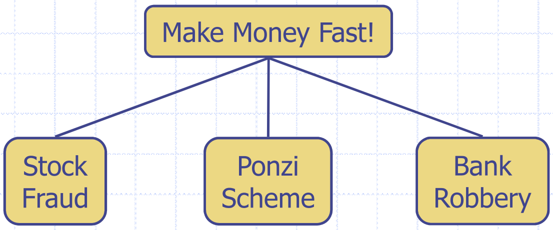
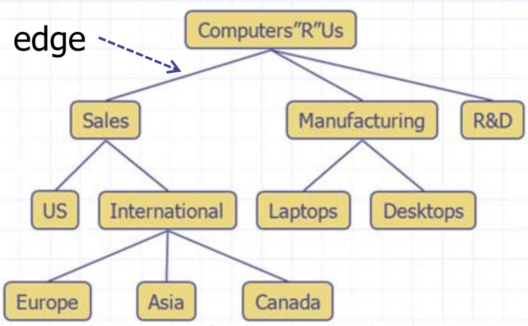
Trees

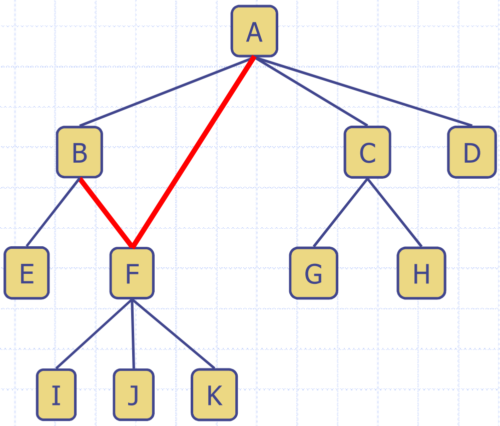


What is a Tree

- A tree is an abstract model of a hierarchical structure
- A tree T is a collection of nodes with nonlinear structure, called a ____________________ relation
  - If nonempty, it has a special node, called the _____ of T, that has no parent
  - Each node v of T, except root, has a unique ***parent*** node w; every node with parent w is a ***child*** of w
- A ___________ exists from the root to every other node.
- Applications:
  - Organization charts
  - File systems

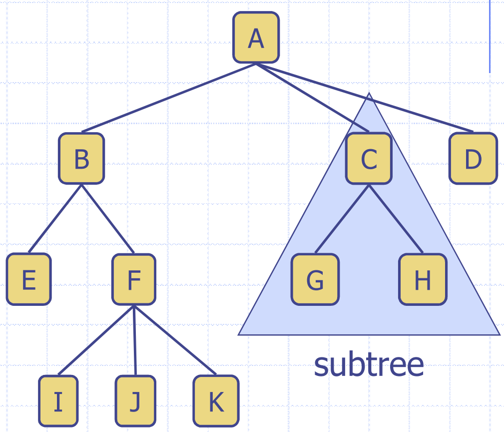


Not a Tree



Tree Terminology

- **Root**: node without parent (A)
- **Internal node**: node with at least one child (A, B, C, F)
- **External node** (a.k.a. _____): node without children (E, I, J, K, G, H, D)
- **Ancestors** of a node: parent, grandparent, grand-grandparent, etc.
- **Descendants** of a node: child, grandchild, grand-grandchild, etc.
- **Subtree**: tree consisting of a node and its descendants
- **Ordered tree**: linear ordering defined for children of each node



