DS286 | 2016-09-23,26

L13,14: Trees, Binary Trees, Expression Trees

Yogesh Simmhan

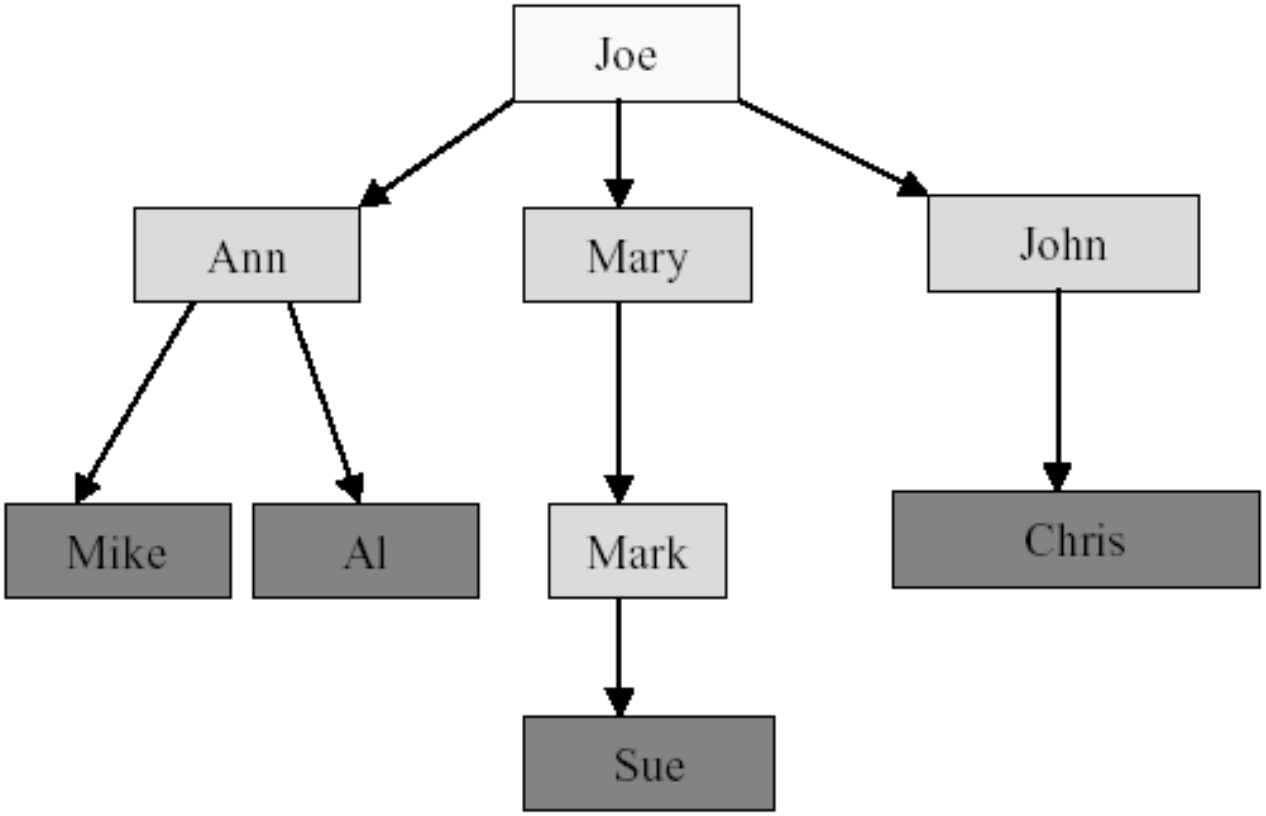
simmhan@cds.iisc.ac.in

Slides courtesy Venkatesh Babu, CDS & Sahni textbook

CDS
The Department of Computational and Data Science

Linear Lists and Trees

- Linear lists are useful for serially ordered data
  - $(e_1, e_2, e_3, \ldots, e_n)$
  - Days of week
  - Months in a year
  - Students in a class
- Trees are useful for hierarchically ordered data
  - Joe's descendants
  - Corporate structure
  - Government Subdivisions
  - Software structure



What are other examples of hierarchically ordered data?

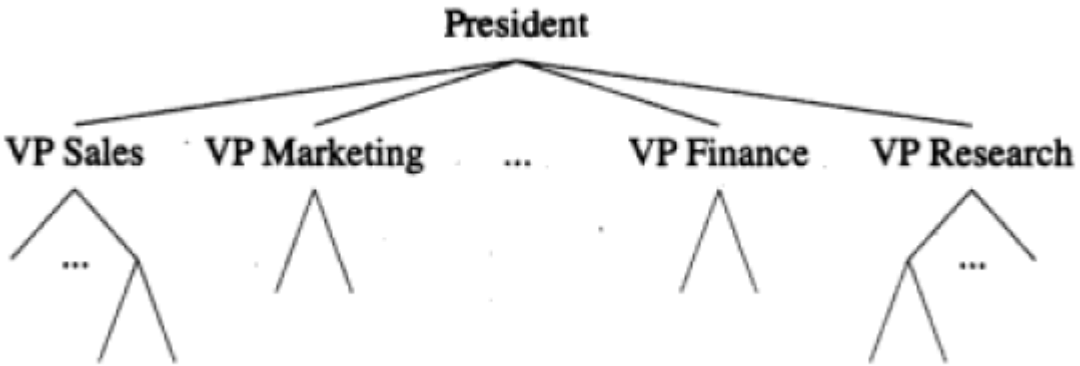


Figure 11.2 Hierarchical administrative structure of a corporation

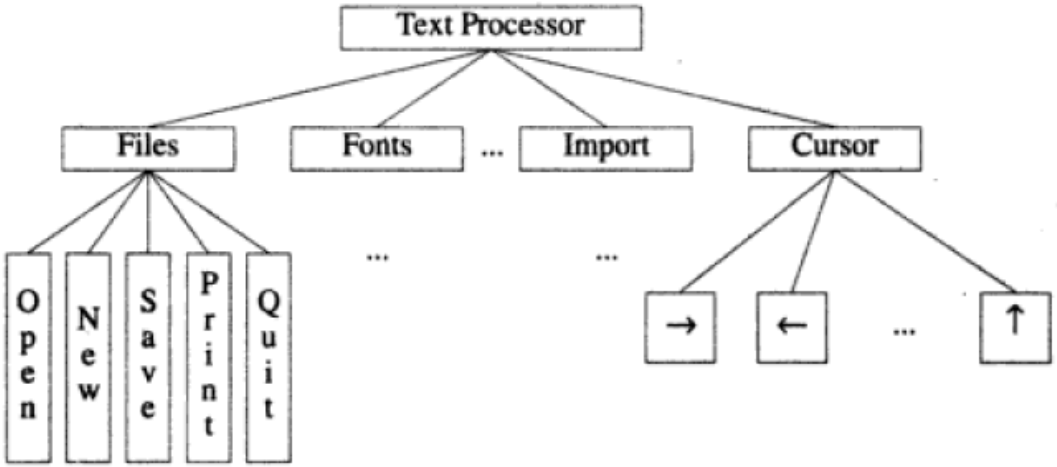
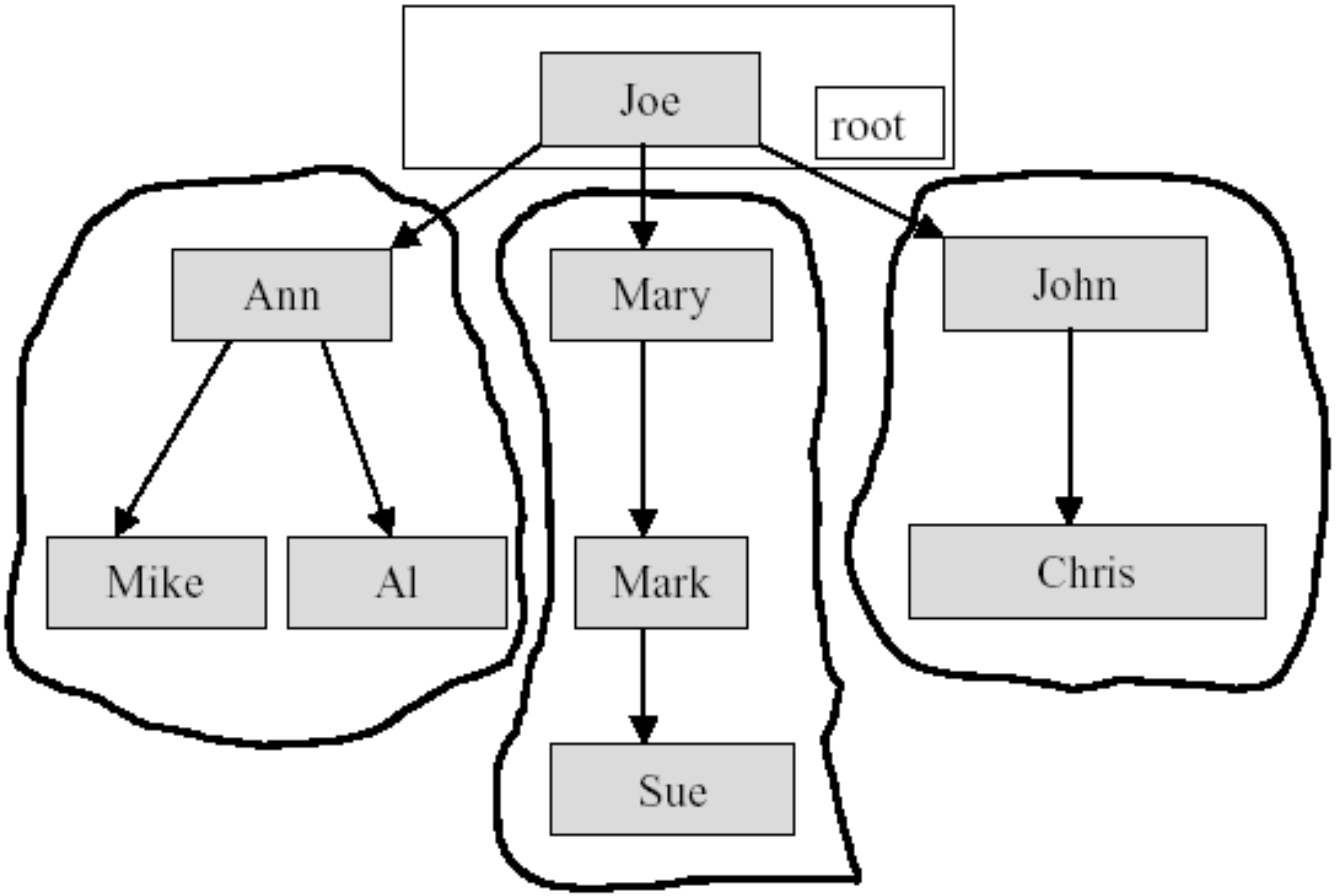


