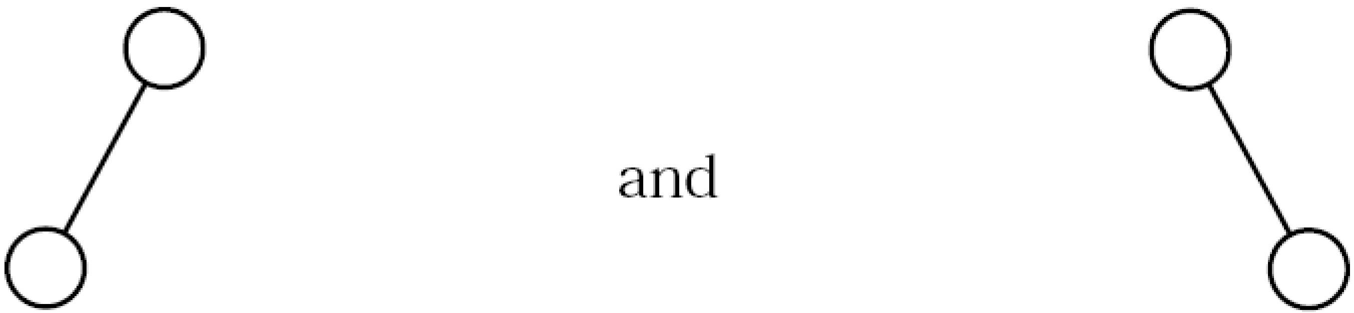
Tree Degree = Max Node Degree



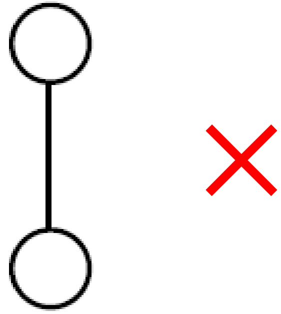
Degree of tree = 3.

10.1.1 Definitions of Binary Trees

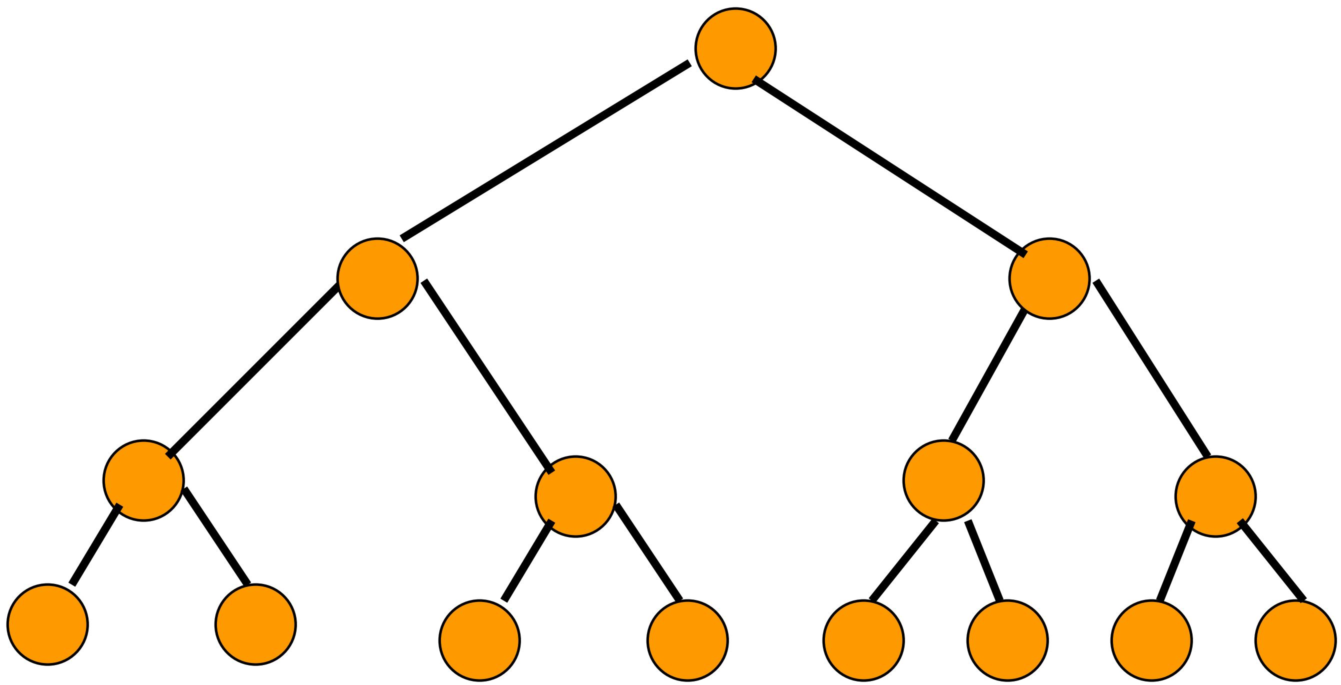
A ***binary tree*** is either empty, or it consists of a node called the ***root*** together with two binary trees called the ***left subtree*** and the ***right subtree*** of the root.



注：2-树不能为空，且每一个节点要么有0个子节点，要么有两个子节点。



10.1.1 Definitions of Binary Trees



10.1.1 Definitions of Binary Trees

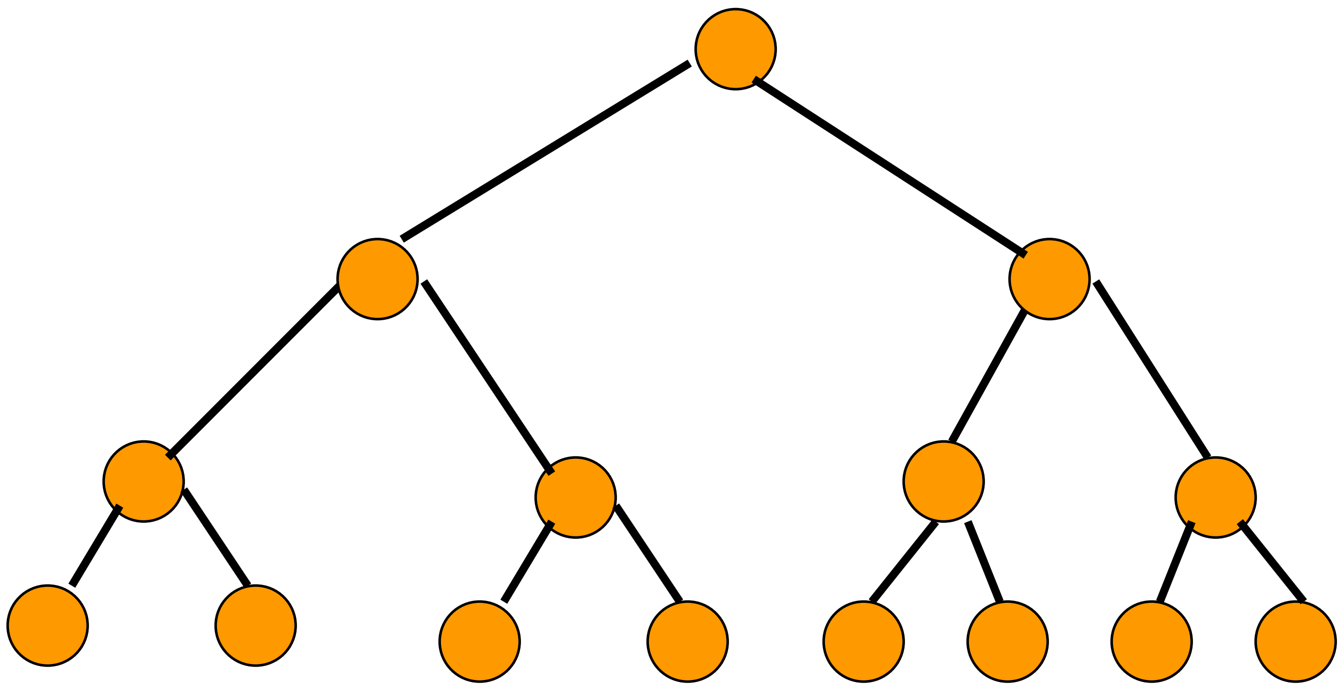
- Finite (possibly empty) collection of elements.
- A nonempty binary tree has a root element.
- The remaining elements (if any) are partitioned into two binary trees.
- These are called the left and right subtrees of the binary tree.

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.

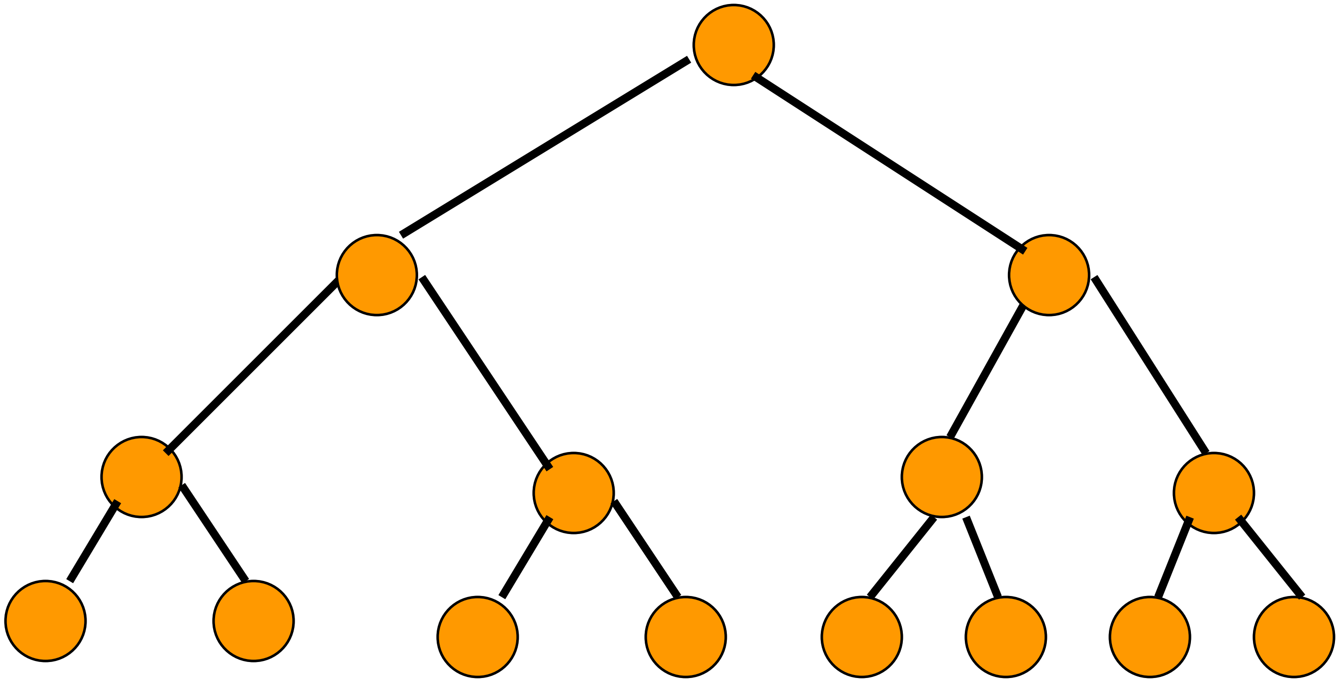
10.1.1 Definitions of Binary Trees

The drawing of every binary tree with *n* elements, *n*>0, has exactly n-1 edges.



Maximum Number Of Nodes

- All possible nodes at first h levels are present.



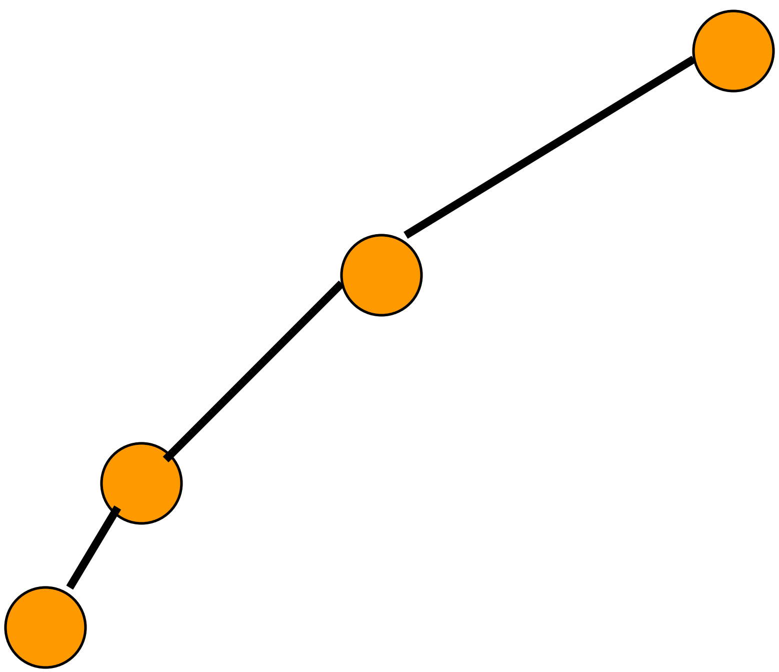
Maximum number of nodes

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

$= 2^h - 1$

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.