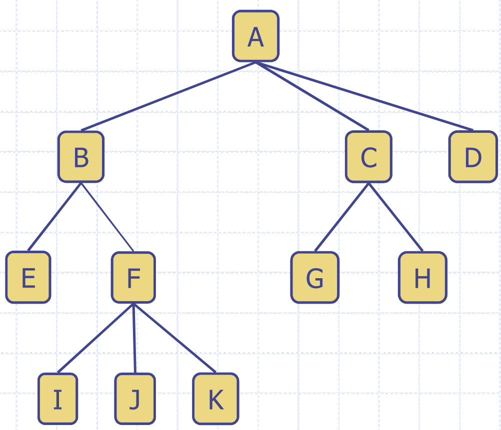
Tree ADT

- We use positions to abstract nodes (same as node in tree)
- Generic methods:
  - integer **size**()
  - boolean **isEmpty**()
  - Iterator **iterator**()
  - Iterable **positions**()
- Accessor methods:
  - position **root**()
  - position **parent**(p)
  - Iterable **children**(p)
- Query methods:
  - boolean **isInternal**(p)
  - boolean **isExternal**(p)
  - boolean **isRoot**(p)
- Update method:
  - element **replace** (p, o)
- Additional update methods may be defined by data structures implementing the Tree ADT

Depth of a Node

- The depth of a node is the number of its ancestors, excluding itself
  - depth(A) = 0, depth(B) = , depth(J) =

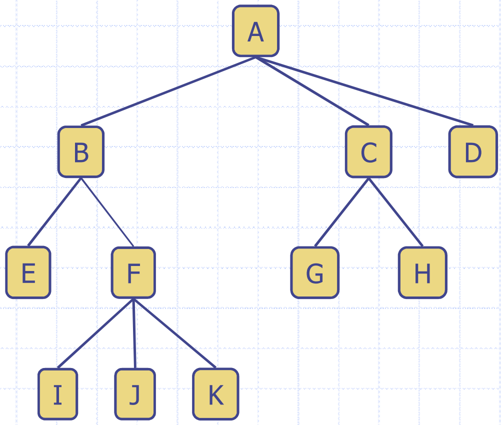
```
Algorithm depth(T, v)
    if T.isRoot(v)
        return 0
    else
        return 1+depth(T, T.parent(v))
```



Height of a Node

- The height(v) in a tree T is
  - If v is an external node, then height(v) = ______
  - Otherwise, height(v) = 1 + max. height of its children

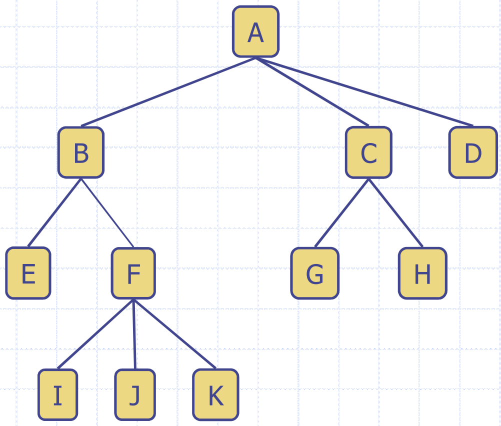
```
Algorithm height(T, v)
    if T.isExternal(v)
        return 0
    else
        h ← 0
        for each child w of v in T do
            h ← max(h, height(T, w))
        return 1+h
```



Height of a Tree

- The height of a tree T is the height of the root
- The height of a tree T is equal to the maximum ______ of a external node of T

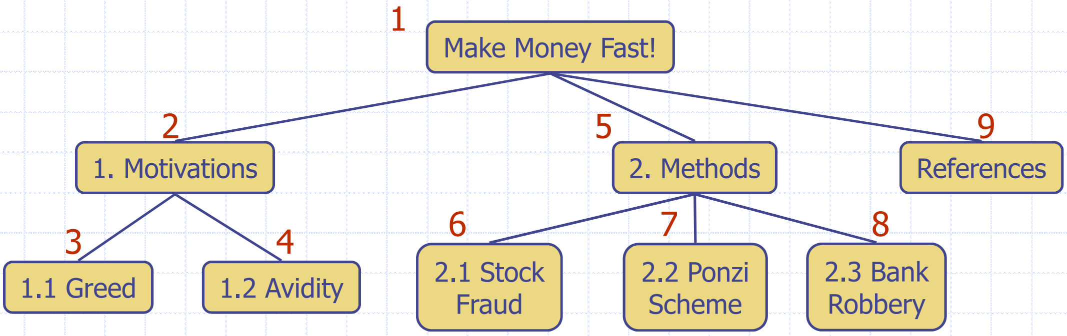
```
Algorithm height(T)
    h ← 0
    for each node v in T do
        if T.isExternal(v) then
            h ← max(h, depth(T, v))
    return h
```