Figure 11.4 Module hierarchy for text processor

Definition of Tree

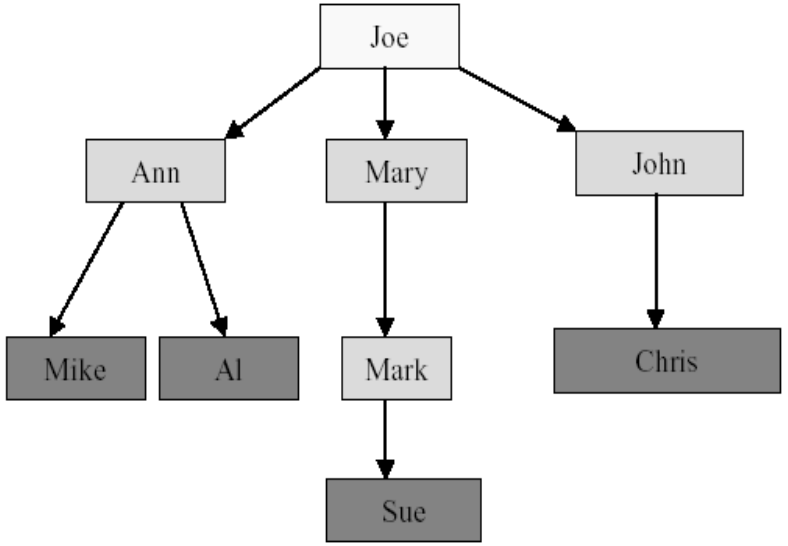
- A tree *t* is a finite non-empty set of elements
- One of these elements is called the root
- The remaining elements, if any, are partitioned into trees, which are called the subtrees of *t*.

Subtrees



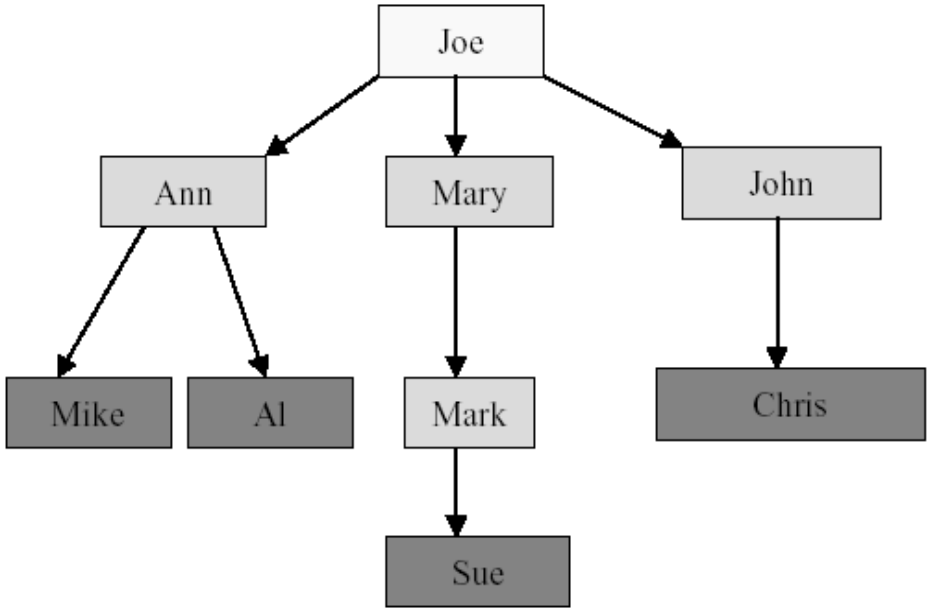
Tree Terminology

- 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 roo and so on.
- Elements at the lowest level of the hierarchy are the **leaves**.



Other Definitions

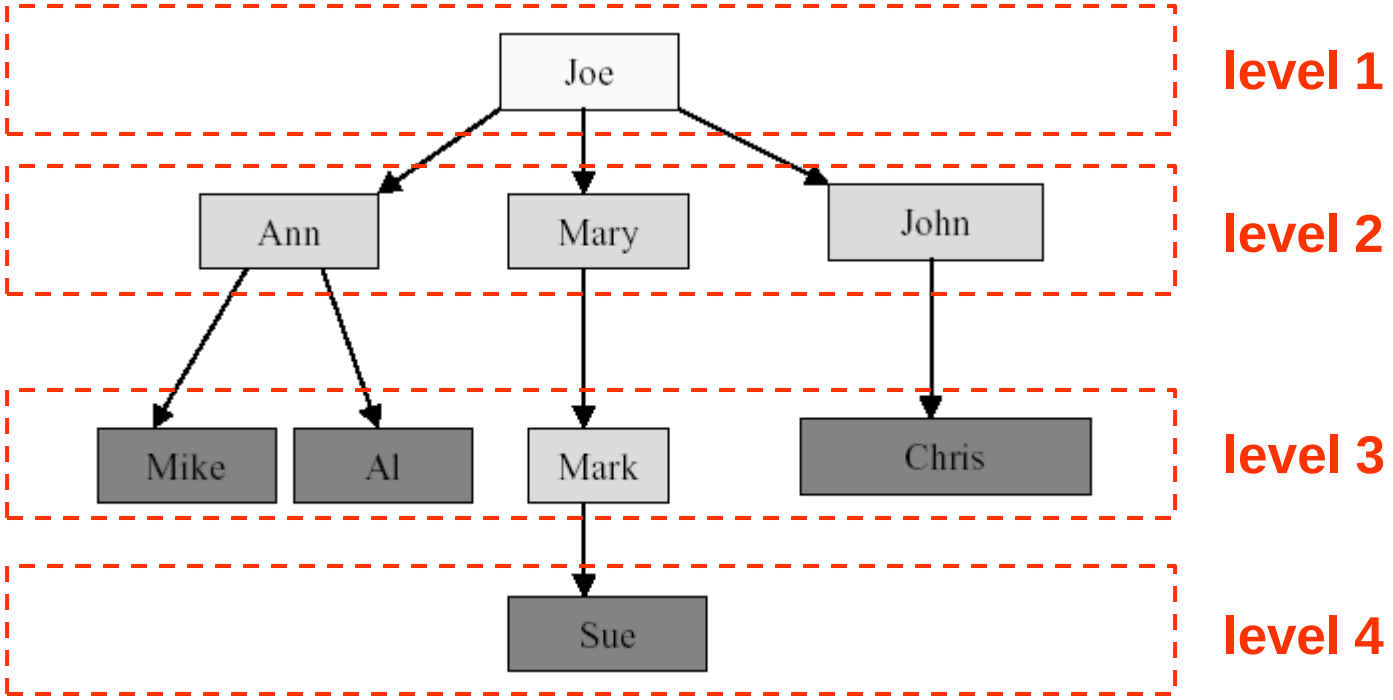
- Leaves, Parent, Grandparent, Siblings, Ancestors, Descendents




```
Leaves = {Mike,AI,Sue,Chris}

Parent(Mary) = Joe

Grandparent(Sue) = Mary

Siblings(Mary) = {Ann,John}

Ancestors(Mike) = {Ann,Joe}

Descendents(Mary)={Mark,Sue}
```

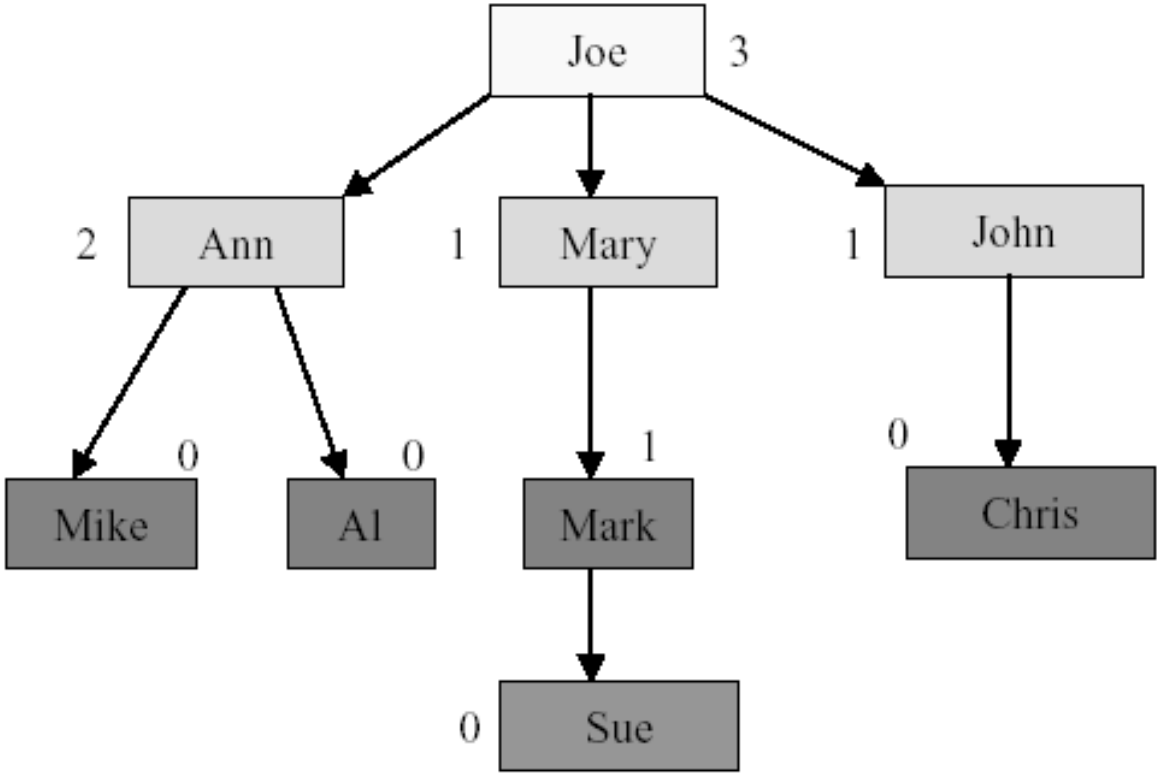
Levels and Height

- Root is at level 1 and its children are at level 2.
- Height = depth = number of levels



