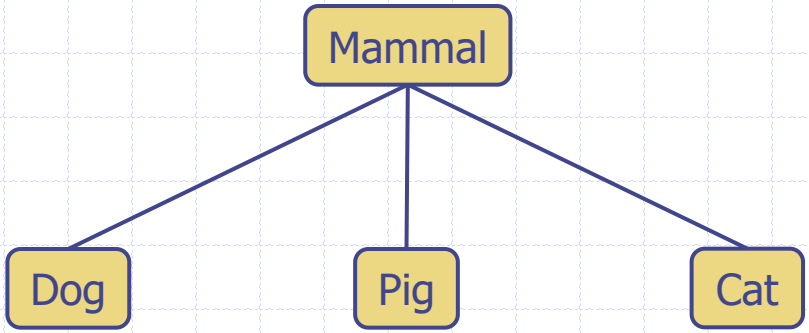
Presentation for use with the textbook Data Structures and Algorithms in Java, 6th edition, by M. T. Goodrich, R. Tamassia, and M. H. Goldwasser, Wiley, 2014

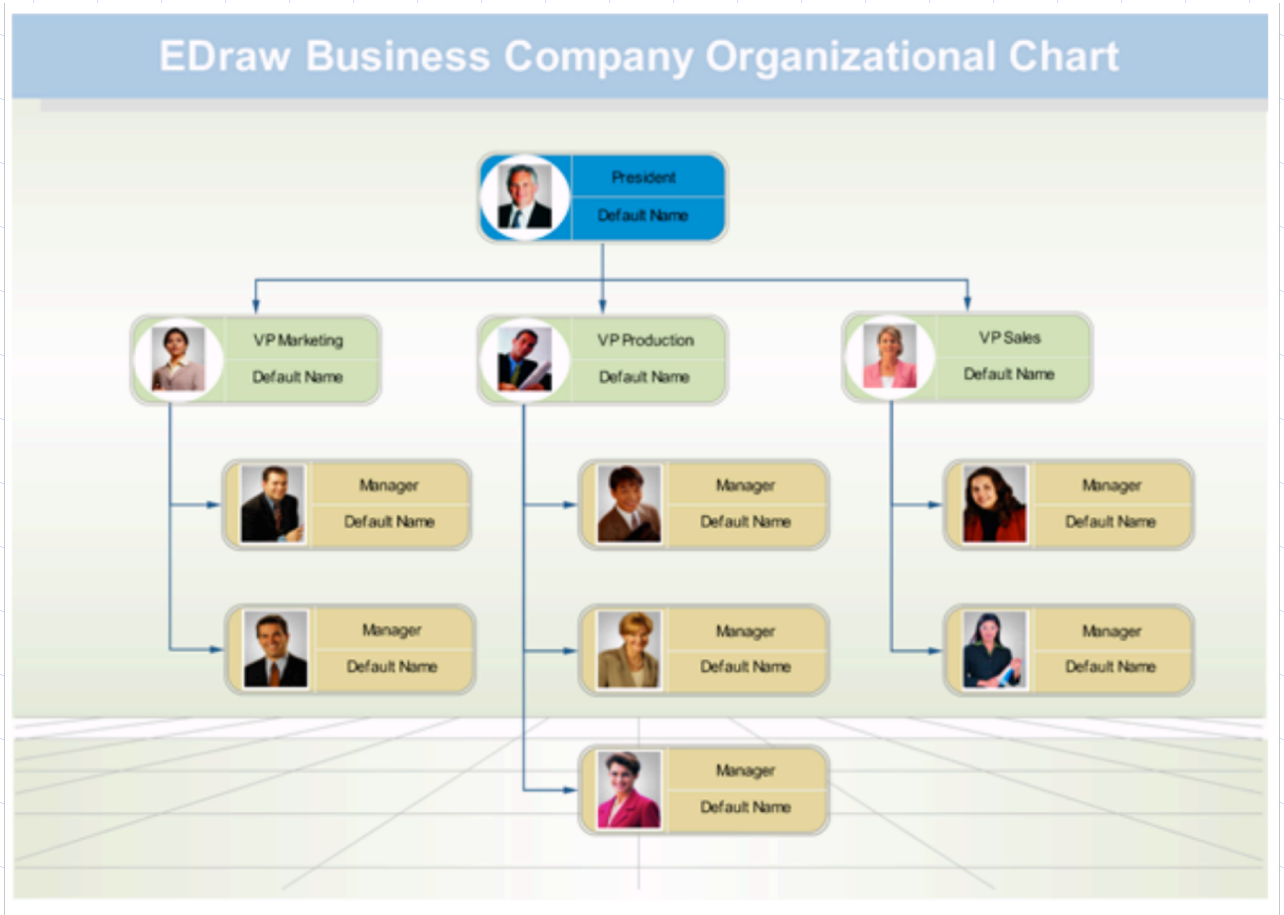
Trees and its variations



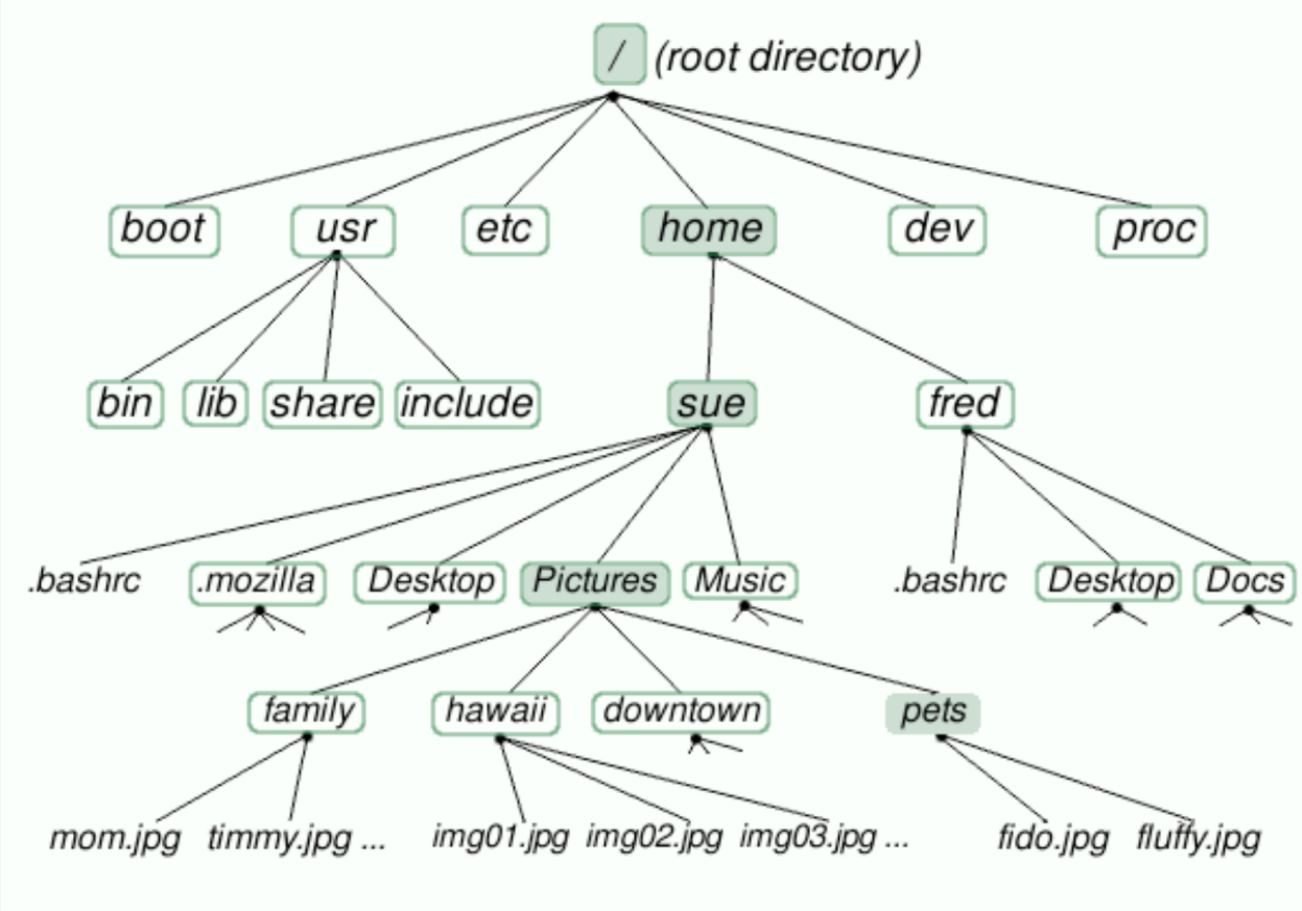
Abstract Data Type (ADT)

- An abstract data type (ADT) is an abstraction of a data structure
- An ADT specifies:
  - Data stored
  - Operations on the data
  - Error conditions associated with operations
- Linear ADT: the elements form a sequence. Examples: Array ADT, List ADT, Queue ADT
- Non-linear ADT: elements do not form a sequence. Example: **Trees**