Preorder Traversal

- A traversal visits the nodes of a tree in a systematic manner
- In a preorder traversal, a node is visited before its descendants
- Application: print a structured document

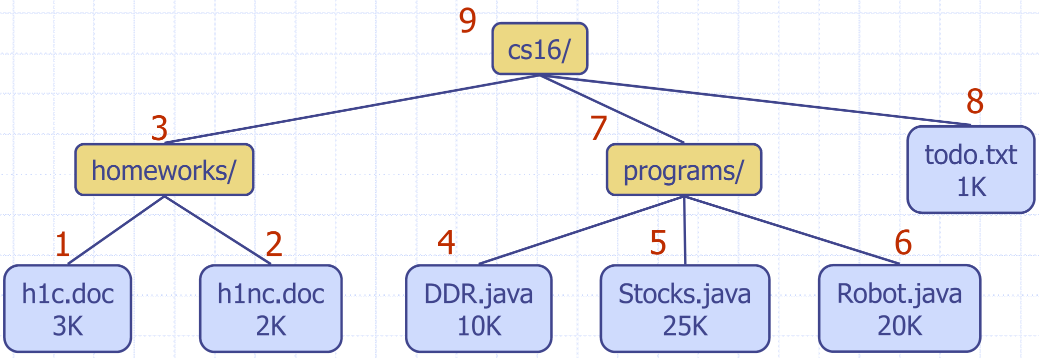
Algorithm *preOrder*(v)
  visit(v)
  for each child w of v
    preorder (w)



Postorder Traversal

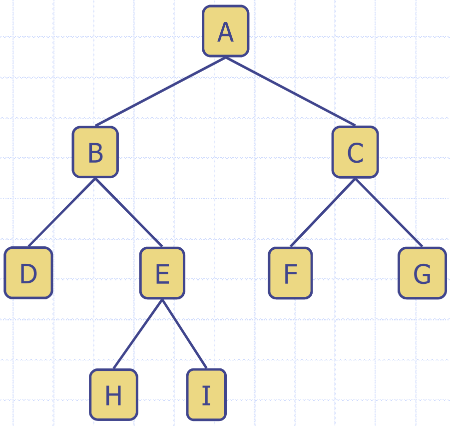
- In a postorder traversal, a node is visited after its descendants
- Application: compute space used by files in a directory and its subdirectories

Algorithm *postOrder*(v)
  for each child w of v
    postOrder (w)
  visit(v)



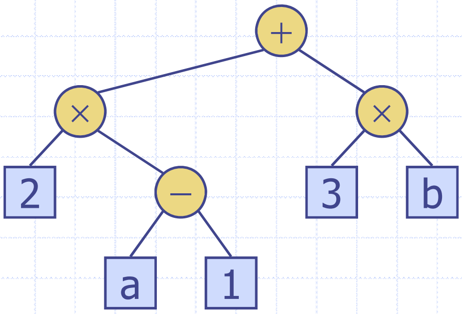
Binary Trees

- A binary tree is an ordered tree with the following properties:
  - Each internal node has at most two children (exactly two for proper binary trees)
  - The children of a node are an ordered pair
- We call the children of an internal node _____ child and _____ child
- Alternative recursive definition: a binary tree is either
  - a tree consisting of a single node, or
  - a tree whose root has an ordered pair of children, each of which is a binary tree
- Applications:
  - arithmetic expressions
  - decision processes
  - searching