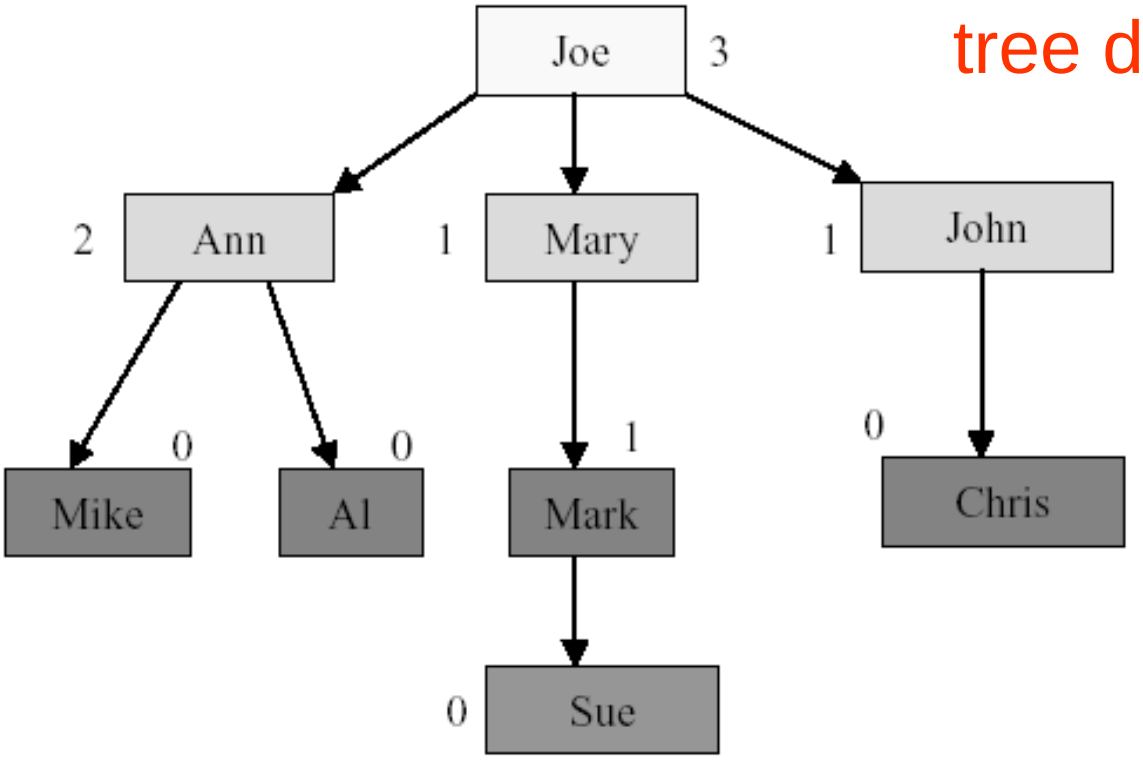
Node Degree

- Node degree is the number of children it has



Tree Degree

- Tree degree is the maximum of node degrees



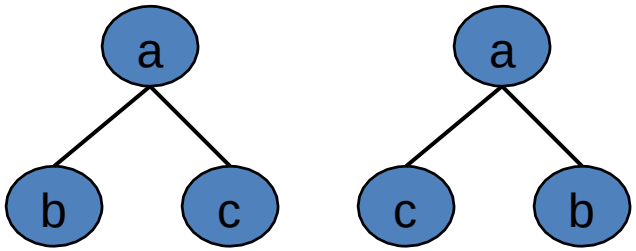
Binary Trees

Binary Tree

- A finite (possibly empty) collection of elements
- A non-empty binary tree has a root element and the remaining elements (if any) are partitioned into two binary trees
- They are called the left and right sub-trees of the binary tree

Tree vs. Binary Tree

- A binary tree may be empty; a tree cannot be empty.
- 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.
- The subtrees of a binary tree are ordered; those of a tree are not ordered.



- **different when viewed as a binary tree**
- **same when viewed as a tree**

Binary Tree for Expressions

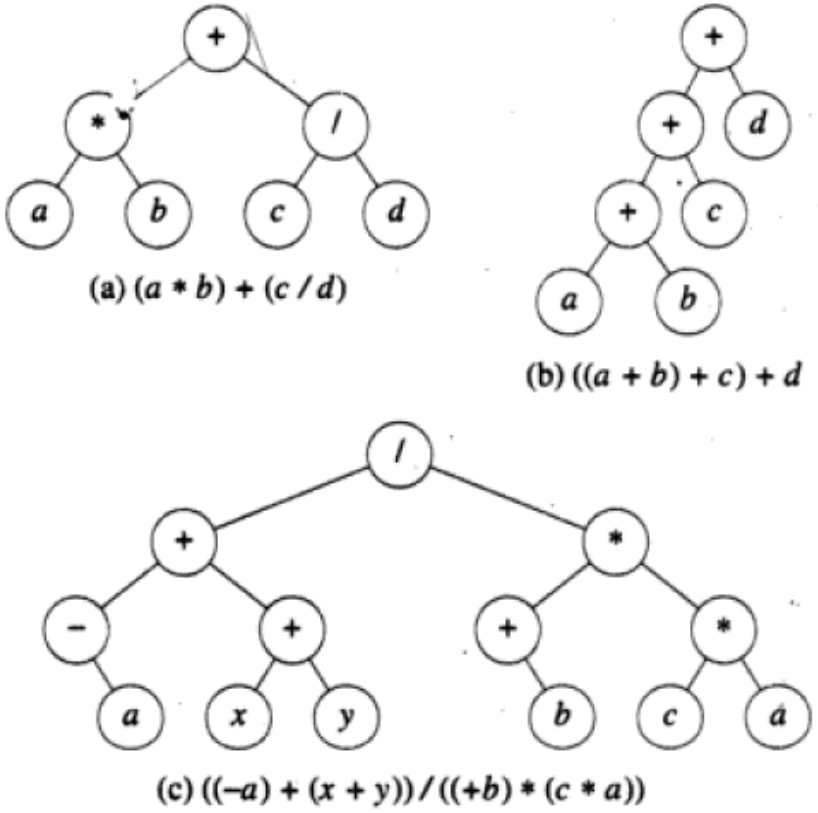


Figure 11.5 Expression trees

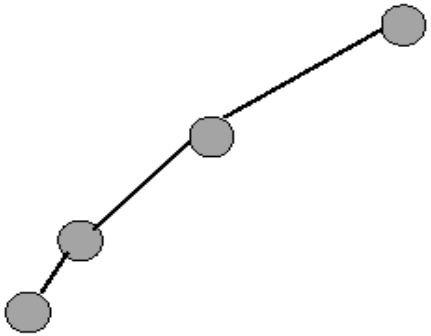
Binary Tree Properties

1. The drawing of every binary tree with n elements, $n > 0$, has exactly $n-1$ edges.
   - Each node has exactly 1 parent (except root)
2. A binary tree of height h, $h >= 0$, has at least h and at most 2^h-1 elements in it.
   - At least 1 element at each level → #elements = h
   - At most 2^{i-1} elements at i-th level → $\Sigma\ 2^{i-1} = 2^h - 1$

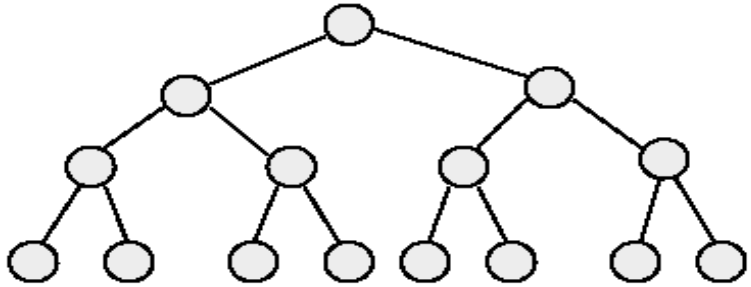
$$a+ar^1+ar^2+\ldots+ ar^{n-1} = a(r^n-1)/(r-1)$$

Binary Tree Properties

3. The height of a binary tree that contains *n* elements, *n* >= 0, is at least $\lceil (log_2(n+1)) \rceil$ and at most *n*.
   - At least one element at each level → h_{max} = #elements (n)
   - From prev: h_{min} = ceil(log(n+1))



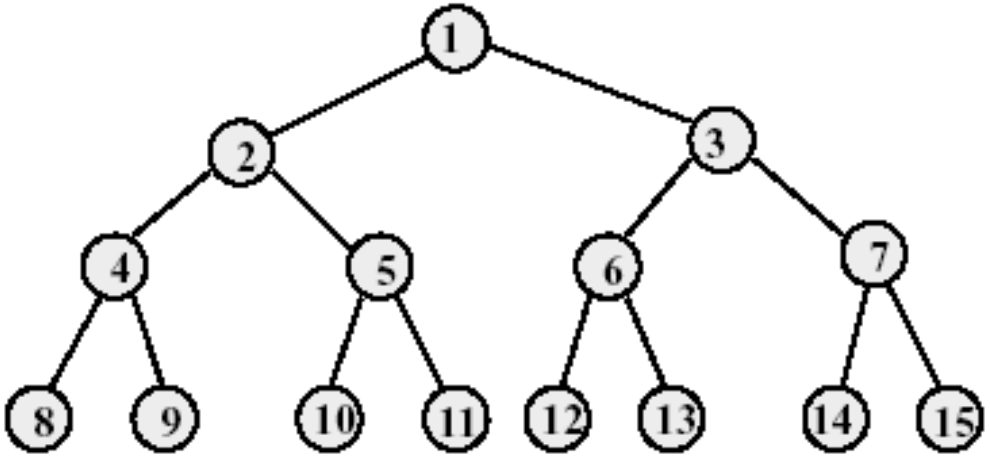
minimum number of elements



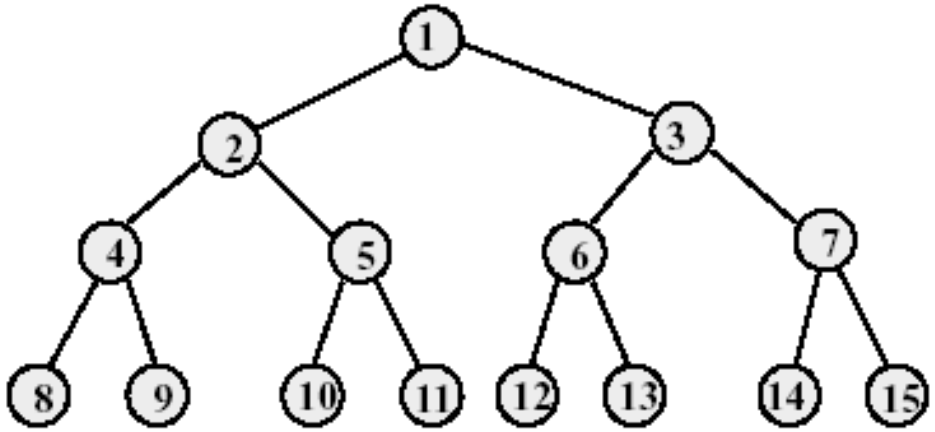
maximum number of elements

Full Binary Tree

- A full binary tree of height h has exactly 2^h-1 nodes
- Numbering the 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