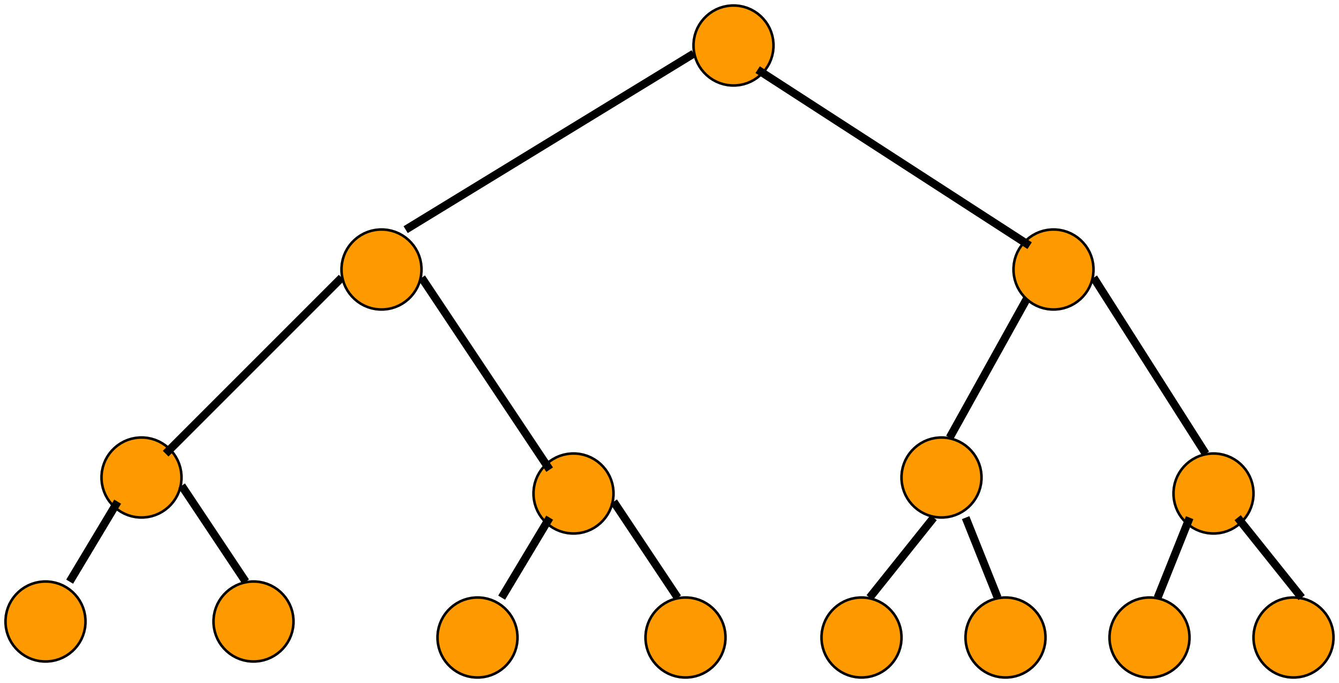
minimum number of nodes is h

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

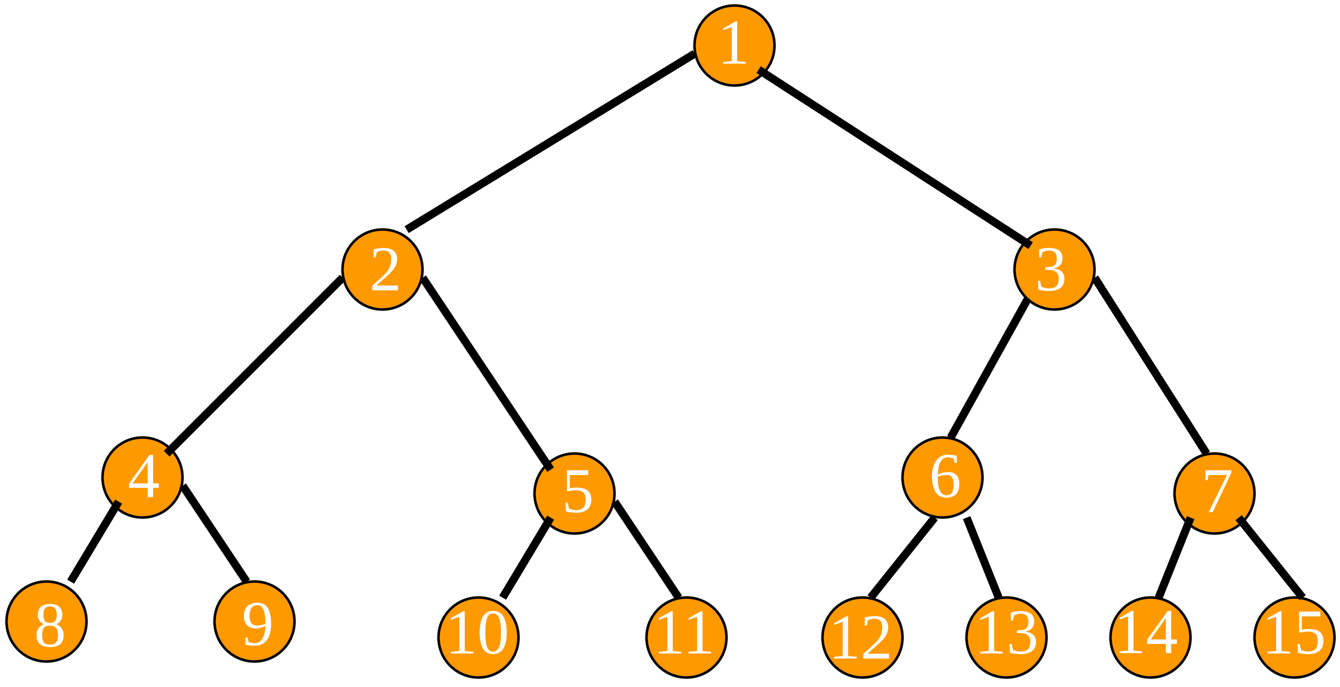
- A full binary tree of a given height h has $2^h - 1$ nodes.



Height 4 full binary tree.

Numbering Nodes In A Full Binary Tree

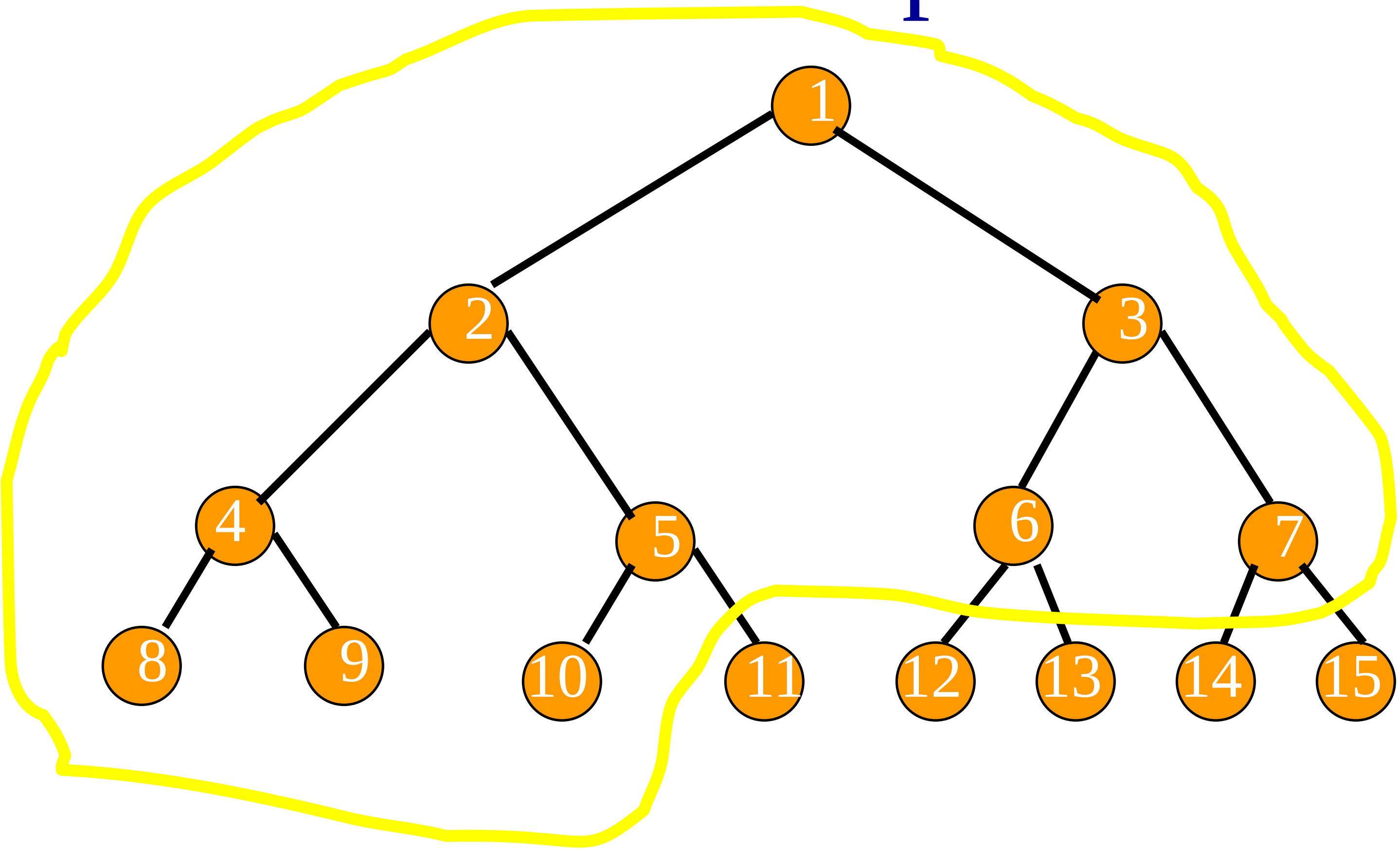
- Number the nodes 1 through $2^h - 1$.
- Number by levels from top to bottom.
- Within a level number from left to right.



Complete Binary Tree With *n* Nodes

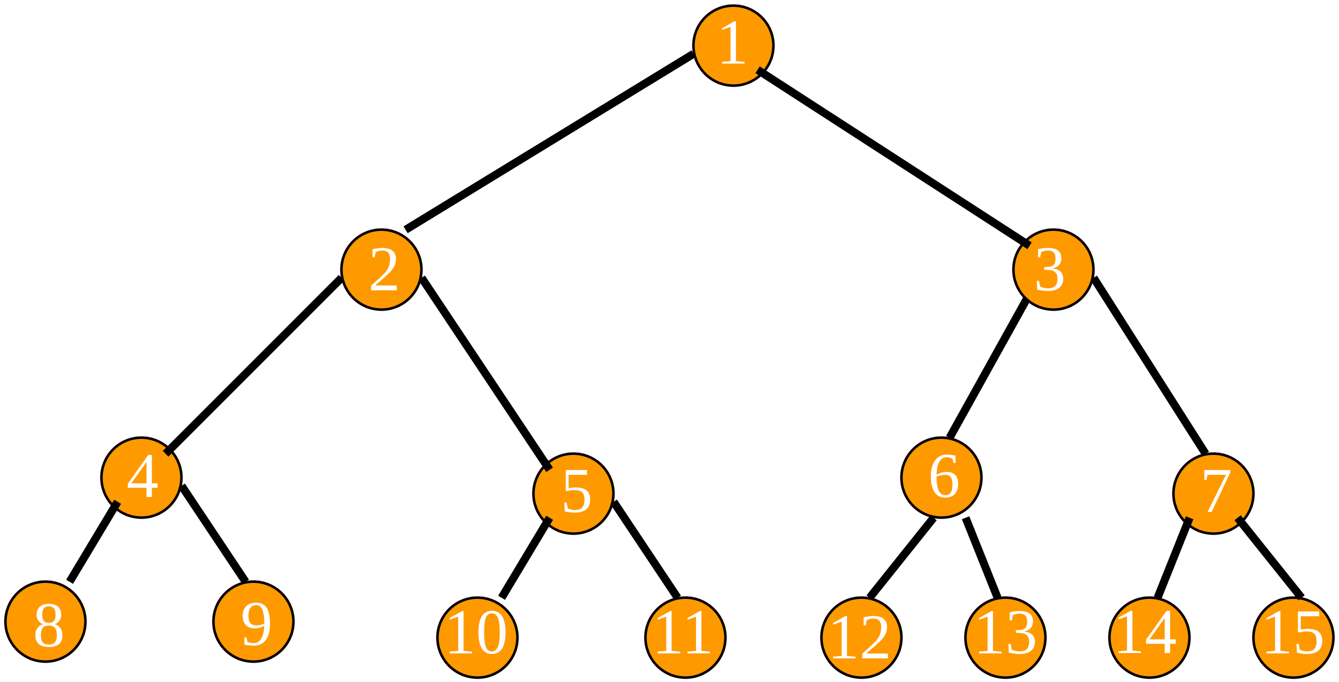
- Start with a full binary tree that has at least n nodes.
- Number the nodes as described earlier.
- The binary tree defined by the nodes numbered 1 through n is the unique n node complete binary tree.