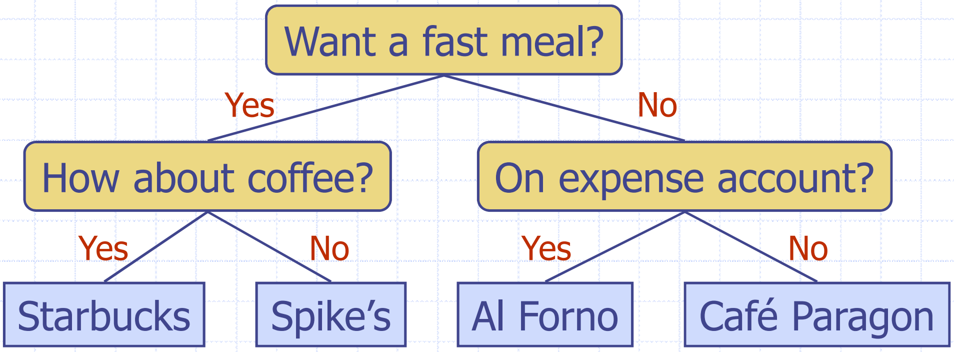
Arithmetic Expression Tree

- Binary tree associated with an arithmetic expression
  - internal nodes: operators
  - external nodes: operands
- Example: arithmetic expression tree for the expression $(2 \times (a - 1) + (3 \times b))$

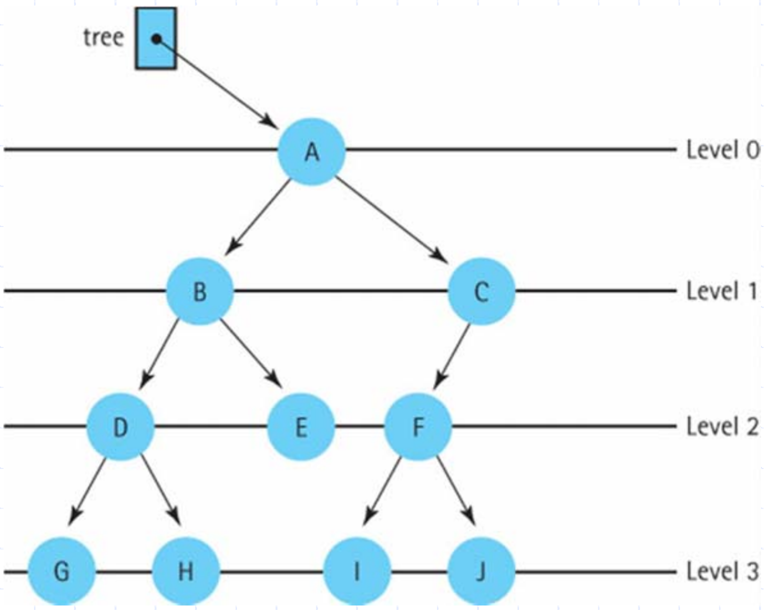


Decision Tree

- Binary tree associated with a decision process
  - internal nodes: questions with yes/no answer
  - external nodes: decisions
- Example: dining decision

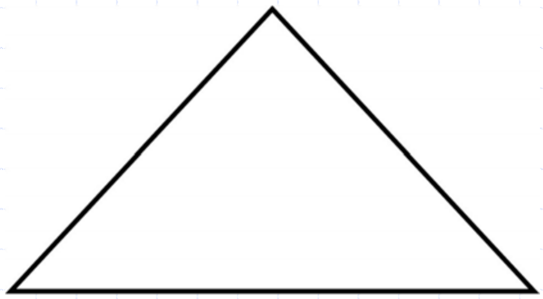


A Binary Tree and Levels



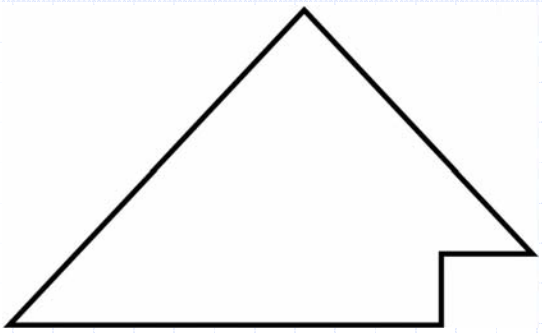
Full Binary Tree

- **Full Binary Tree:** A binary tree in which all of the leaves are on the same level and every nonleaf node has two children