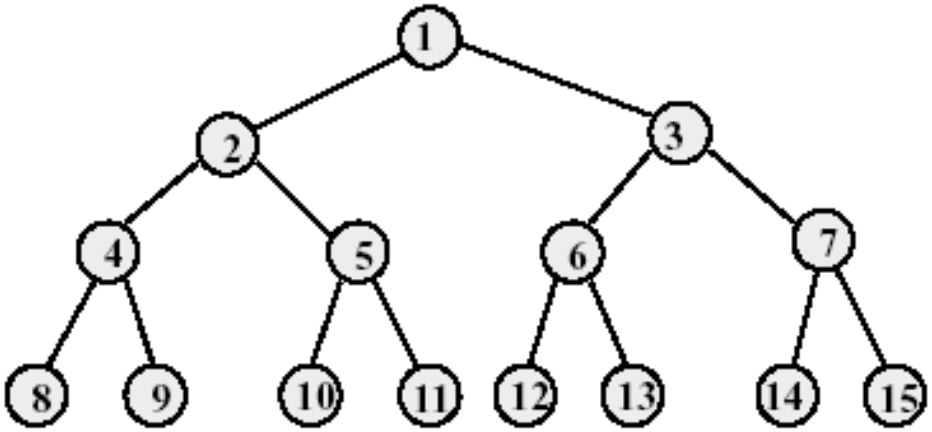


Node Number of Full Binary Tree



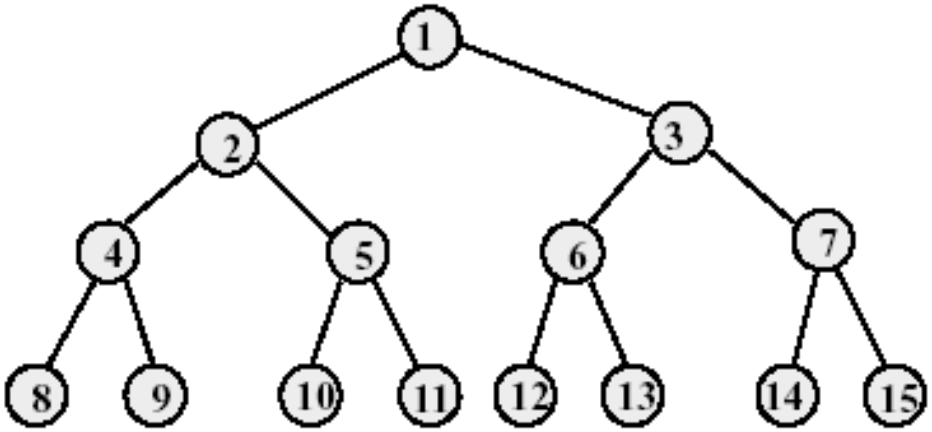
- Parent of node *i* is node $\lfloor (i/2) \rfloor$, unless i = 1
- Node 1 is the root and has no parent

Node Number of Full Binary Tree



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

Node Number of Full Binary Tree

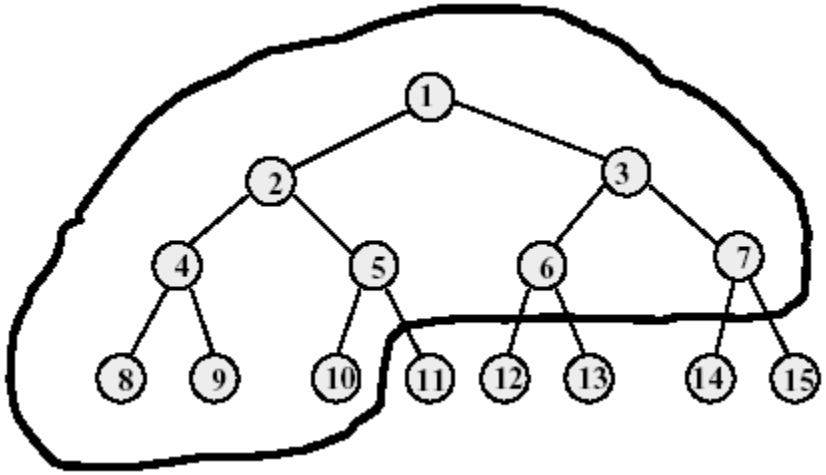


- Right child of node *i* is node *2i*+1
- If *2i+1 > n*, node *i* has no right child.

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 n-node complete binary tree
- A full binary tree is a special case of a complete binary tree

Complete Binary Tree



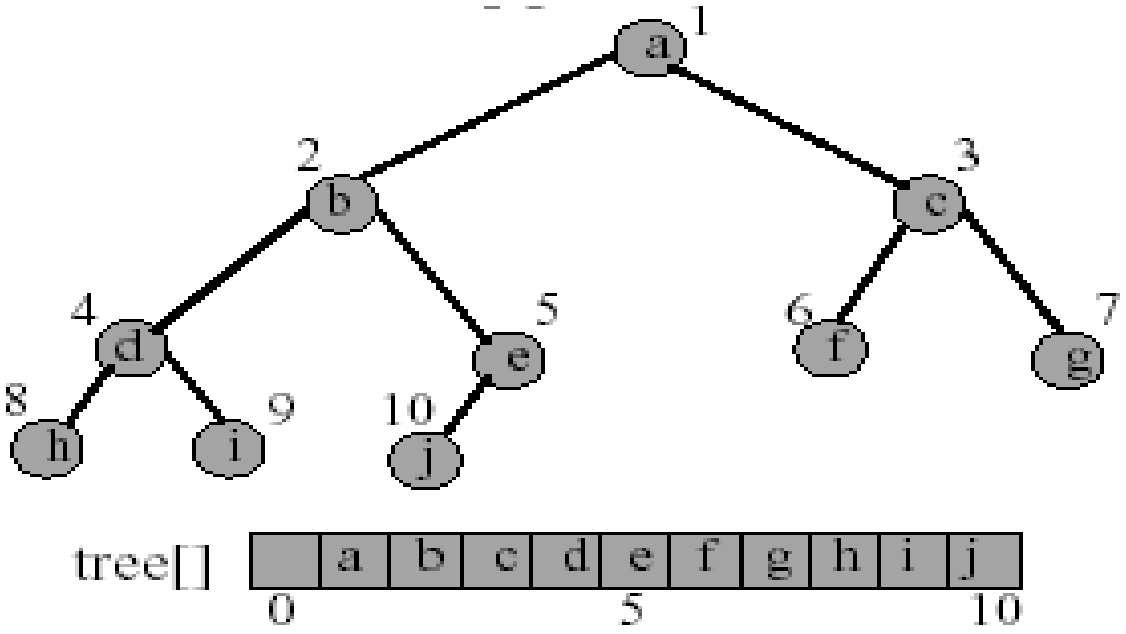
- Complete binary tree with 10 nodes.
- Same node number properties (as in full binary tree) also hold here.

Binary Tree Representation

- Array representation
- Linked representation

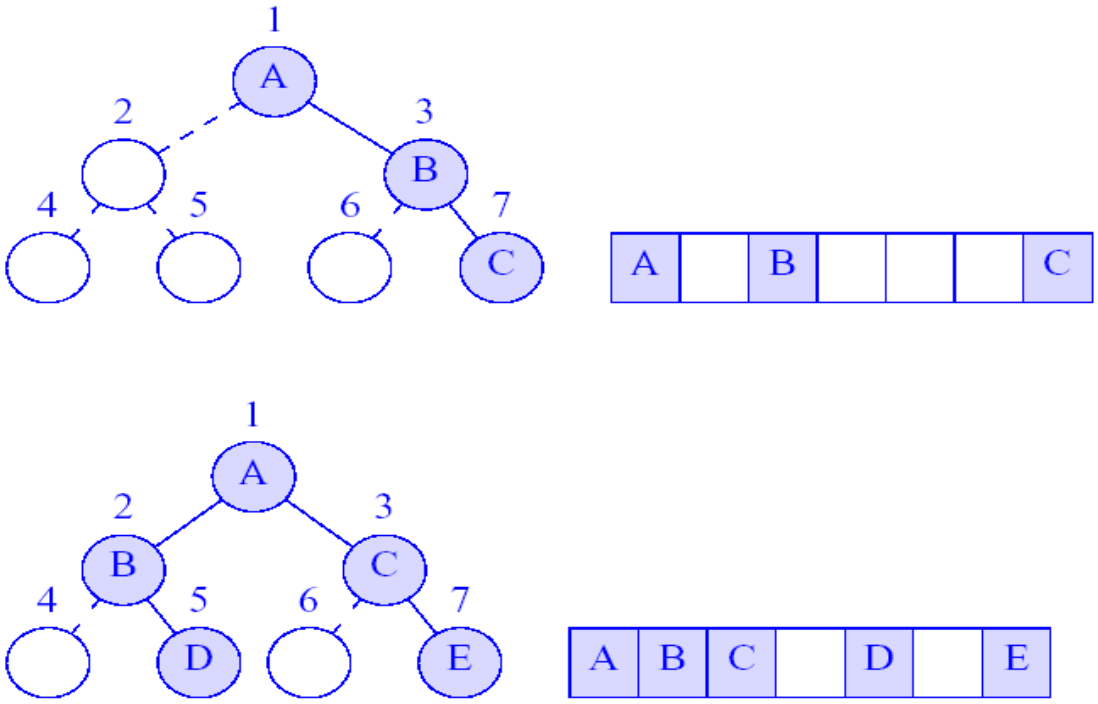
Array Representation

- The binary tree is represented in an array by storing each element at the array position corresponding to the number assigned to it.