Example



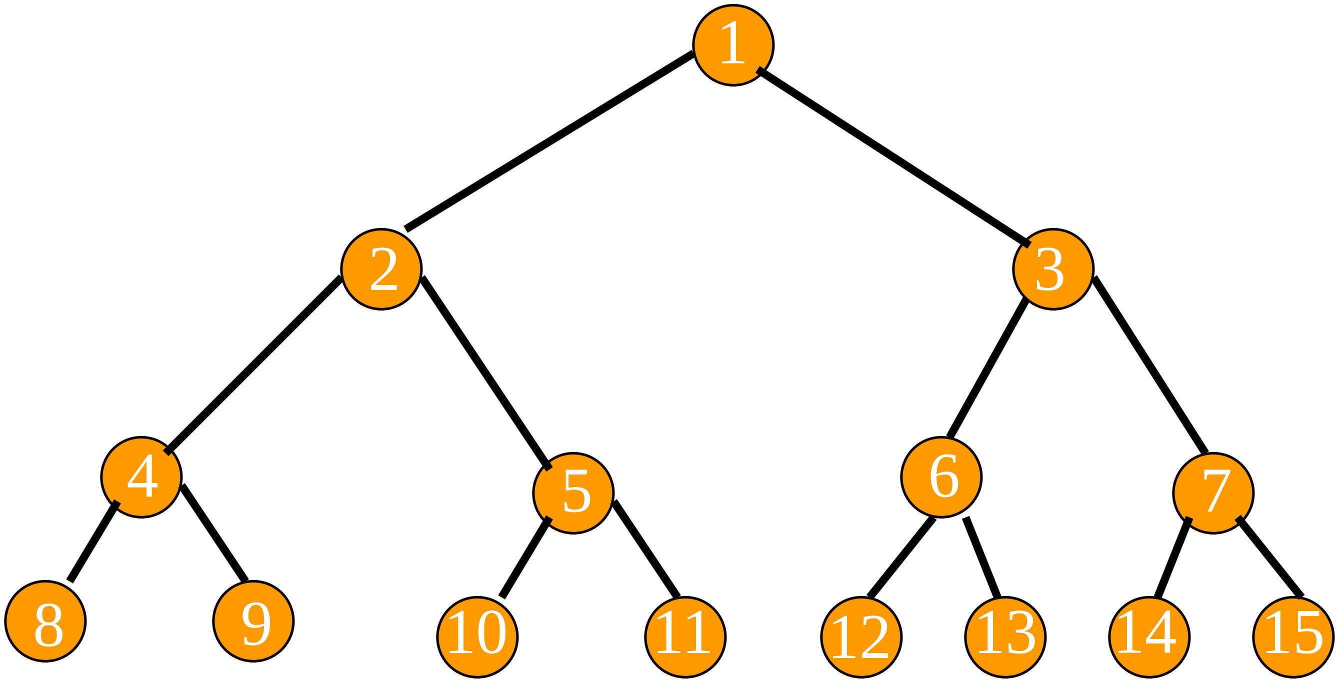
- Complete binary tree with 10 nodes.

Node Number Properties



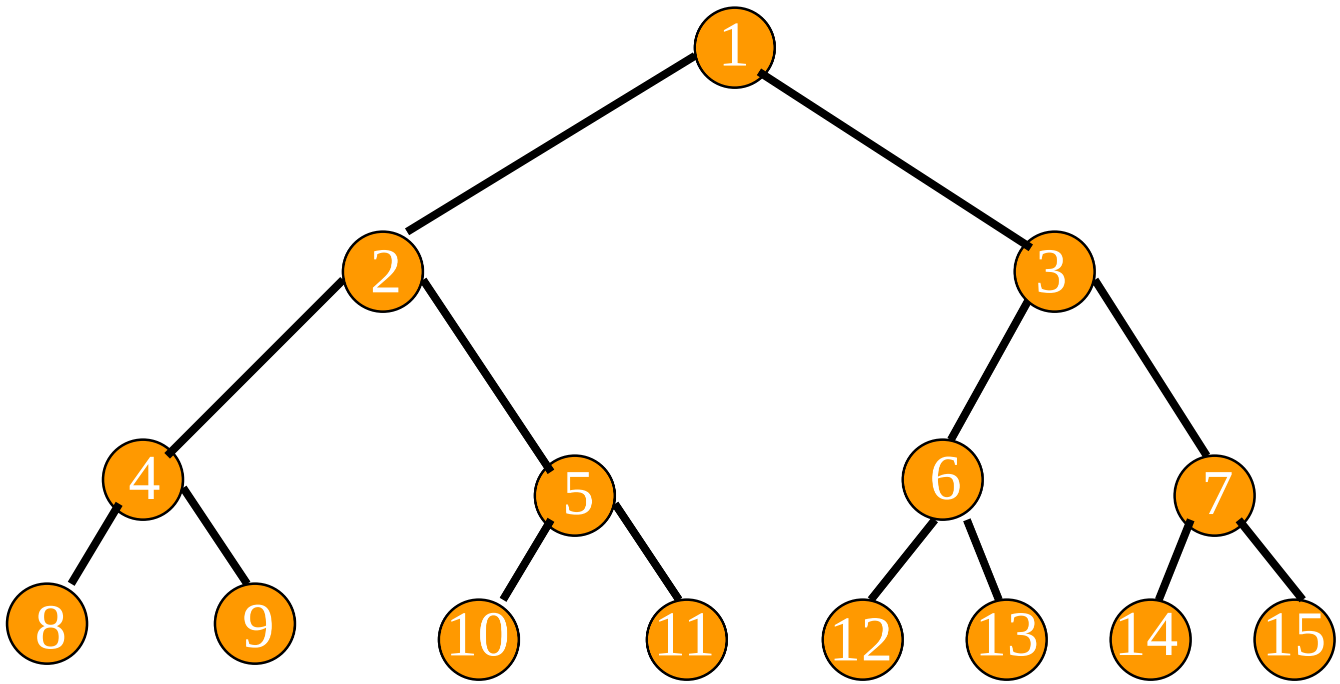
- Parent of node i is node $i / 2$, unless $i = 1$.
- Node 1 is the root and has no parent.

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



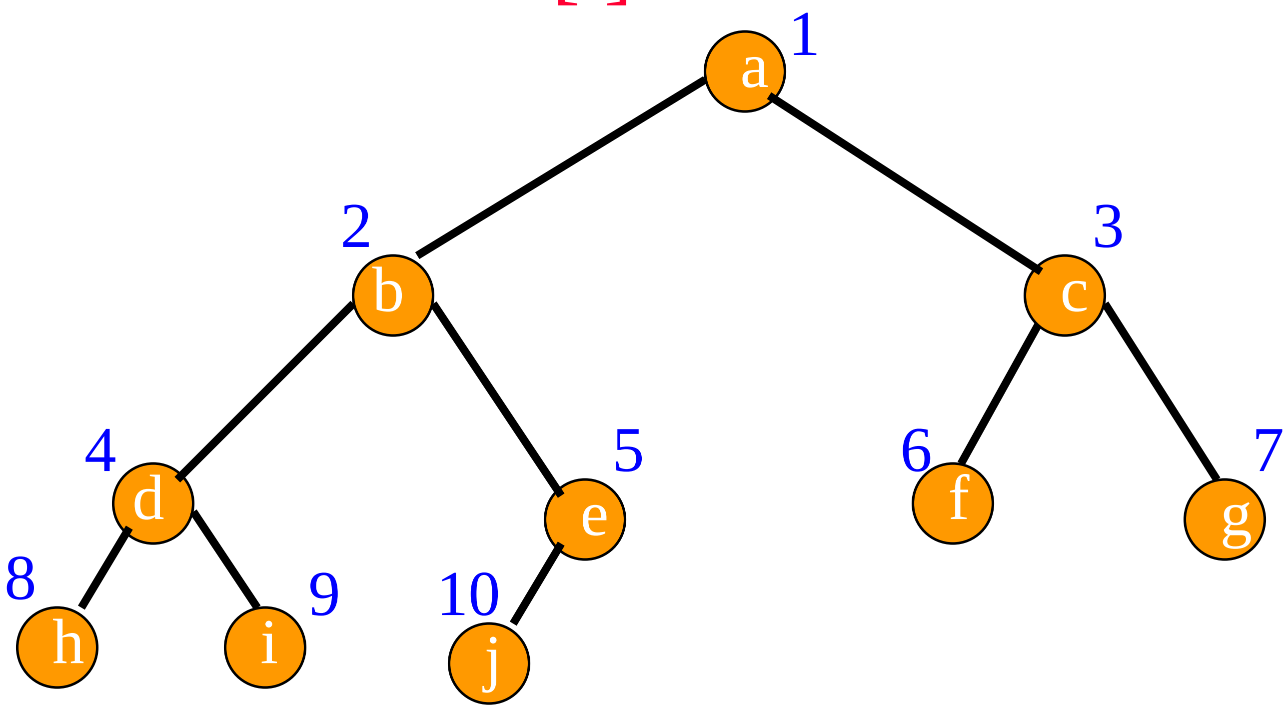
- Right child of node i is node $2i+1$, unless $2i+1 > n$, where n is the number of nodes.
- If $2i+1 > n$, node i has no right child.

10.1.3 Implementation of Binary Trees

- Array representation.
- Linked representation.

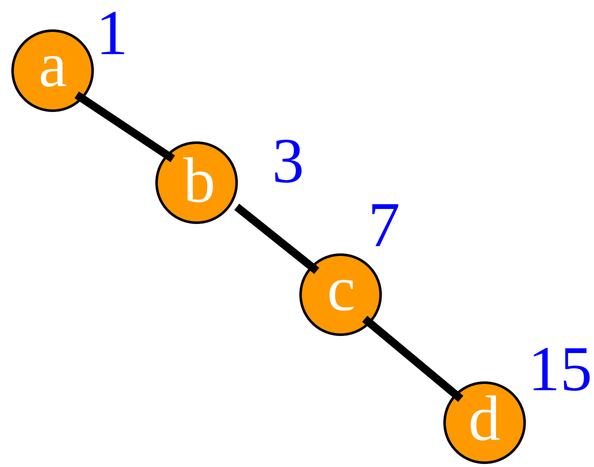
Array Representation

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



tree[]		a	b	c	d	e	f	g	h	i	j

Right-Skewed Binary Tree



tree[]		a	-	b	-	-	-	c	-	-	-	-	-	-	-	d

- 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 BinaryTreeNode.
- The space required by an n node binary tree is

$$n \times (\text{space required by one node}).$$

Linked Representation Example

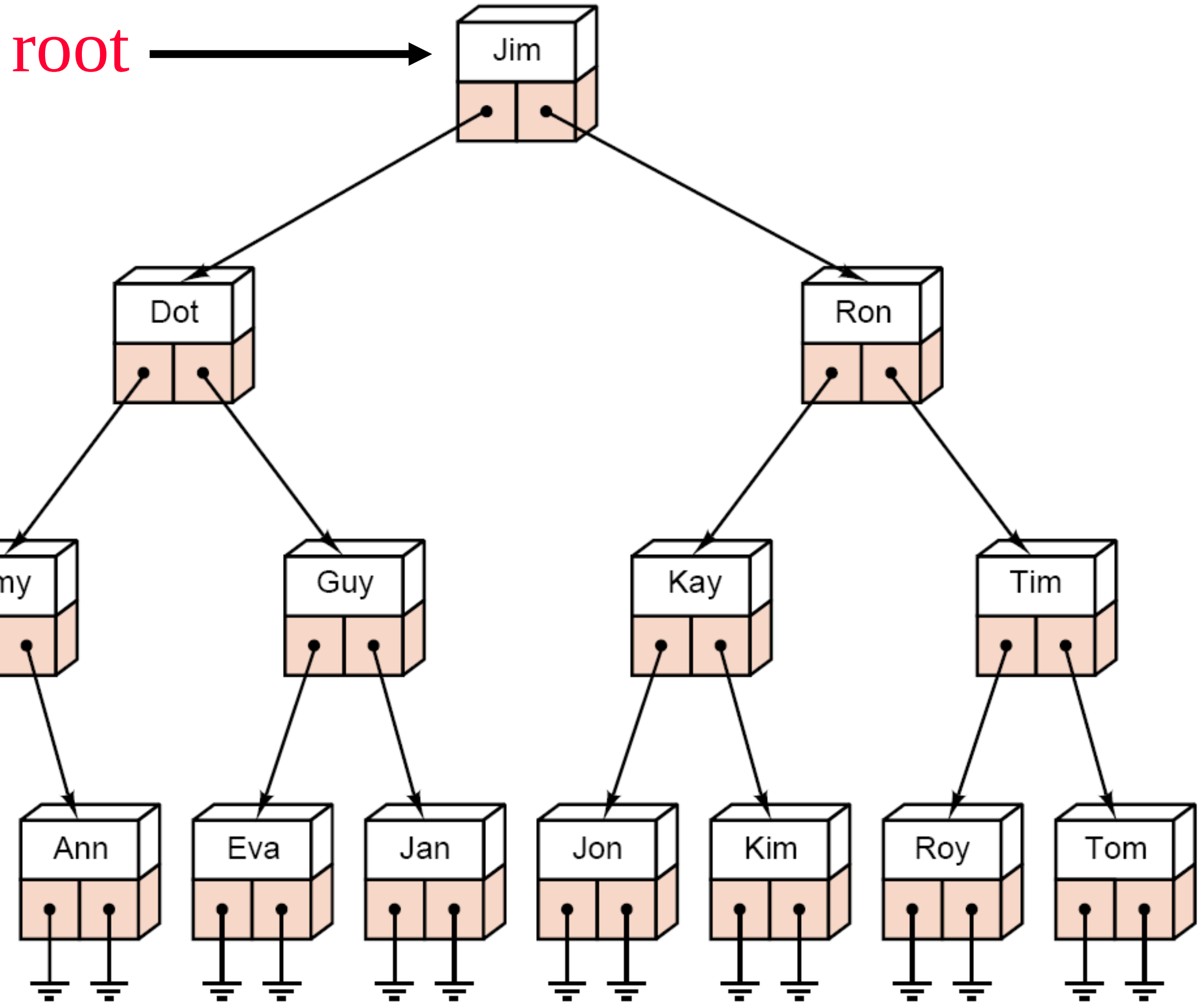


Figure 10.6. A linked binary tree

The Class Binary TreeNode

```
template <class Entry>
struct Binary_node {
//    data members:
   Entry data;
   Binary_node<Entry> *left;
   Binary_node<Entry> *right;
//    constructors:
   Binary_node();
   Binary_node(const Entry &x);

};
```