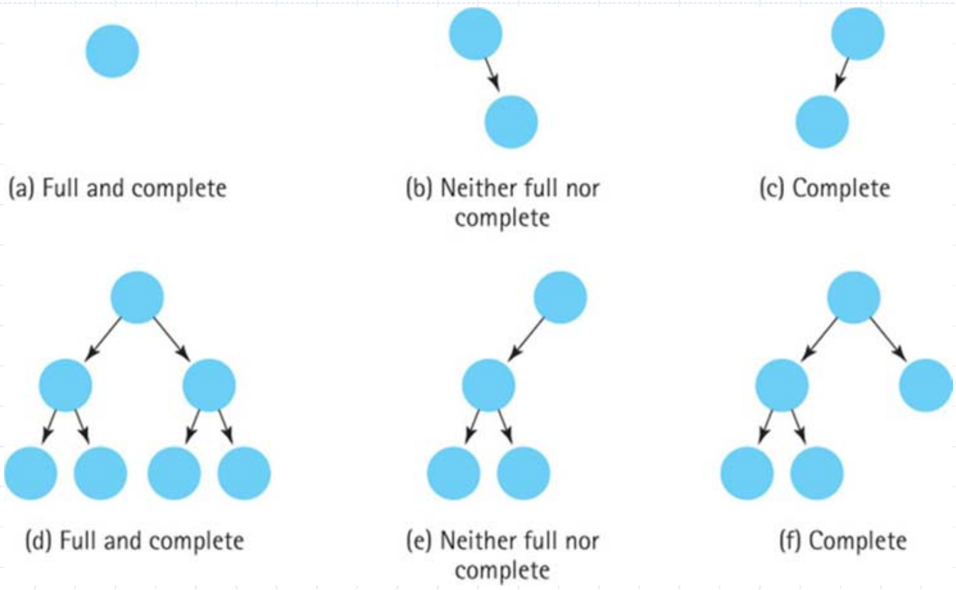


Complete Binary Tree

- **Complete Binary Tree:** A binary tree that is either full or full through the next-to-last level, with the leaves on the last level as far to the left as possible



Examples of Different Types of Binary Trees



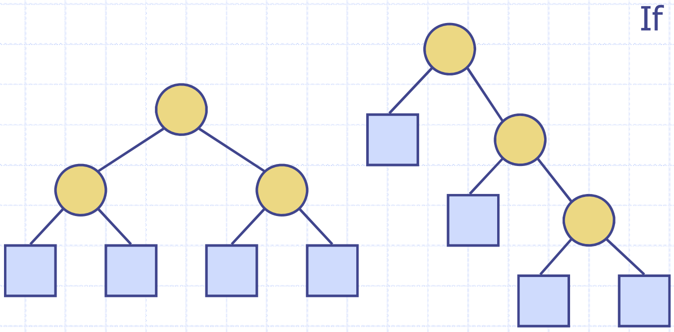
Properties of Binary Trees

- Notation
  - n number of nodes
  - n_e number of external nodes
  - n_i number of internal nodes
  - h height

- Properties:
  - $h+1 \leq n \leq 2^{h+1} - 1$
  - $1 \leq n_e \leq 2^h$
  - $h \leq n_i \leq 2^h - 1$
  - $\log_2 (n+1) - 1 \leq h \leq (n-1)$

If proper trees:

- $2h+1 \leq n \leq 2^{h+1} - 1$
- $h+1 \leq n_e \leq 2^h$
- $h \leq n_i \leq 2^h - 1$
- $\log_2 (n+1) - 1 \leq h \leq (n-1)/2$
- $n_e = n_i + 1$