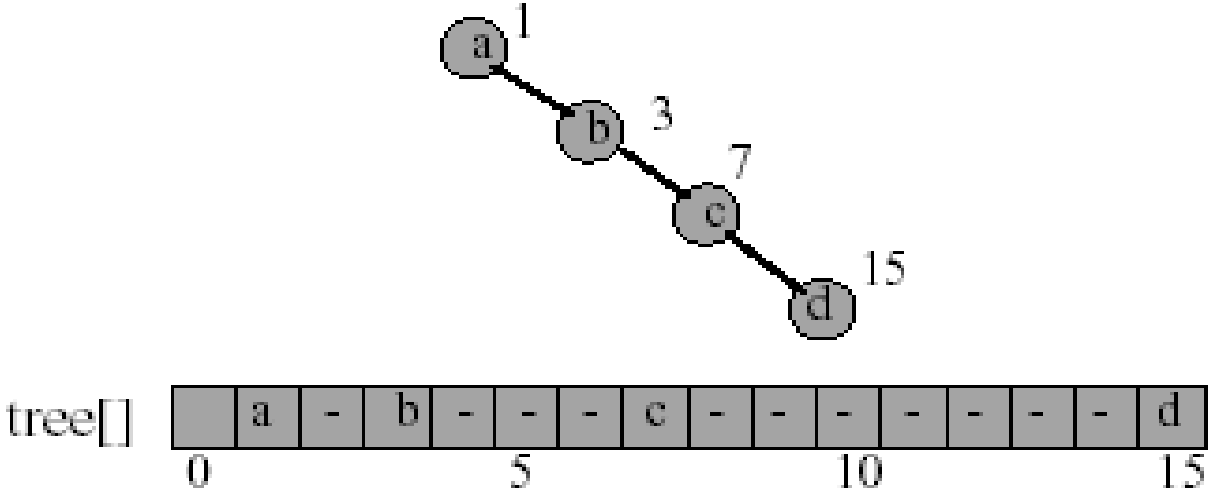


Incomplete Binary Trees

Complete binary tree with some missing elements



Right-Skewed Binary Tree

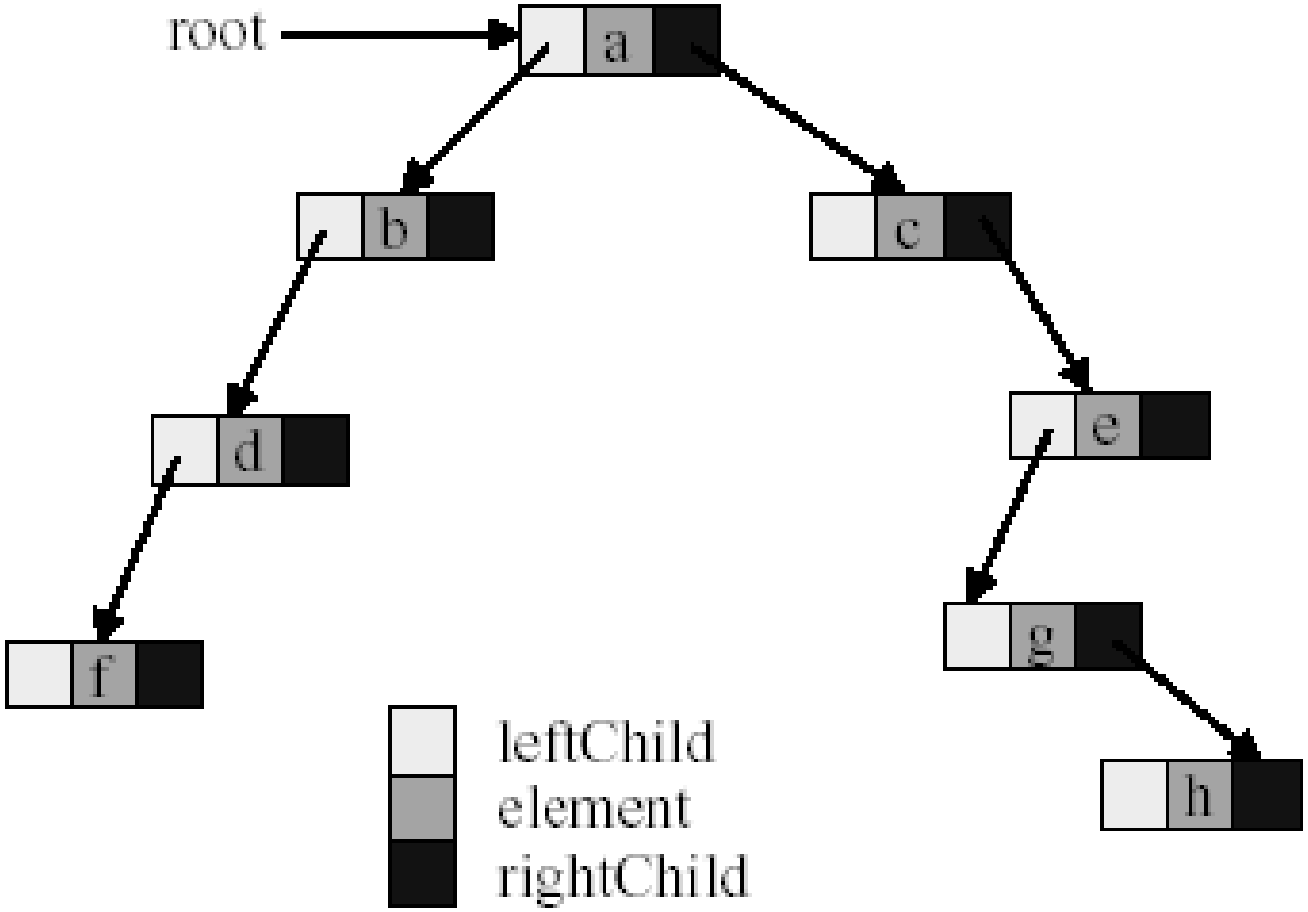


- An *n* node binary tree needs an array whose length is between ***n+1*** and $\boldsymbol{2^n}$.
- Right-skewed binary tree wastes the most space
- What about left-skewed binary tree?
  - *Equally bad, though with trailing blanks that could be trimmed if known ahead*

Linked Representation

- The most popular way to present a binary tree
- Each element is represented by a node that has two link fields (leftChild and rightChild) plus an item field
- Each binary tree node is represented as an object whose data type is BinTreeNode
- The space required by an *n* node binary tree is *n*`*sizeof(BinTreeNode)`

Linked Representation



Node Class For Linked Binary Tree

```
class BinTreeNode {
   int item;
   BinTreeNode *left, *right;

   BinTreeNode() {
      left = right = NULL;
   }
}
```

Common Binary Tree Operations

- Determine the height
- Determine the number of nodes
- Make a copy
- Determine if two binary trees are identical
- Display the binary tree
- Delete a tree
- If it is an expression tree, evaluate the expression
- If it is an expression tree, obtain the parenthesized form of the 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 **actions** *(make a copy, display, evaluate the operator, etc.)* with respect to this element are taken

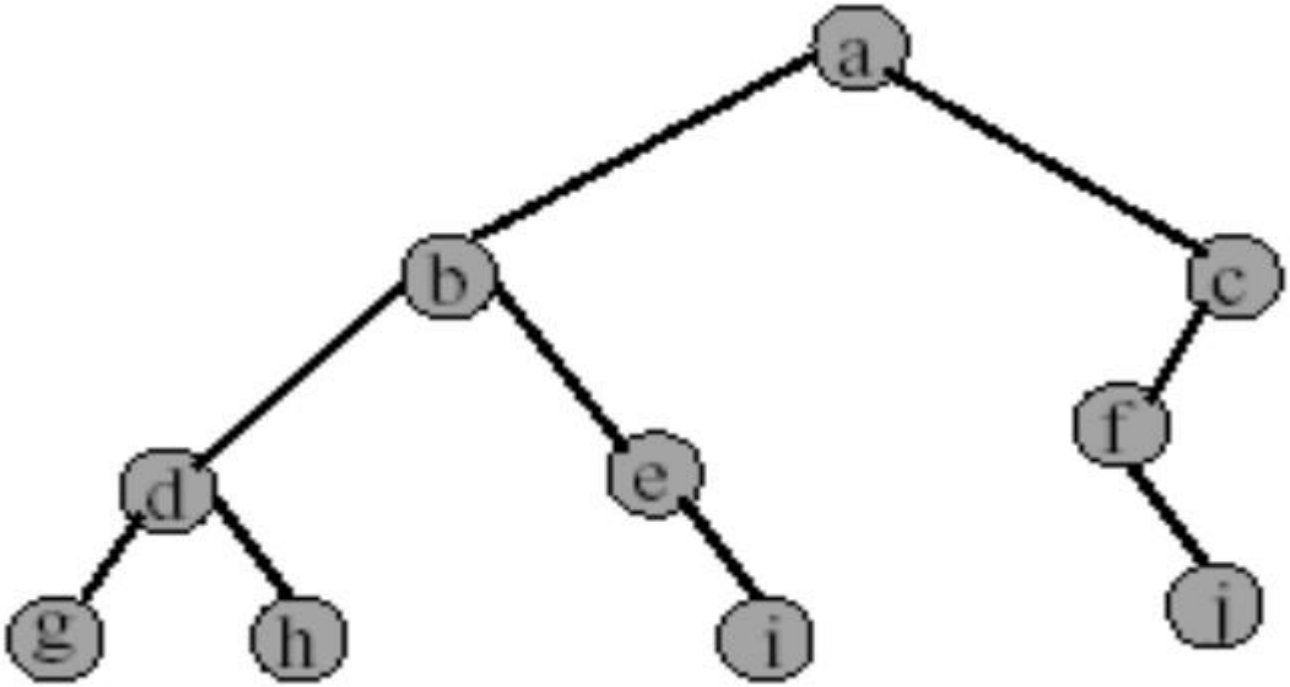
Binary Tree Traversal Methods

- Preorder
  - The **root** of the subtree is processed **first** before going into the **left then right subtree** (root, left, right)
- Inorder
  - After the complete processing of **the left subtree first** the **root** is processed followed by the processing of the complete **right subtree** (left, root, right)
- Postorder
  - The **left and right subtree** are completely processed, before the **root** is processed (left, right, root)
- Level order
  - The tree is processed one level at a time
  - First all nodes in level *i* are processed from left to right
  - Then first node of level *i+1* is visited, and rest of level *i+1* processed

Preorder Traversal

```
void preOrder(BinTreeNode *t) {
  if (t != NULL) {
    visit(t);                // Visit root 1st
    preOrder(t->left);  // Left Subtree
    preOrder(t->right); // Right Subtree
  }
}
```