The Class Binary Tree

```
template <class Entry>
class Binary_tree {
public:
   Binary_tree( );
   bool empty( ) const;
   void preorder(void (*visit)(Entry &));
   void inorder(void (*visit)(Entry &));
   void postorder(void (*visit)(Entry &));

   int size( ) const;
   void clear( );
   int height( ) const;
   void insert(const Entry &);

   Binary_tree (const Binary_tree<Entry> &original);
   Binary_tree & operator = (const Binary_tree<Entry> &original);
    ~Binary_tree( );
protected:
   //    Add auxiliary function prototypes here.
   Binary_node<Entry> *root;
};
```

The Class Binary Tree

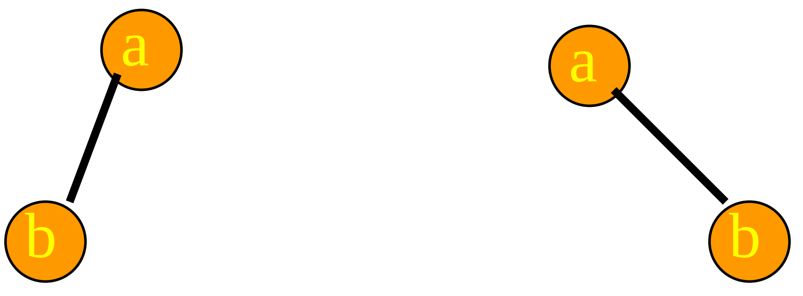
```
template <class Entry>
Binary_tree<Entry> :: Binary_tree()
/* Post:  An empty binary tree has been created. */
{
   root = NULL;
}
```

```
template <class Entry>
bool Binary_tree<Entry> :: empty() const
/* Post:  A result of true is returned if the binary tree is empty.  Otherwise, false is
          returned. */
{
   return root ==  NULL;
}
```

10.1.2 Traversal of Binary Trees

Differences Between A Tree & A Binary Tree

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

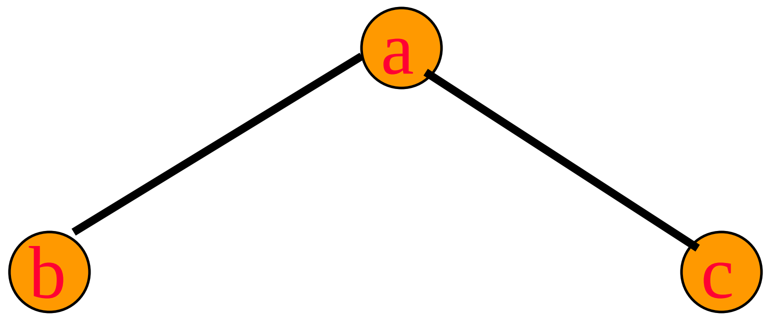


- Are different when viewed as binary trees.
- Are the same when viewed as trees.

10.1.2 Binary Tree Traversal Methods

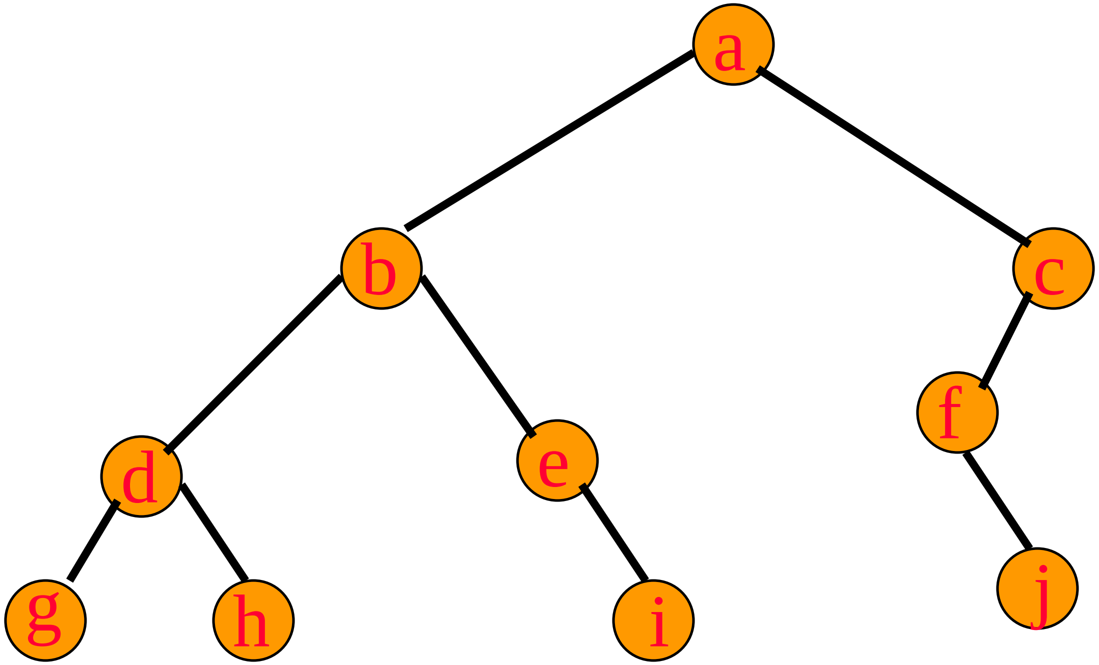
- Inorder
- Preorder
- Postorder
- Level order

Inorder Example (visit = print)



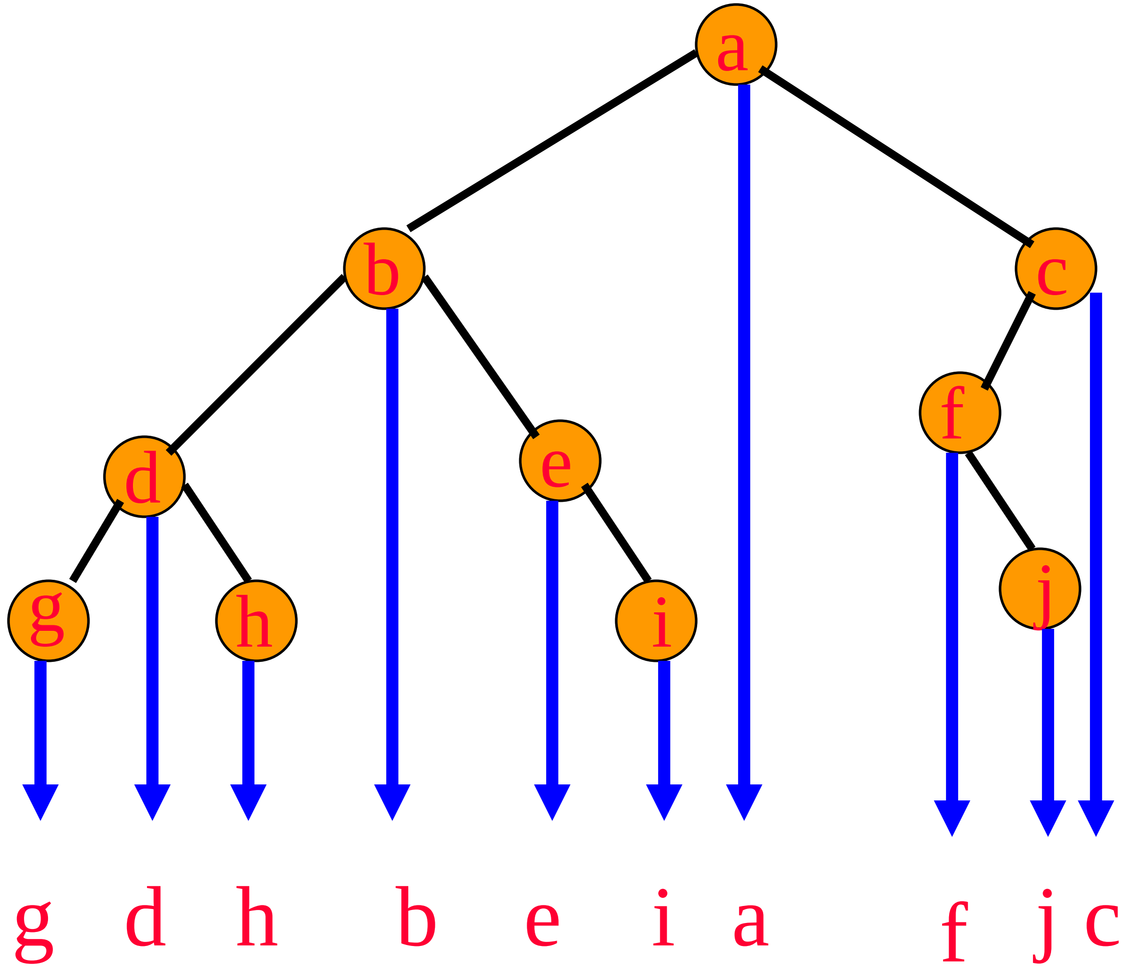
b a c

Inorder Example (visit = print)

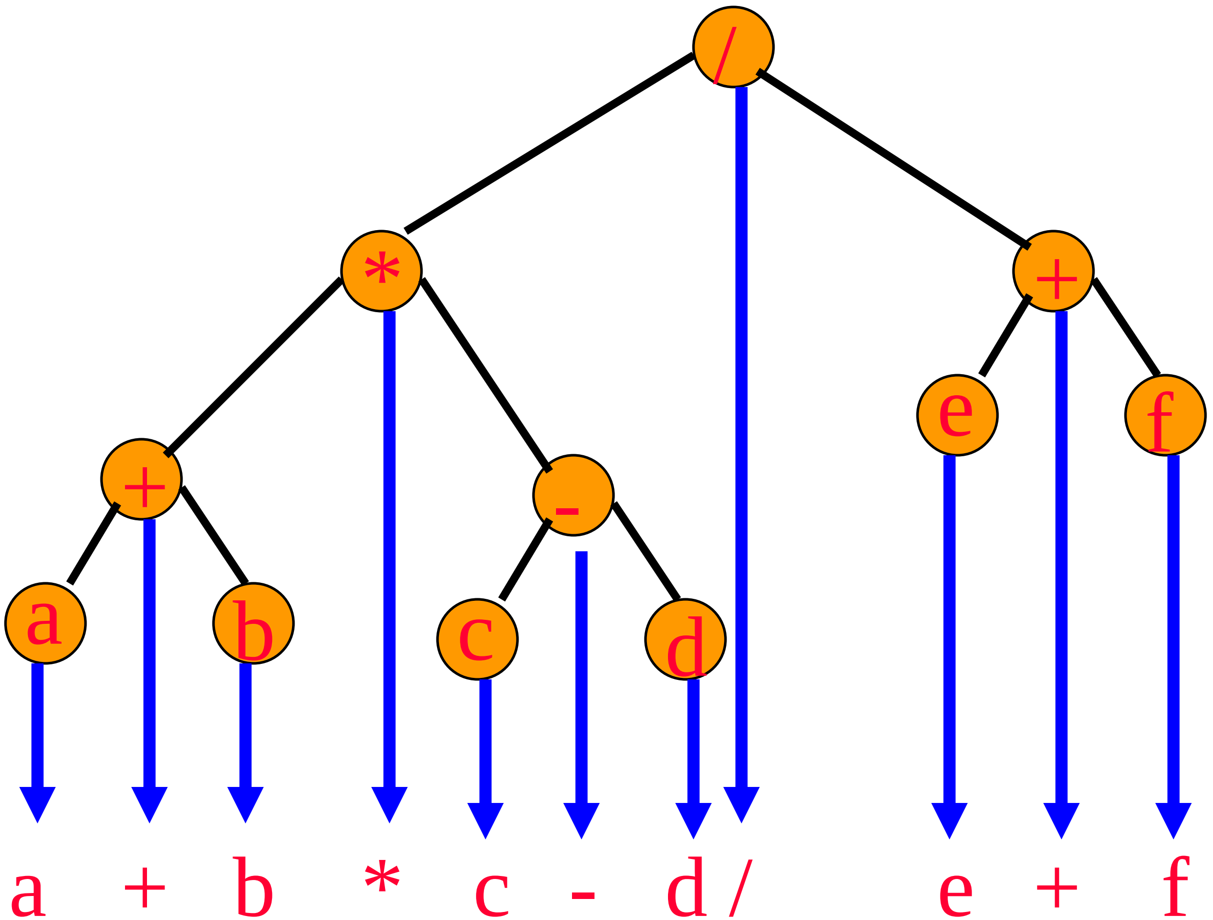


g d h b e i a f j c

Inorder By Projection (Squishing)



Inorder Of Expression Tree



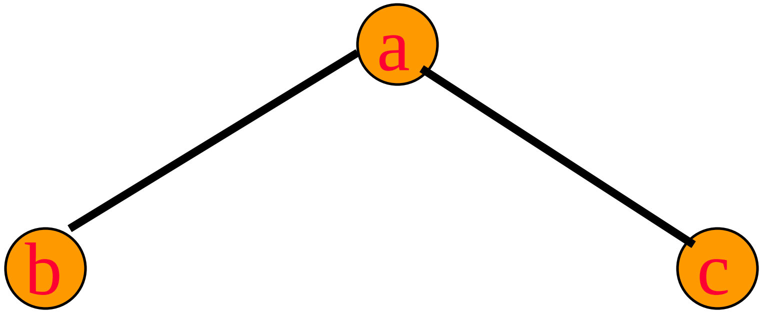
Gives infix form of expression (sans parentheses)!