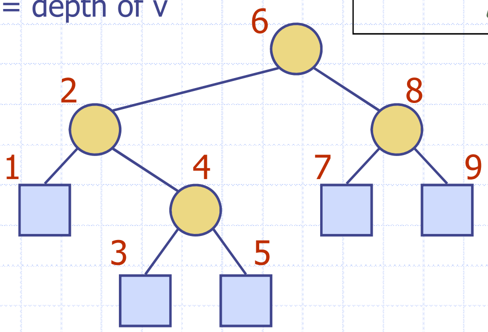
BinaryTree ADT

- The BinaryTree ADT extends the Tree ADT, i.e., it inherits all the methods of the Tree ADT
- Additional methods:
  - position left(p)
  - position right(p)
  - boolean hasLeft(p)
  - boolean hasRight(p)
- Update methods may be defined by data structures implementing the BinaryTree ADT

Inorder Traversal

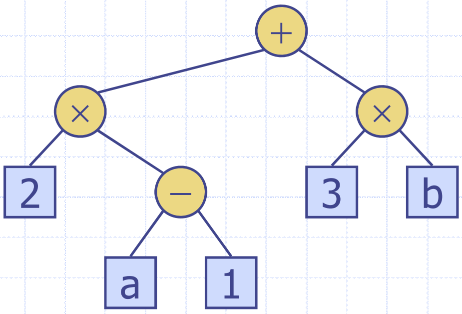
- In an inorder traversal a node is visited after its left subtree and before its right subtree
- Application: draw a binary tree
  - x(v) = inorder rank of v
  - y(v) = depth of v

Algorithm *inOrder*(*v*)
    if *hasLeft* (*v*)
        inOrder (*left* (*v*))
    visit(*v*)
    if *hasRight* (*v*)
        inOrder (*right* (*v*))



Print Arithmetic Expressions

- Specialization of an inorder traversal
  - print operand or operator when visiting node
  - print "(" before traversing left subtree
  - print ")" after traversing right subtree

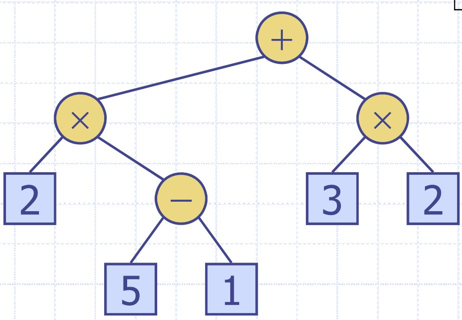


Algorithm *printExpression*(*v*)
  if *hasLeft* (*v*)
    print("(")
    inOrder (*left*(*v*))
  print(*v.element* ())
  if *hasRight* (*v*)
    inOrder (*right*(*v*))
    print (")")