Preorder Example
(visit action = print)

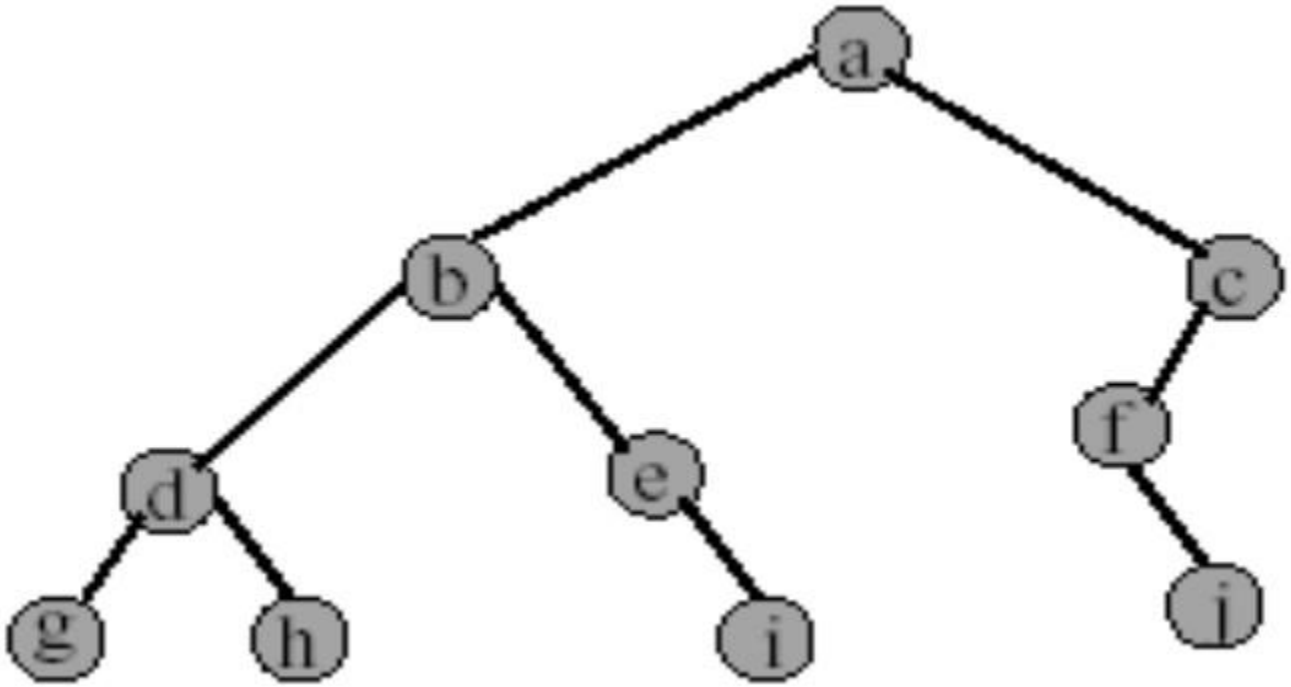


a b d g h e i c f j

Inorder Traversal

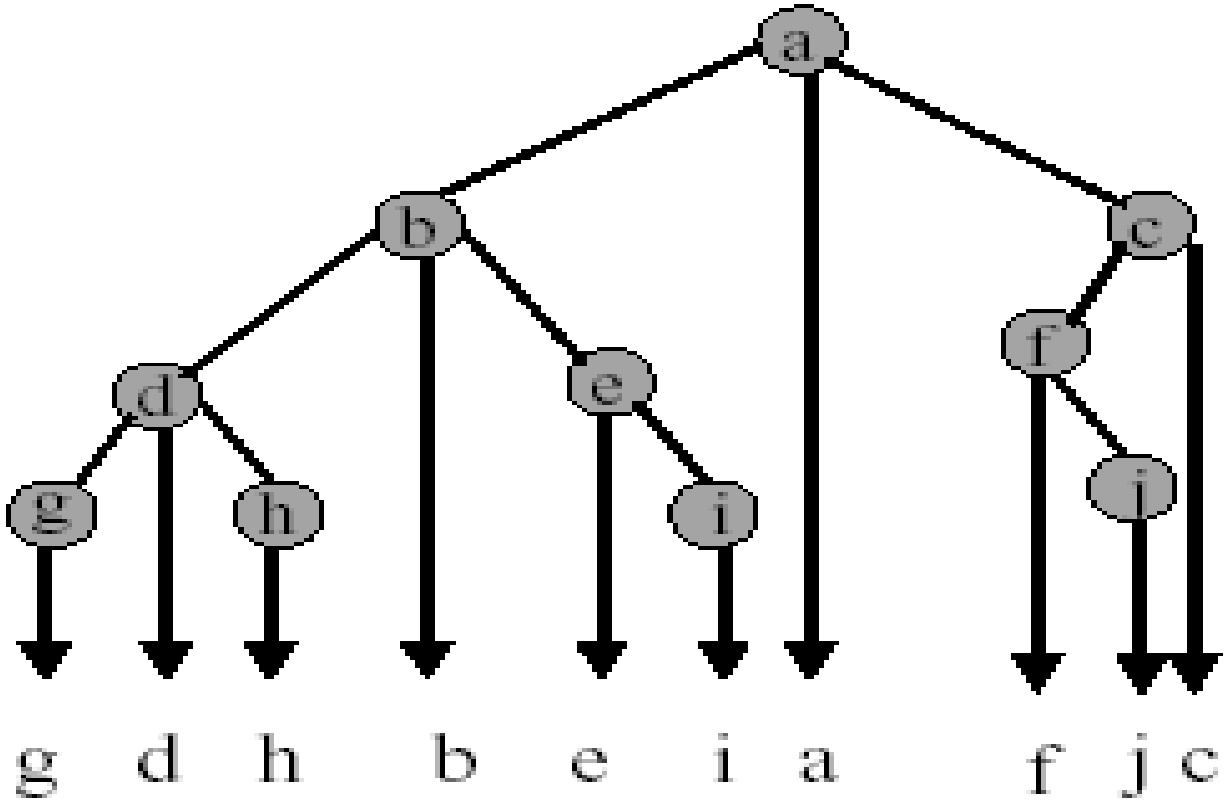
```
void inOrder(BinTreeNode *t) {
  if (t != NULL) {
    inOrder(t->left);   // Left Subtree 1st
    visit(t);           // Visit root
    inOrder(t->right);  // Right Subtree last
  }
}
```

Inorder example



g d h b e i a f j c

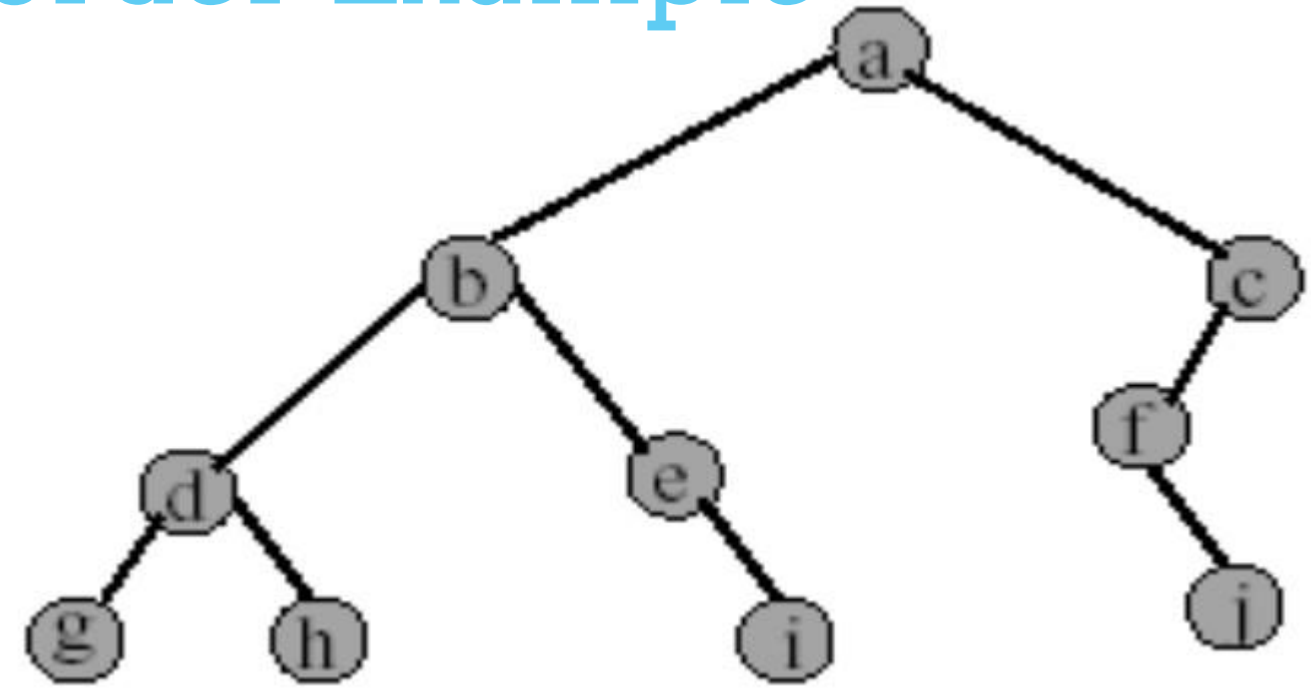
Inorder by Projection (Squishing)



Postorder Traversal

```
void postOrder(BinTreeNode *t) {
  if (t != NULL) {
    postOrder(t->left); // Left Subtree 1st
    postOrder(t->right);// Right Subtree
    visit(t);           // Visit root last
  }
}
```