Inorder Traversal

```
template <class Entry>
void Binary_tree<Entry> :: inorder(void (*visit)(Entry &))
/* Post: The tree has been been traversed in infix order sequence.
   Uses: The function recursive_inorder */
{
   recursive_inorder(root, visit);
}
```

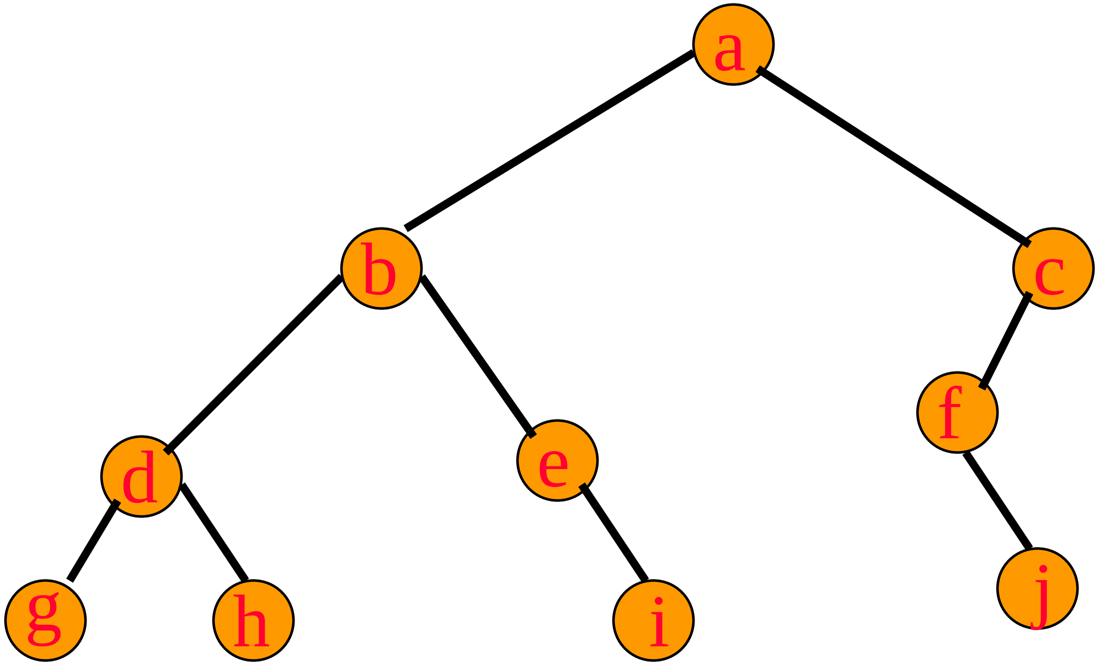
```
template <class Entry>
void Binary_tree<Entry> :: recursive_inorder(Binary_node<Entry> *sub_root,
                                              void (*visit)(Entry &))
/* Pre:  sub_root is either NULL or points to a subtree of the Binary_tree.
   Post: The subtree has been been traversed in inorder sequence.
   Uses: The function recursive_inorder recursively */
{
   if (sub_root != NULL) {
      recursive_inorder(sub_root->left, visit);
      (*visit)(sub_root->data);
      recursive_inorder(sub_root->right, visit);
   }
}
```

Preorder Example (visit = print)



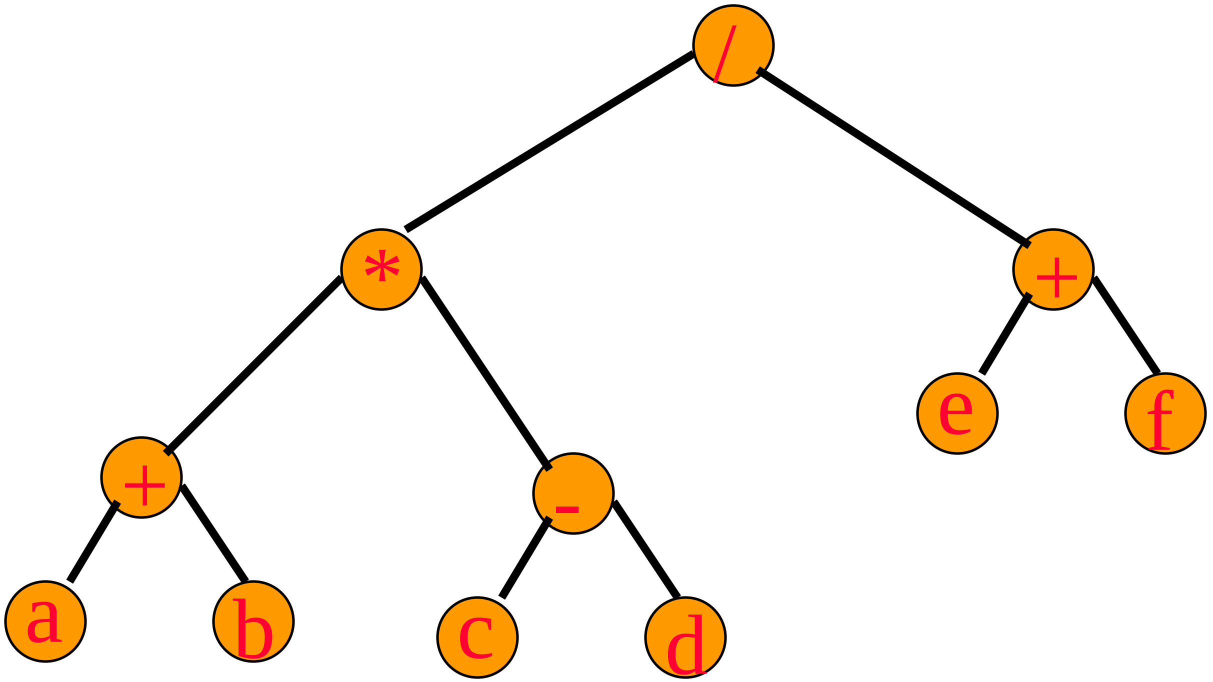
a b c

Preorder Example (visit = print)



a b d g h e i c f j

Preorder Of Expression Tree



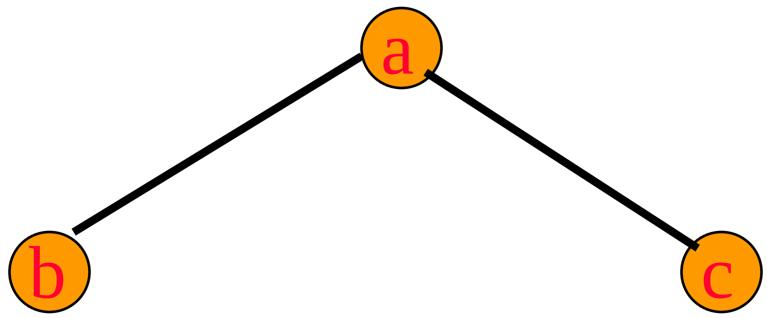
/ * + a b - c d + e f

Gives prefix form of expression!

Preorder Traversal

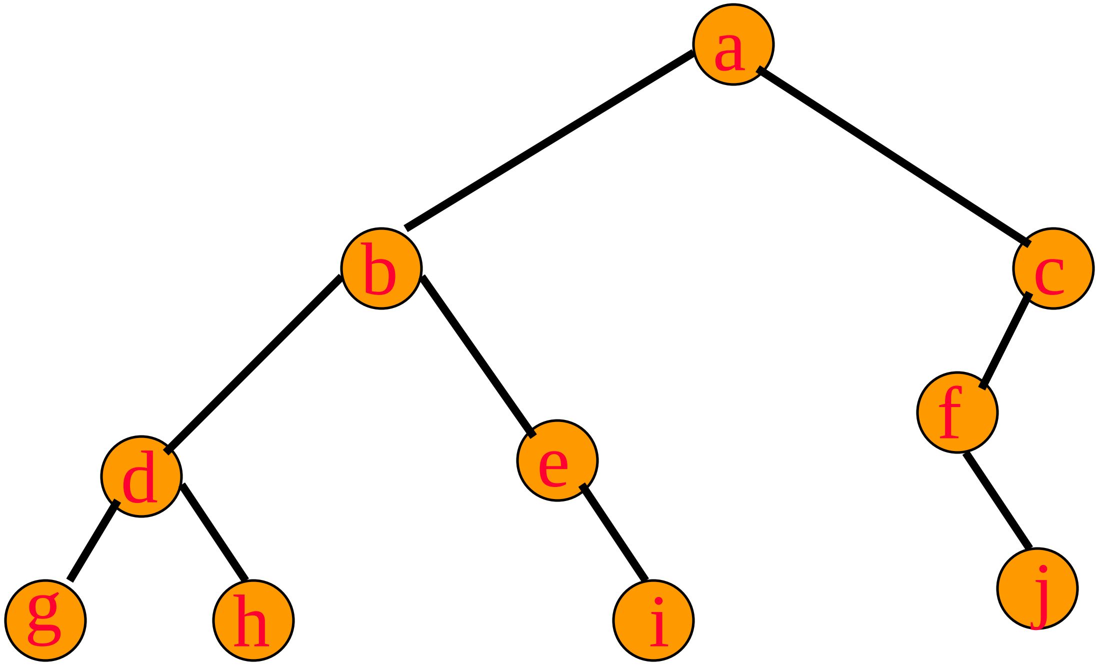
```
template <class Entry>
void Binary_tree<Entry> :: recursive_preorder(Binary_node<Entry> *sub_root,
                                              void (*visit)(Entry &))
/* Pre:  sub_root is either NULL or points to a subtree of the Binary_tree.
   Post: The subtree has been been traversed in preorder sequence.
   Uses: The function recursive_preorder recursively */
{
  if (sub_root != NULL) {
    (*visit)(sub_root->data);
    recursive_preorder(sub_root->left, visit);
    recursive_preorder(sub_root->right, visit);
  }
}
```

Postorder Example (visit = print)



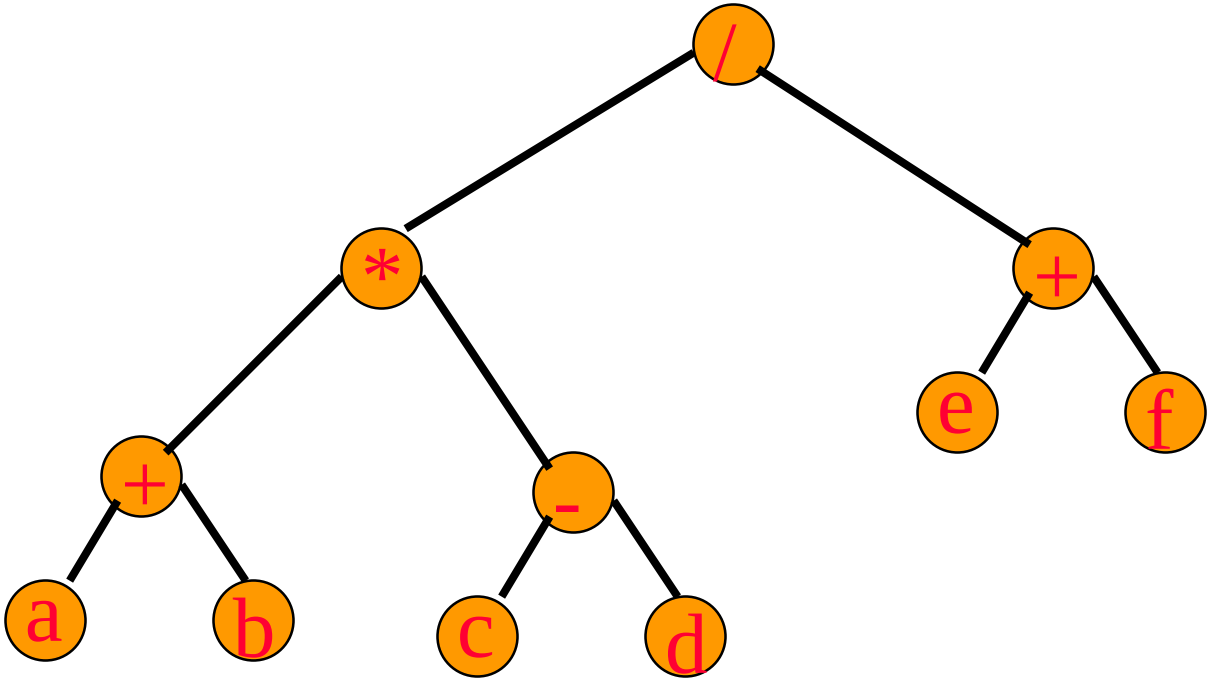
b c a

Postorder Example (visit = print)



g h d i e b j f c a

Postorder Of Expression Tree



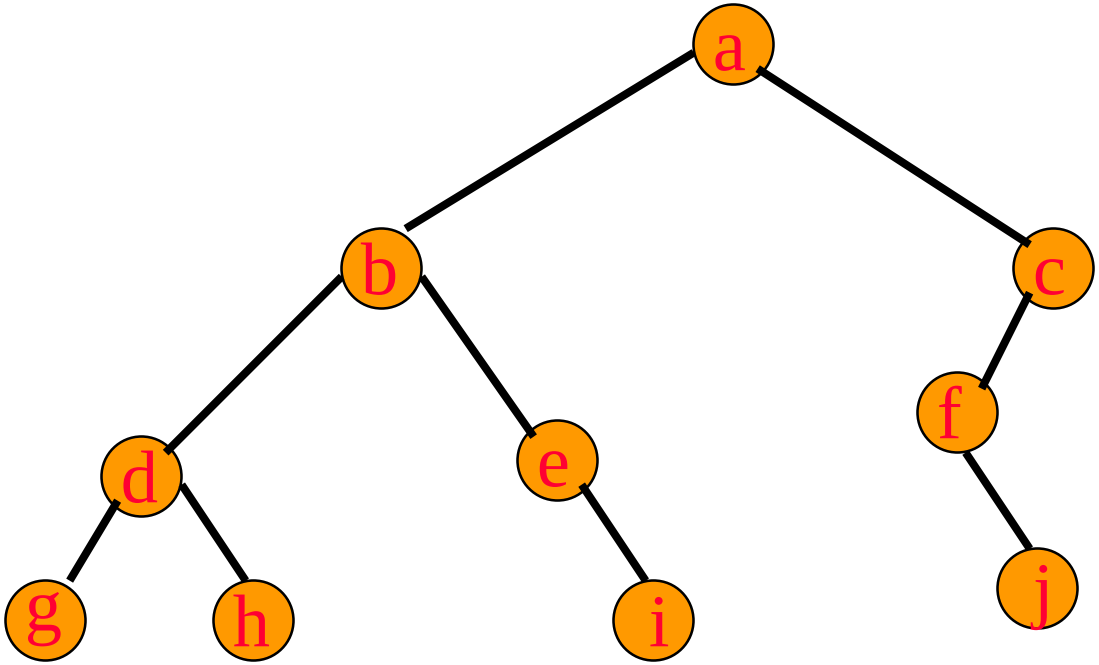
a b + c d - * e f + /

Gives postfix form of expression!