$((2 \times (a - 1)) + (3 \times b))$

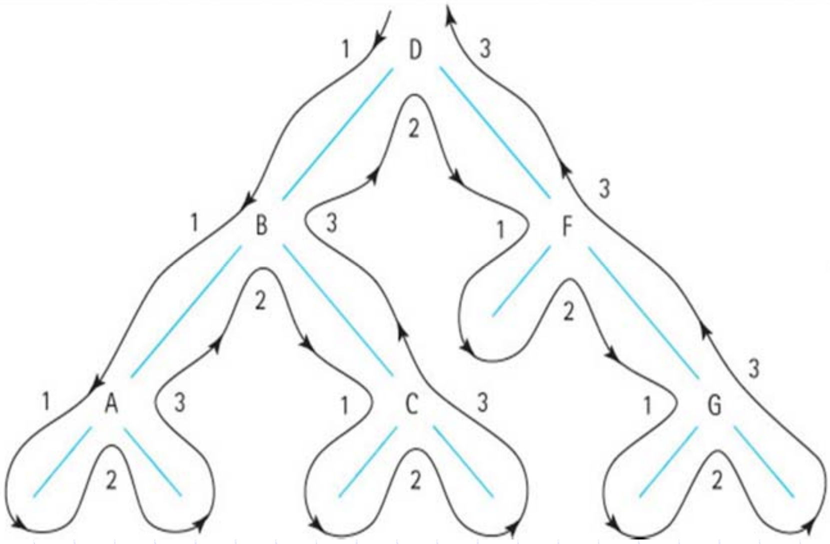
Evaluate Arithmetic Expressions

- Specialization of a postorder traversal
  - recursive method returning the value of a subtree
  - when visiting an internal node, combine the values of the subtrees



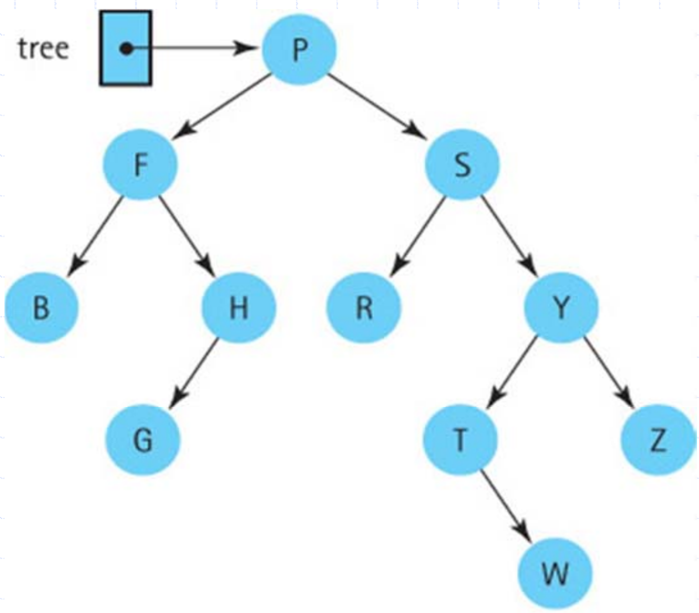
Algorithm *evalExpr*(*v*)
  if *isExternal* (*v*)
    return *v.element* ()
  else
    $x \leftarrow$ *evalExpr*(*leftChild* (*v*))
    $y \leftarrow$ *evalExpr*(*rightChild* (*v*))
    $\Diamond \leftarrow$ operator stored at *v*
    return $x \Diamond y$

Euler Tour Traversal



Preorder: DBACFG
Inorder: ABCDFG
Postorder: ACBGFD

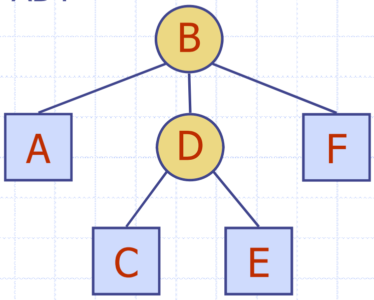
Three Binary Tree Traversals

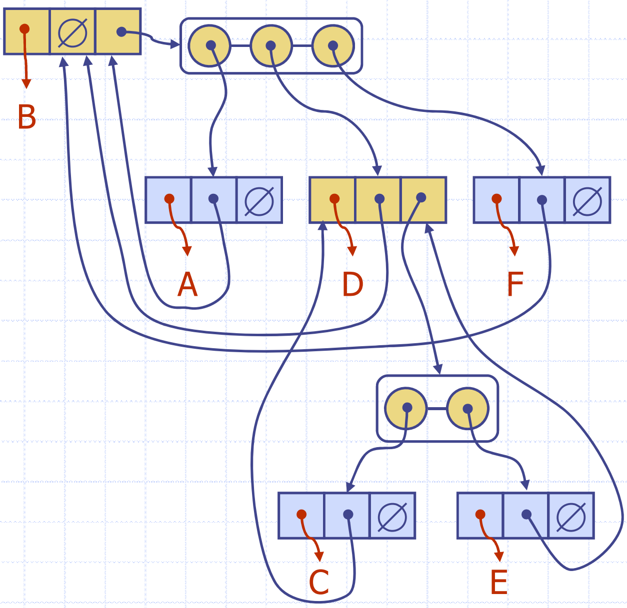


Inorder: B F G H P R S T W Y Z
Preorder: P F B H G S R Y T W Z
Postorder: B G H F R W T Z Y S P