Postorder Example

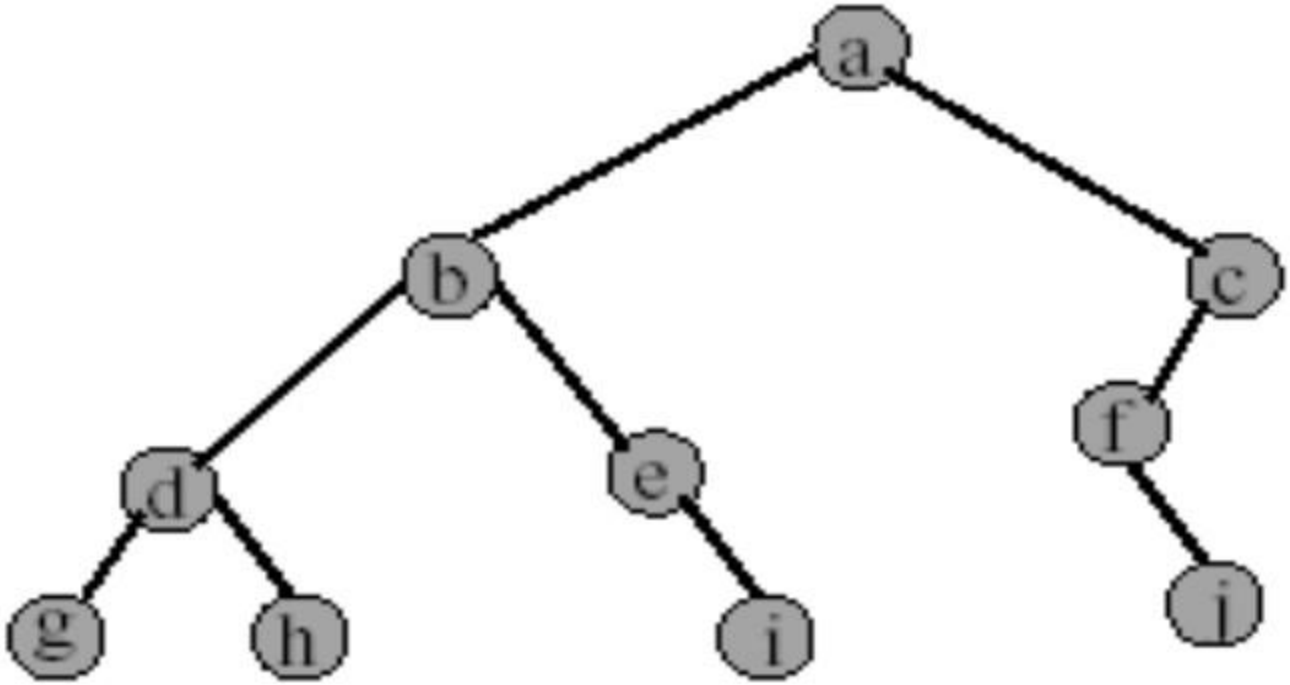


g h d i e b j f c a

Level Order Traversal

```
void levelOrder(BinTreeNode *t){
  Queue<BinTreeNode*> q;
  while (t != NULL) {
    visit(t); // visit t
    // push children to queue
    if (t->left) q.push(t->left);
    if (t->right) q.push(t->right);
    t = q.pop();   // next node to visit
  }
}
```

Level Order Example



- Add and delete nodes from a queue
- Output: a b c d e f g h i j

Space and Time Complexity

- The space complexity of each of the four traversal algorithms is **O(n)**
  - Why not Θ(n)? Size of recursion stack/level queue is variable.
- The time complexity of each of the four traversal algorithm is **Θ(n)**
  - Each node visited only one

Expression 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
  - B O DM AS: Brackets, Order of powers, Division, Multiplication, Addition, Subtraction
  - `( = ) > ^ > * = / > + = -`
- When an operand lies between two operators, the operand associates with the operator with higher priority

Tie Breaker

- When an operand lies between two operators with same priority, the operand associates with the operator on the left
  - `a+b-c` ➲ `(a+b)-c`
  - `a*b/c/d` ➲ `((a*b)/c)/d`

Delimiters

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

Infix Expression Hard To Parse

- Need operator priorities, tie breaker, and delimiters
- Makes evaluation by program more difficult
- **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

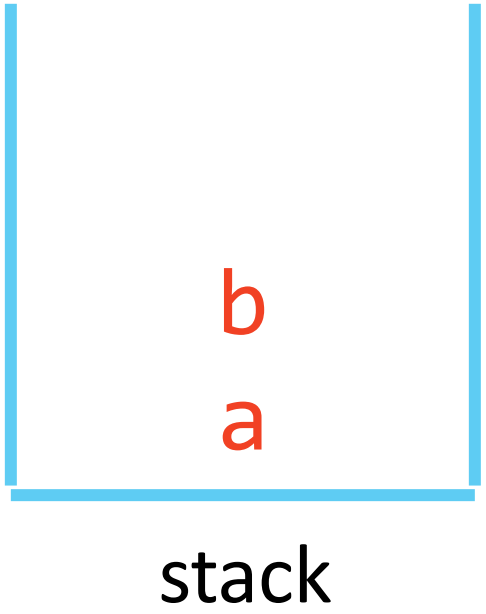
(a+b)*(c-d)/(e+f)

a b+ c d-* e f+/

a b+ c d-* e f+/

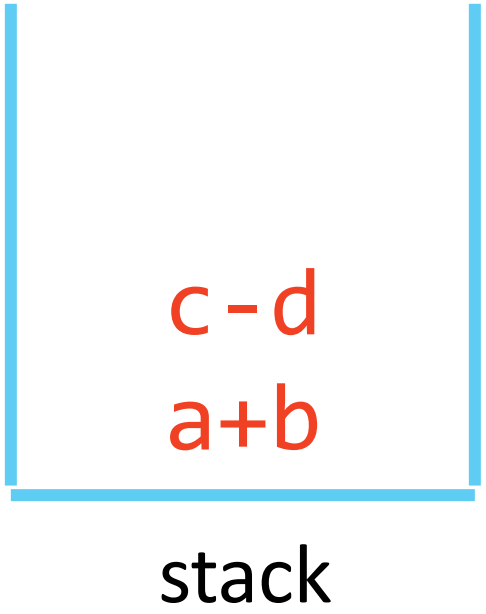
a b+ c d-* e f+/

a b+ c d-* e f+/



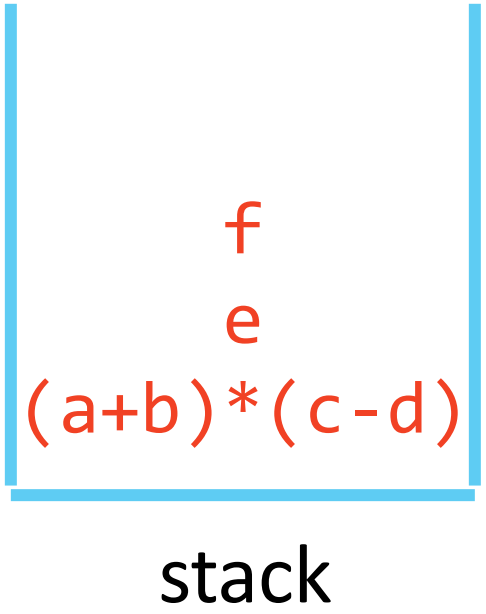
Postfix Evaluation

```
a b+ c d-* e f+/
a b+ c d-* e f+/
a b+ c d-* e f+/
a b+ c d-* e f+/
```



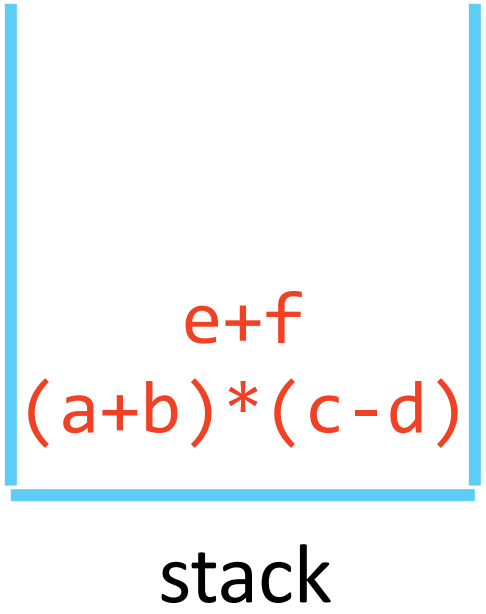
Postfix Evaluation

```
a b+ c d-* e f+/
a b+ c d-* e f+/
a b+ c d-* e f+/
a b+ c d-* e f+/
```



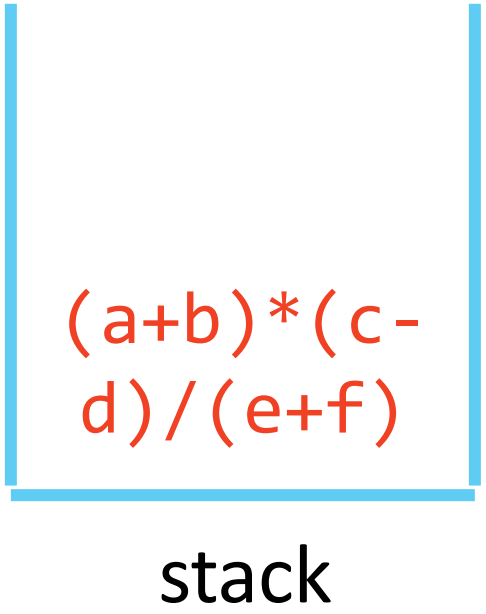
Postfix Evaluation

```
a b+ c d-* e f+/
a b+ c d-* e f+/
```



Postfix Evaluation

```
a b+ c d-* e f+/
a b+ c d-* e f+/
```

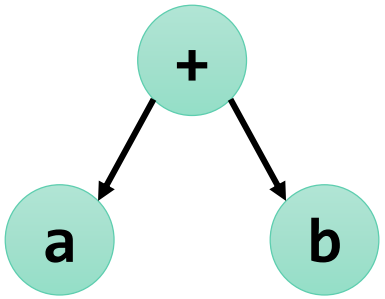


Prefix Form

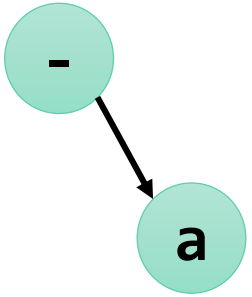
- The prefix form of a variable or constant is the same as its infix form
  - `a, b, 3.25`
- The relative order of operands is the same as in infix and prefix forms
- Operators come immediately before the prefix form of their operands.
  - Infix: `a + b`
  - Postfix: `ab+`
  - Prefix: `+ab`