Postorder Traversal

```
template <class Entry>
void Binary_tree<Entry> :: recursive_postorder(Binary_node<Entry> *sub_root,
                                               void (*visit)(Entry &))
/* Pre:   sub_root is either NULL or points to a subtree of the Binary_tree.
   Post:  The subtree has been been traversed in postorder sequence.
   Uses:  The function recursive_postorder recursively */
{
  if (sub_root != NULL) {
    recursive_postorder(sub_root->left, visit);
    recursive_postorder(sub_root->right, visit);
    (*visit)(sub_root->data);
  }
}
```

Level-Order Example (visit = print)



a b c d e f g h i j

Level Order (程序供参考)

Let t be the tree root.

```
void BinaryTree<int>::LevelOrder(
        void(*Visit)(BinaryTreeNode<int> *u))
{// Level-order traversal.
  LinkedQueue<BinaryTreeNode<T>*> Q;
  BinaryTreeNode<T> *t;
  t = root;
  while (t) {
    Visit(t);
    if (t->LeftChild) Q.Add(t->LeftChild);
    if (t->RightChild) Q.Add(t->RightChild);
    try {Q.Delete(t);}
    catch (OutOfBounds) {return;}
    }
}
```

Expression Trees

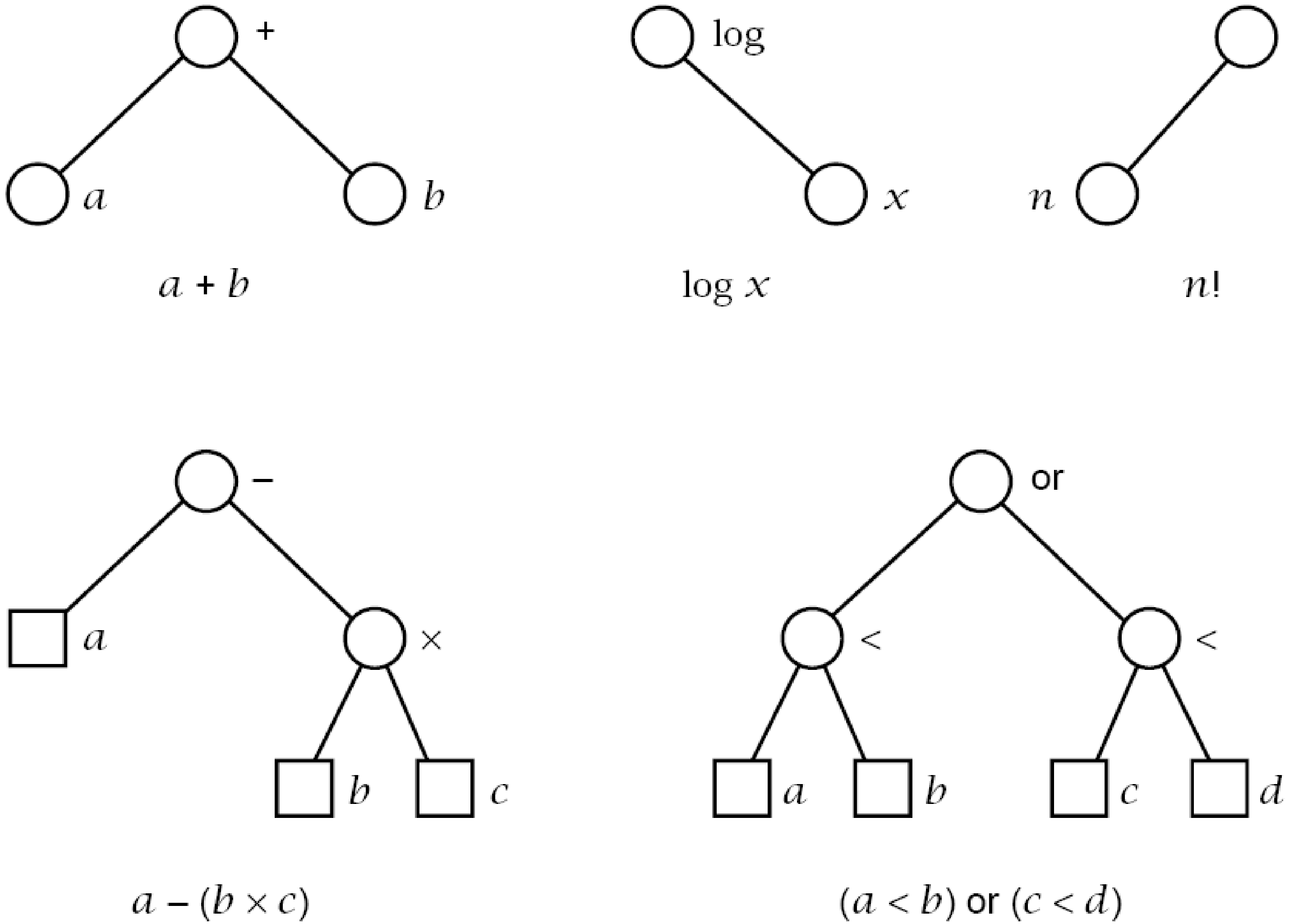
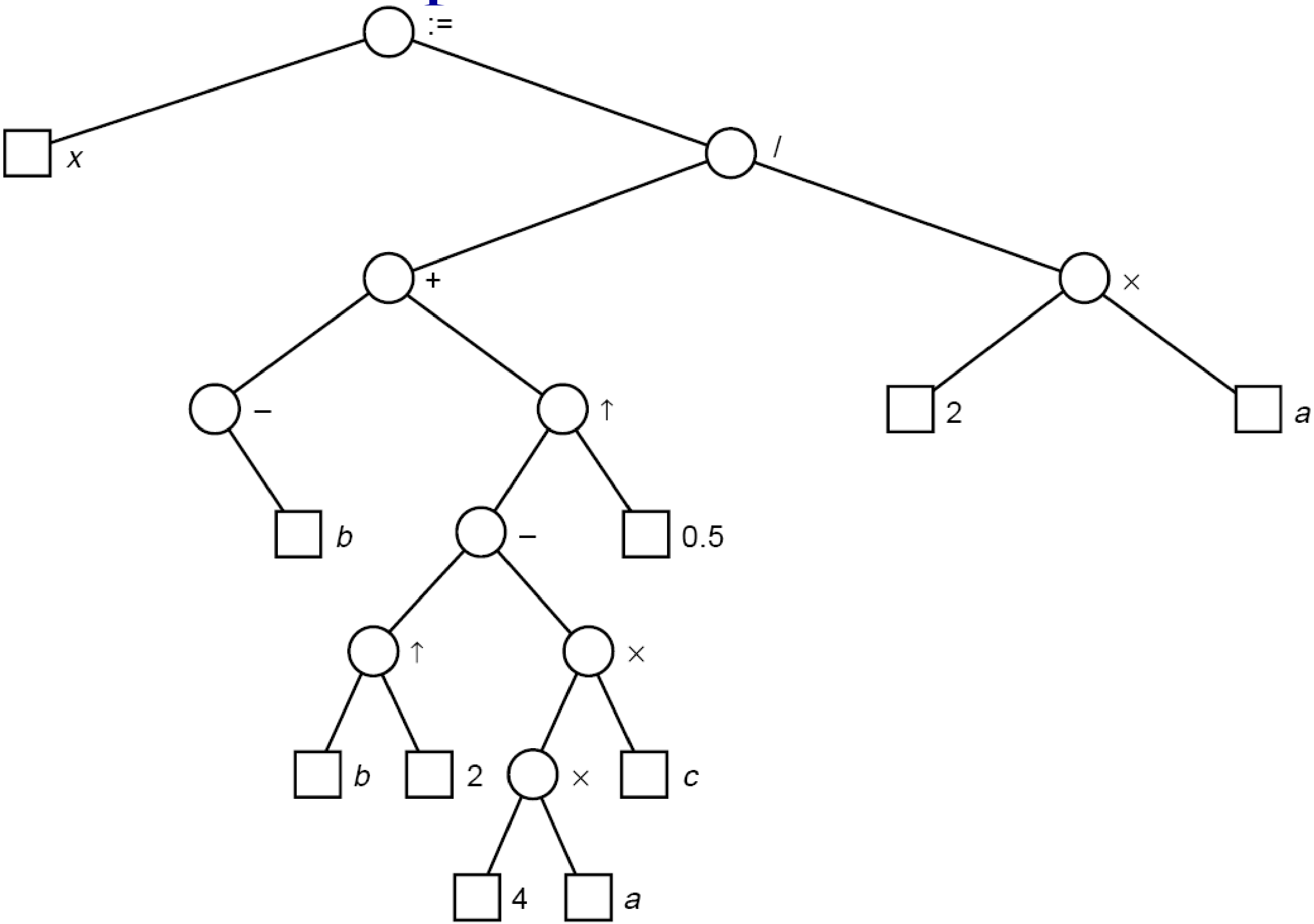


Figure 10.3. Expression trees

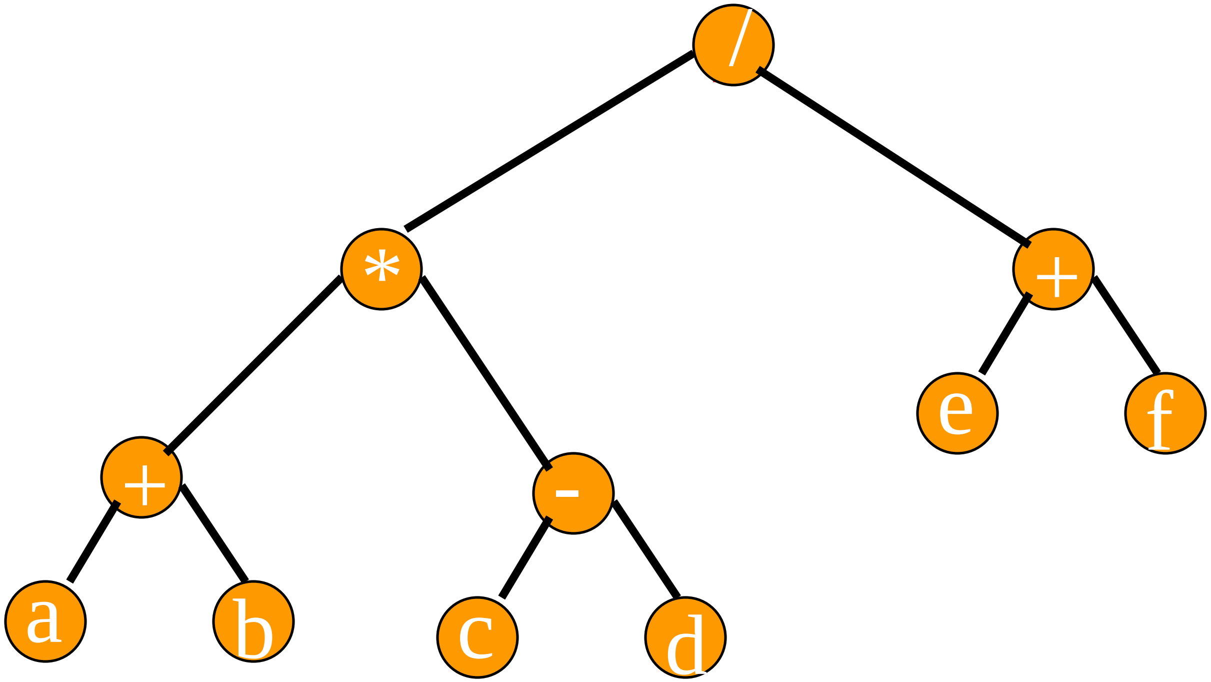
Expression Trees



$x := (-b + (b \uparrow 2 - 4 \times a \times c) \uparrow 0.5)/(2 \times a)$

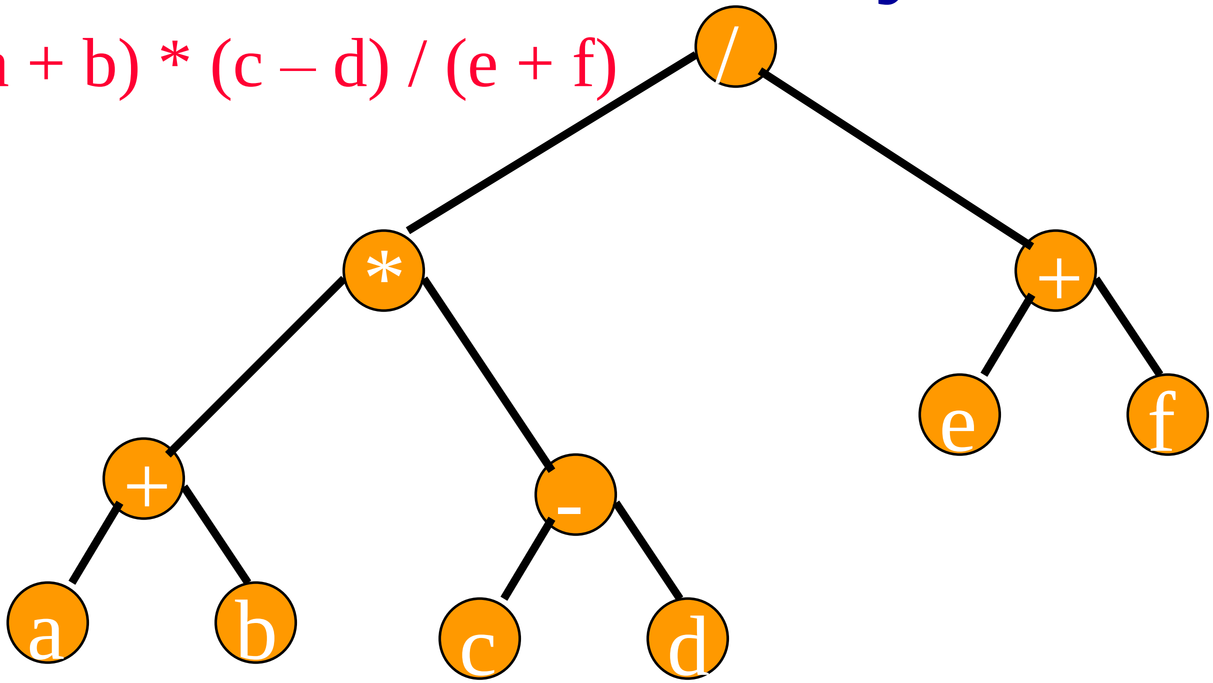
Expression Trees

- (a + b) * (c – d) / (e + f)



Merits Of Binary Tree Form

- (a + b) * (c – d) / (e + f)



- 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.