Linked Structure for Trees

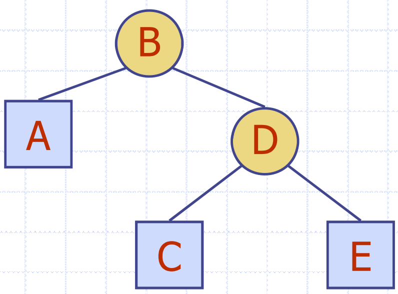
- A node is represented by an object storing
  - Element
  - Parent node
  - Sequence of children nodes
- Node objects implement the Position ADT

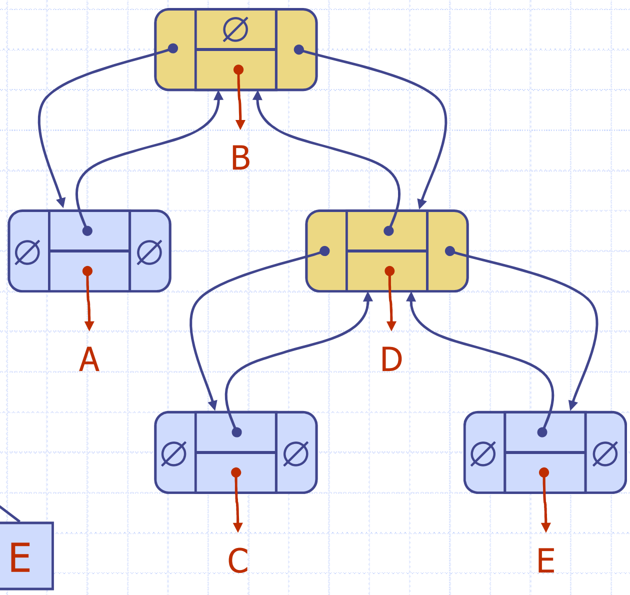




Linked Structure for Binary Trees

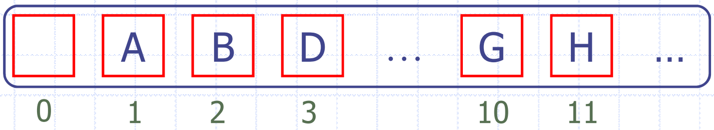
- A node is represented by an object storing
  - Element
  - Parent node
  - Left child node
  - Right child node
- Node objects implement the Position ADT

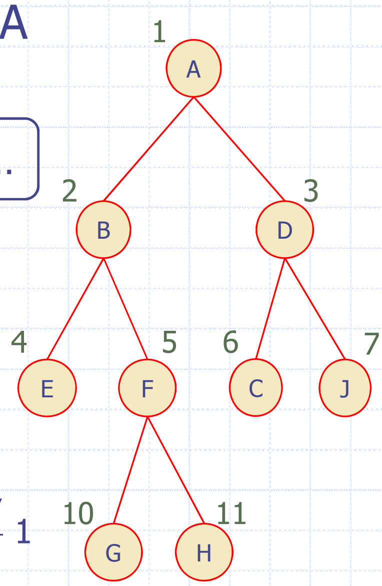




Array-Based Representation of Binary Trees

- Nodes are stored in an array A





- Node v is stored at A[rank(v)]
  - rank(root) = 1
  - if node is the left child of parent(node), rank(node) = 2 · rank(parent(node))
  - if node is the right child of parent(node), rank(node) = 2 · rank(parent(node)) + 1