Binary Tree Form

- a + b

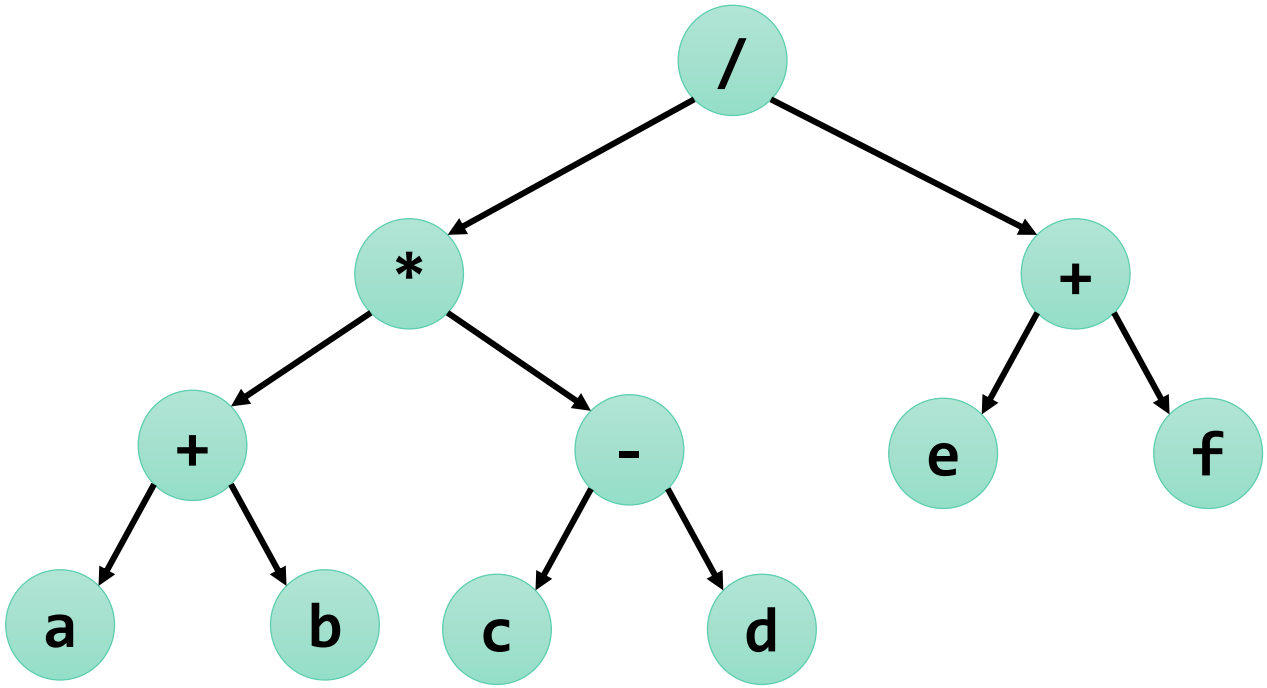


- - a



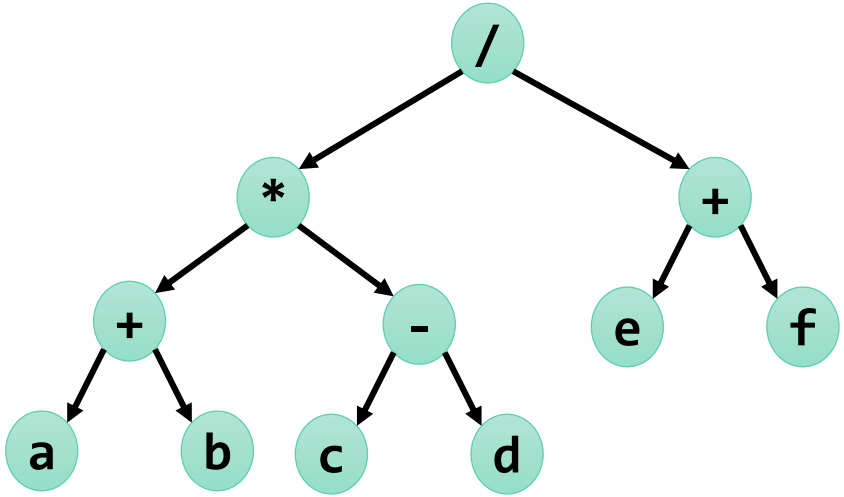
Binary Tree Form

- (a + b) * (c – d) / (e + f)

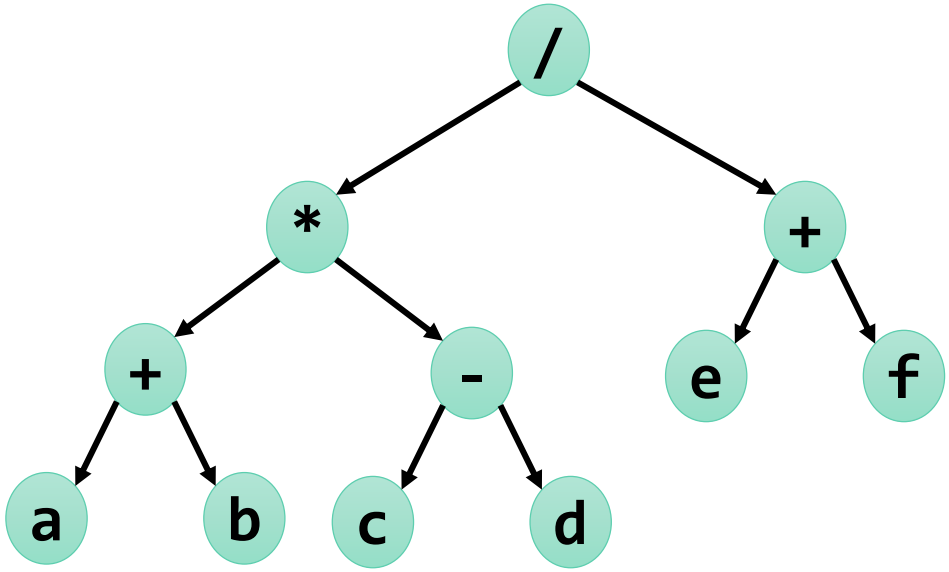


Merits Of Binary Tree Form

- 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



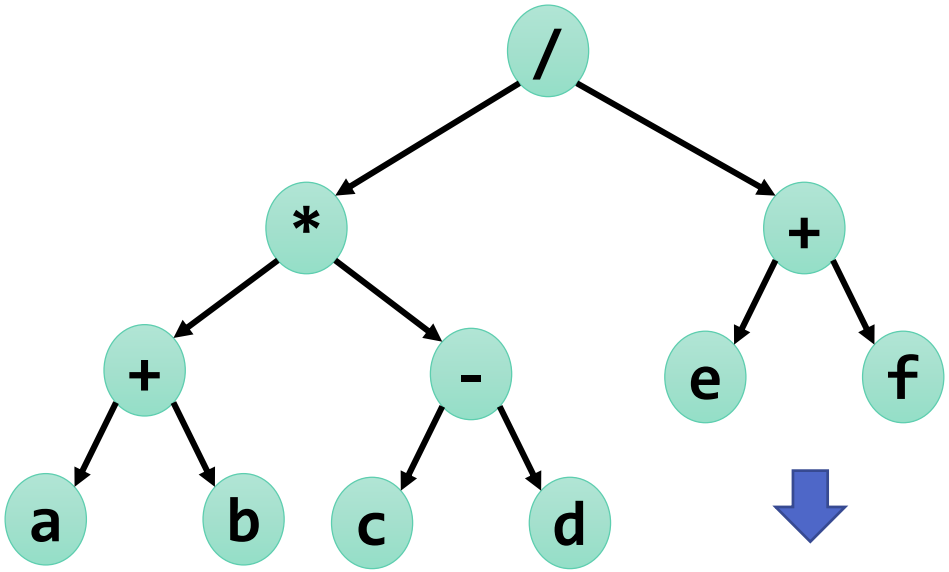
Preorder of Expression Tree



/ * + a b - c d + e f

Gives prefix form of expression.

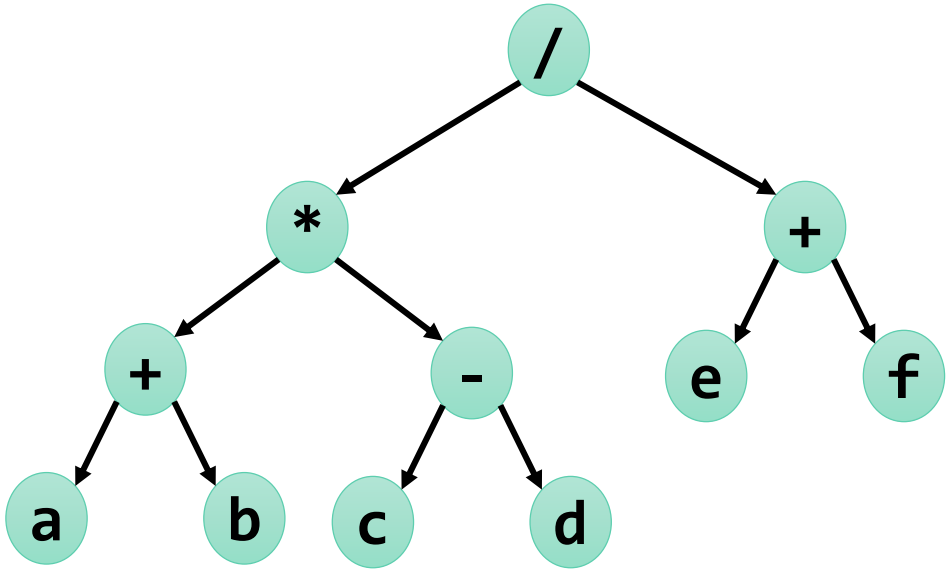
Inorder of Expression Tree



a + b * c – d /e + f

- Gives infix form of expression, which is how we normally write math expressions.
  - What about parentheses?
  - Fully parenthesized output of the above tree?

Postorder of Expression Tree



a b + c d - * e f + /

Gives postfix form of expression.

Tasks

- Self study (Sahni Textbook)
  - **Check**: Have you read Chapter 10.5 Hashtable?
  - **Read**: Chapter 11.0-11.6, Trees & Binary Trees from textbook
  - **Try:** Exercise 7, 9, 13, 15-28 from Chapter 11 of textbook
  - **Try**: In/Pre/Post-order Traversals using Stack, not recursion.
  - **Try**: Prefix expression evaluation
- Finish Assignment 3 by Wed Sep 28 *(75 points)*
  - Late submissions from Thu-Sun will entail **5 points penalty per day**
  - Submissions on or after Mon, Oct 3 will not be accepted
- 26 Sep (Mon) Class instead of tutorial
- 30 Sep (Fri) Institute holiday. We will have class at 10am.
- Move Midterm from Oct 5 to Oct 7
  - All lectures till Trees & Searching will be in syllabus

Questions?