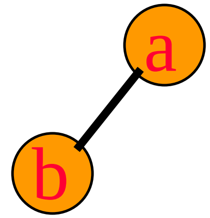
Binary Tree Construction

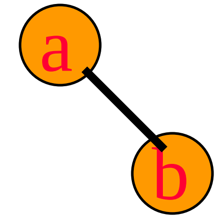
- Suppose that the elements in a binary tree are distinct.
- Can you construct the binary tree from which a given traversal sequence came?
- When a traversal sequence has more than one element, the binary tree is not uniquely defined.
- Therefore, the tree from which the sequence was obtained cannot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

Some Examples

preorder

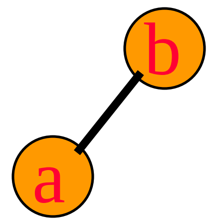
= ab

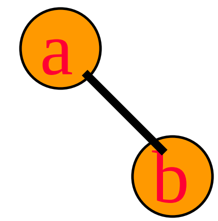




inorder

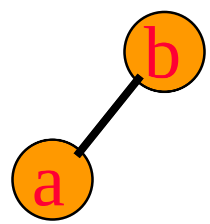
= ab

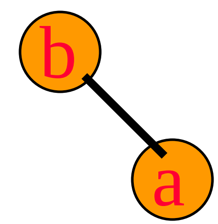




postorder

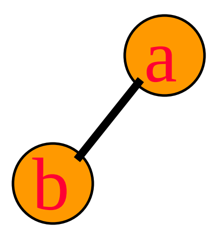
= ab

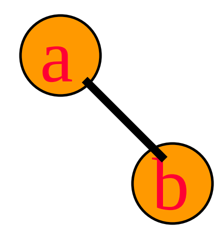




level order

= ab





Binary Tree Construction

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

Preorder And Postorder

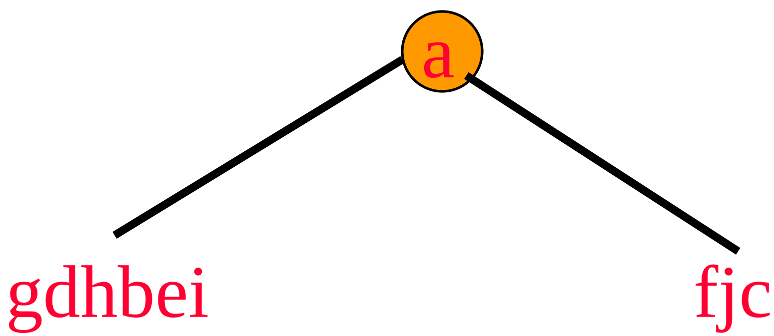
preorder = ab

postorder = ba

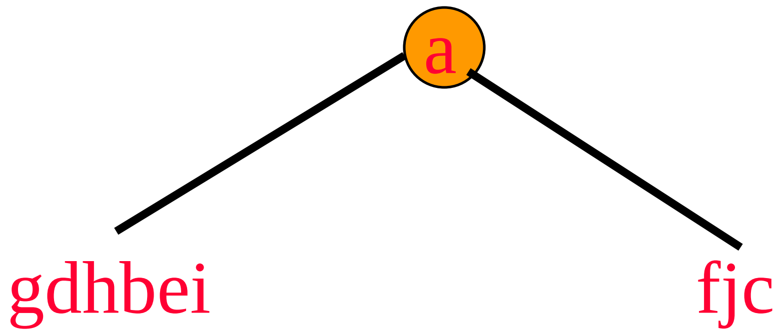
- Preorder and postorder do not uniquely define a binary tree.
- Nor do preorder and level order (same example).
- Nor do postorder and level order (same example).

Inorder And Preorder

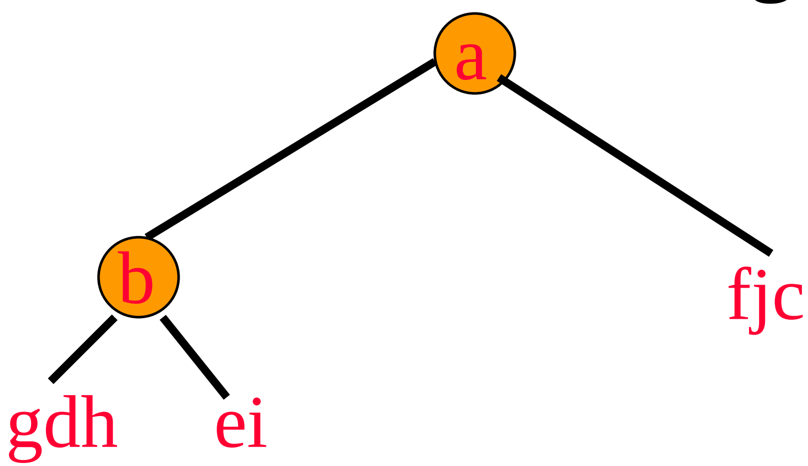
- inorder = g d h b e i a f j c
- preorder = a b d g h e i c f j
- Scan the preorder left to right using the inorder to separate left and right subtrees.
- a is the root of the tree; gdhbei are in the left subtree; fjc are in the right subtree.



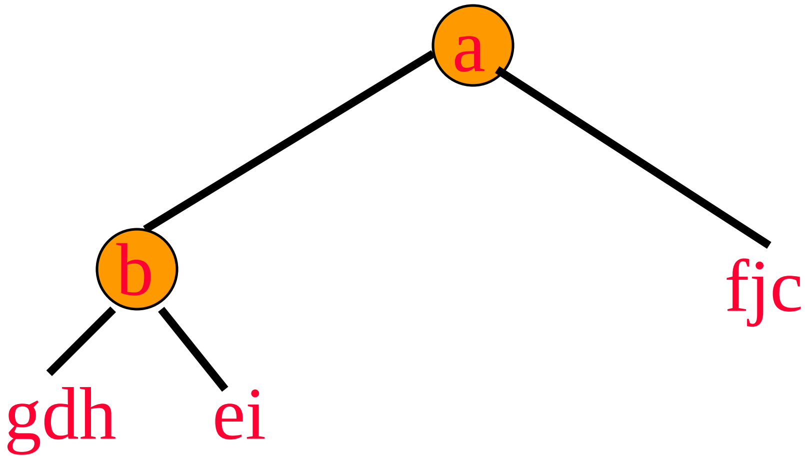
Inorder And Preorder



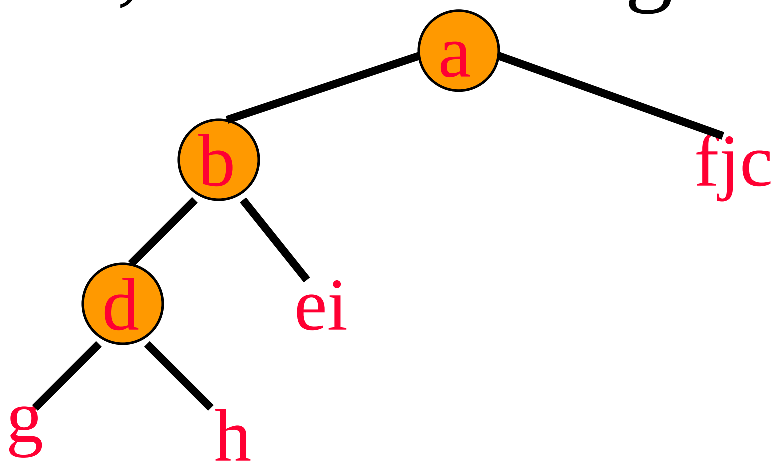
- preorder = a b d g h e i c f j
- b is the next root; gdh are in the left subtree; ei are in the right subtree.



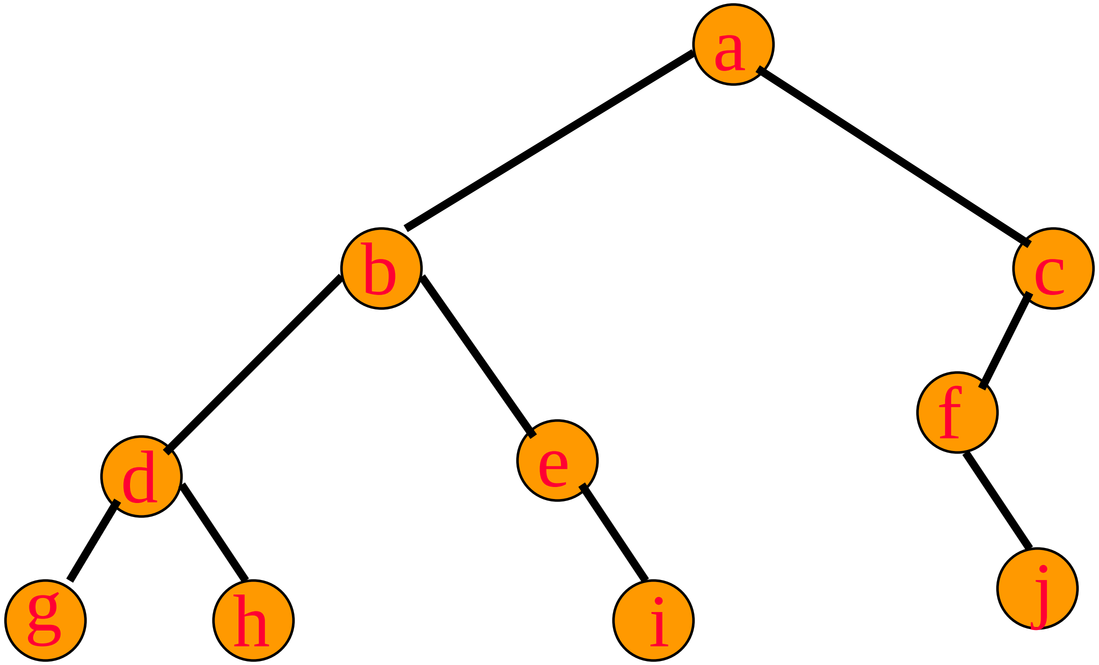
Inorder And Preorder



- 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 Preorder



a b c d e f g h i j

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.

中序遍历配合另外任何一个遍历，能重建二叉树。其他的任意两个序列的组合都不能唯一的确定重建的二叉树。

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 +

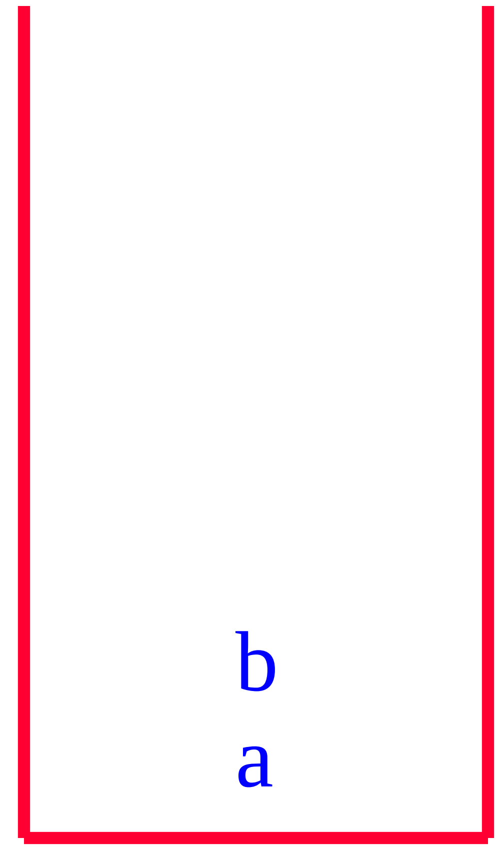
- Infix = (a + b) * (c – d) / (e + f)
  - Postfix = a b + c d - * e f + /

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

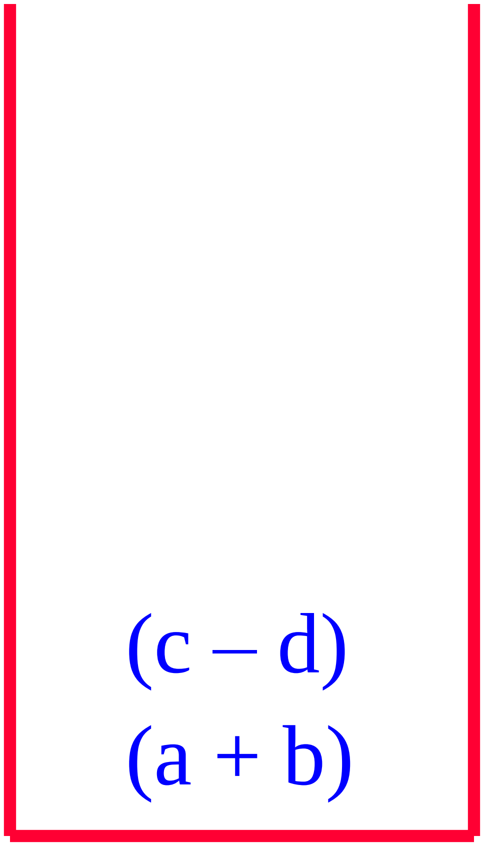
c

(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

Huffman Codes

- Suppose our text is a string that comprises the characters *a*, *u*, *x* and *z*.
- If the length of this string is 1000, then storing it as 1000 one-byte characters will take 1000 bytes (or 8000 bits) of space.
- If we encode the symbols in the string using 2 bits per symbol (00=*a*, 01=*b*, 10=*u*, 11=*z*), then the 1000 symbols can be represented with 2000 bits of space.

Huffman Codes

- In the string *aaxuaxz*, the *a* occurs three times. The number of occurrences of a symbol is called its frequency.
- The frequency of *a*, *x*, *u*, and *z* in the sample string are 3, 2, 1, and 1, respectively.
- If we use the codes (0 = *a*, 10 = *x*, 110 = *u*, 111 = *z*), the encoded version of *aaxuaxz* is 0010110010111. The length of this encoded version is 13 bits compared to 14 bits using the 2 bits per symbol code!

Huffman Codes

例　　要传输的原文为**ABACCDA**

等长编码　　**A：00　　B：01　　　C：10　　D：11**

发送方：将**ABACCDA**　转换成　**00010010101100**

接收方：将 **00010010101100**　还原为 **ABACCDA**

不等长编码　**A：0　　B：00　　　C：1　　D：01**

发送方：将**ABACCDA**　转换成　**000011010**

接收方：**000011010**　　转换成

AAAACCDA

BBCCDA

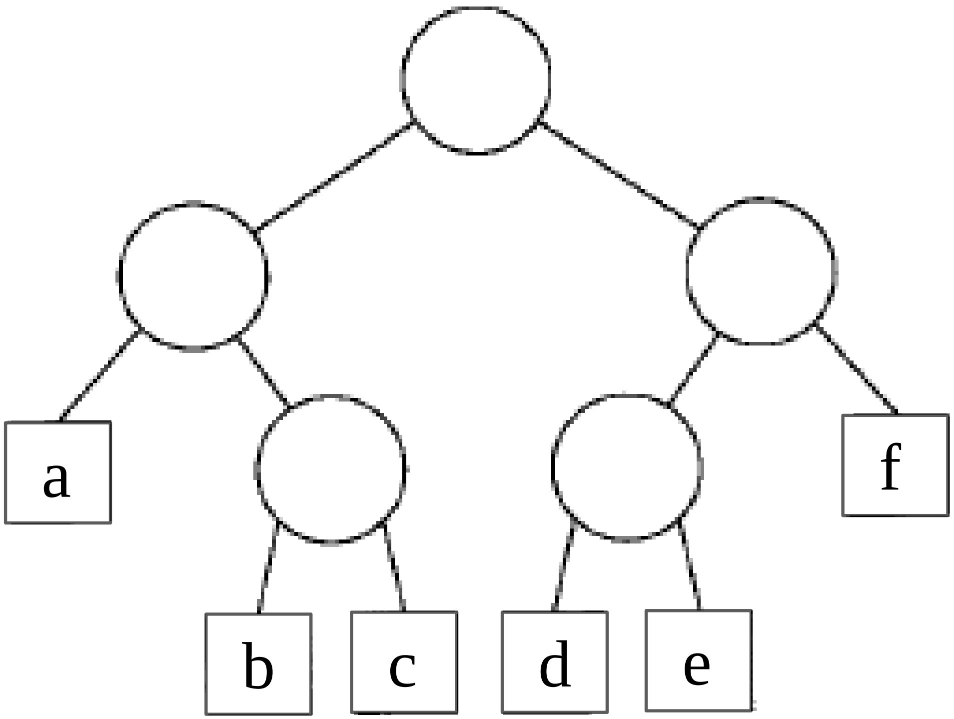
A的编码是B的前缀

设　**A：0　　B：110　　　C：10　　D：111**

发送方：将**ABACCDA**　转换成　**0110010101110**　总长度是**13**

所得的译码是唯一的

Huffman Codes



The root to the external node paths in an extended binary tree may be coded using 0 to representet a move to a left subtree and 1 to a move to a right subtree.

Huffman Codes

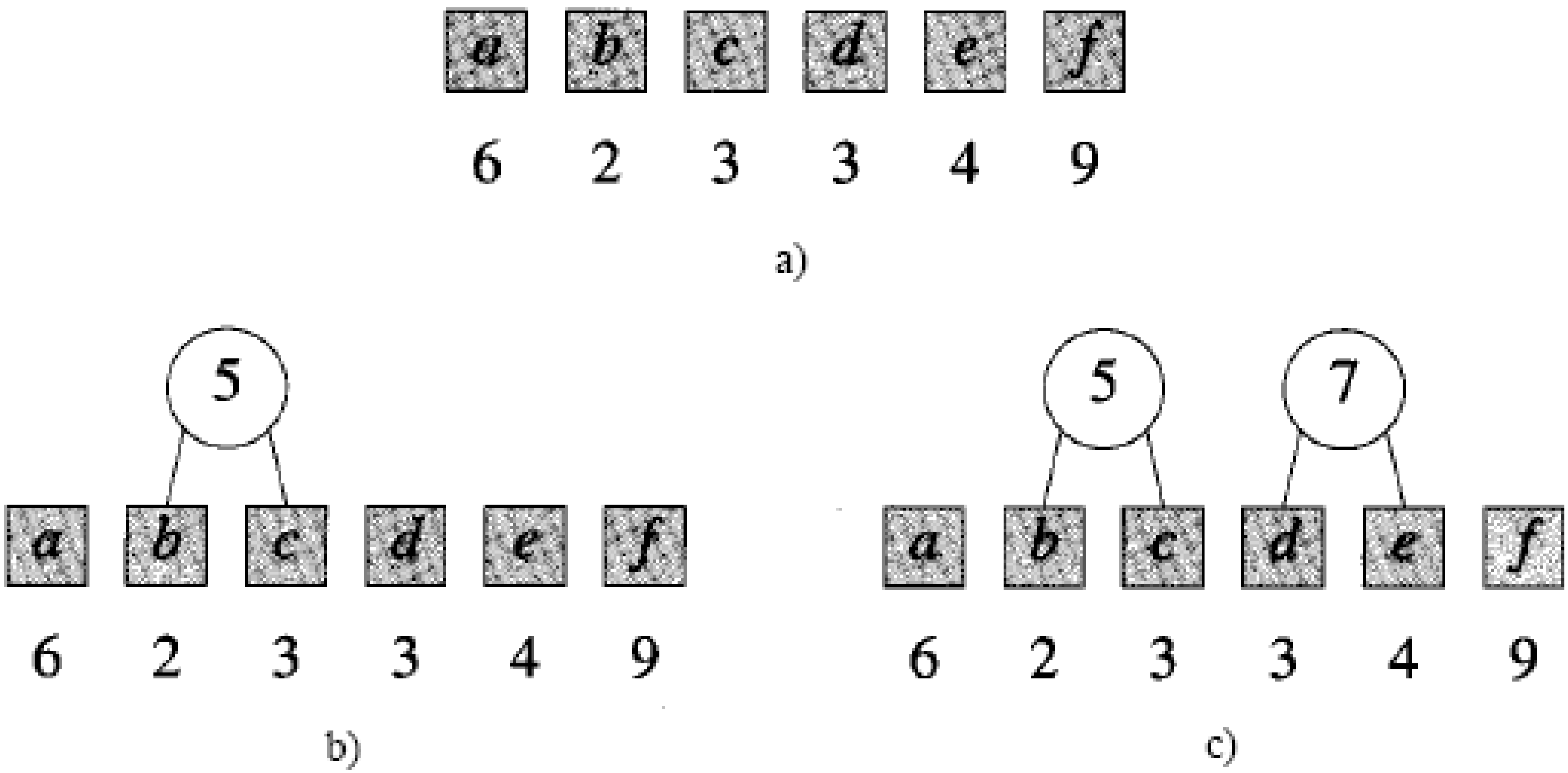
Let S be a string made up of these symbols, and let $F(x)$ be the frequency of the symbol $x \in \{a, b, c, d, e, f\}$. If S is encoded using these codes, the encoded string has a length

$$2 \times F(a) + 3 \times F(b) + 3 \times F(c) + 3 \times F(d) + 3 \times F(e) + 2 \times F(f)$$

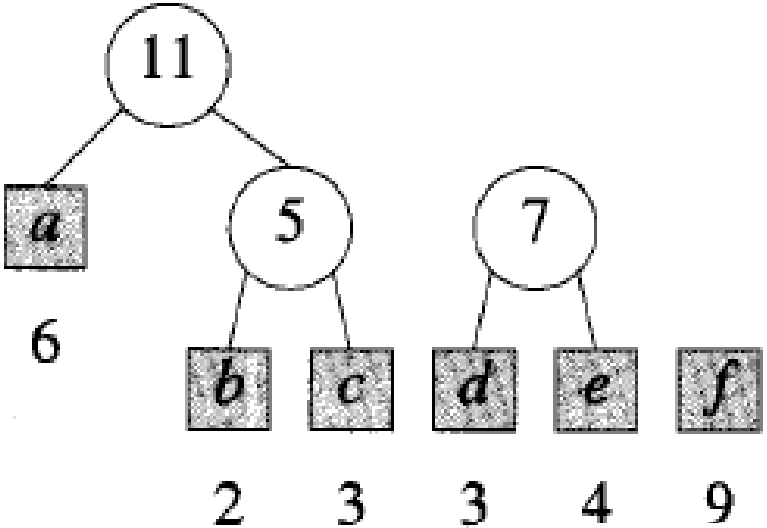
For an extended binary tree with external nodes labeled 1, …, n, the length of the encoded string is

$$WEB = \sum_{i=1}^{n} L(i) \times F(i)$$

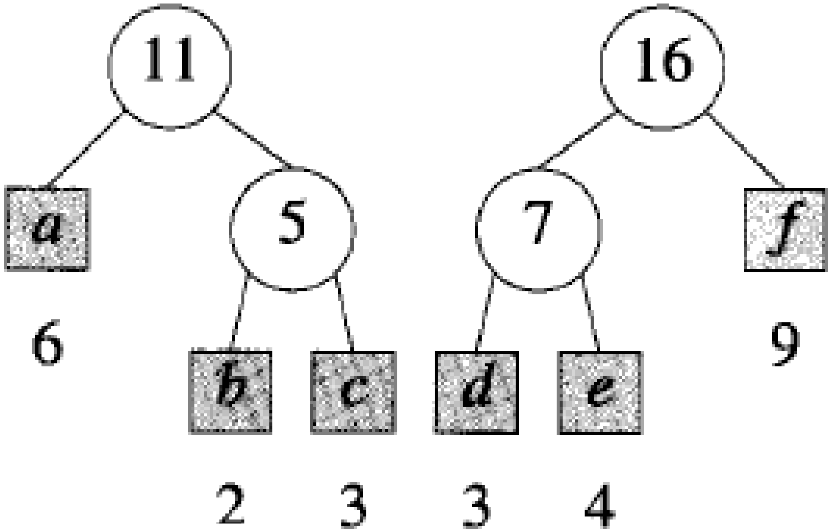
Huffman Codes



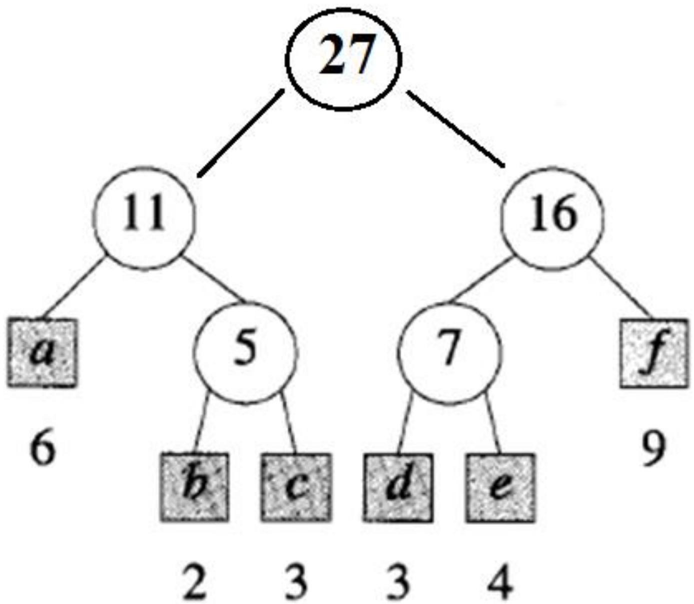
Huffman Codes



d)



e)



f)