A Hierarchical Structure

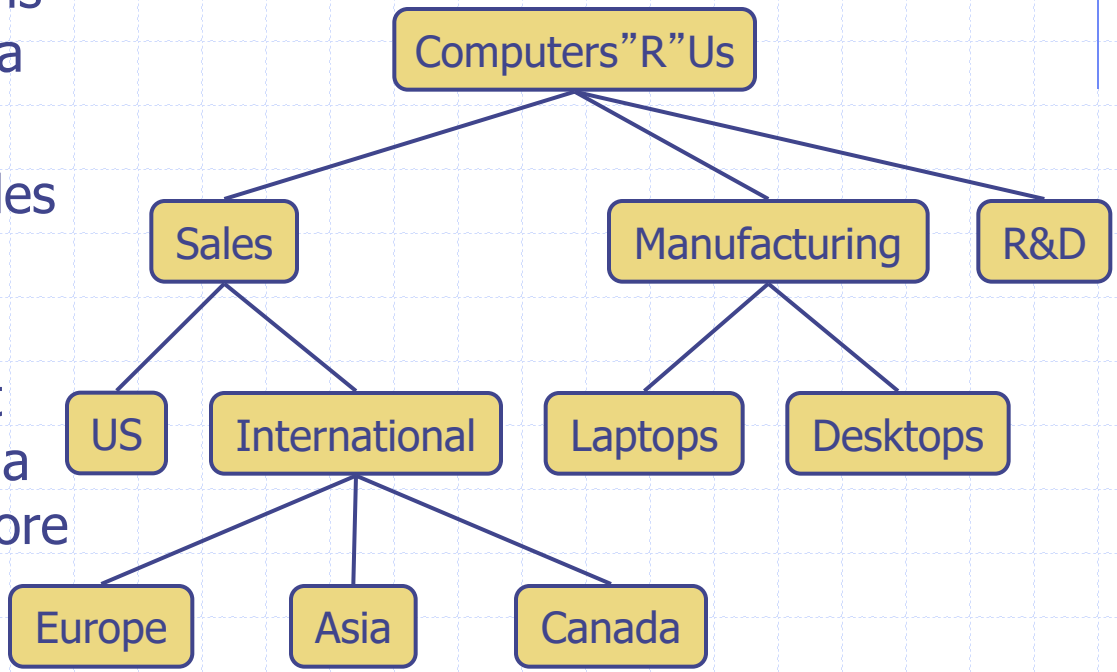


Linux/Unix file systems



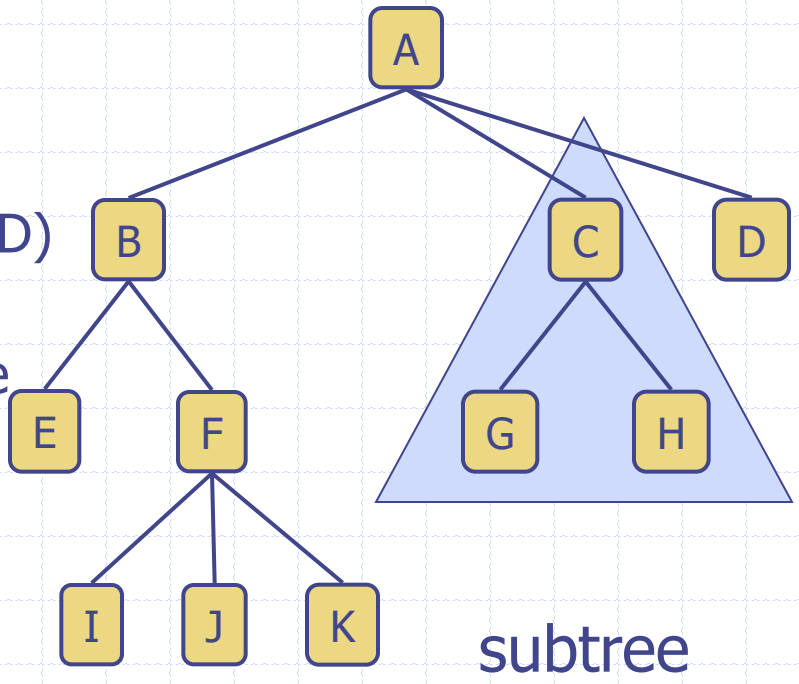
Tree: A Hierarchical ADT

- A tree (upside down) is an abstract model of a hierarchical structure
- A tree consists of nodes with a parent-child relation
- Each element (except the top element) has a parent and zero or more children elements



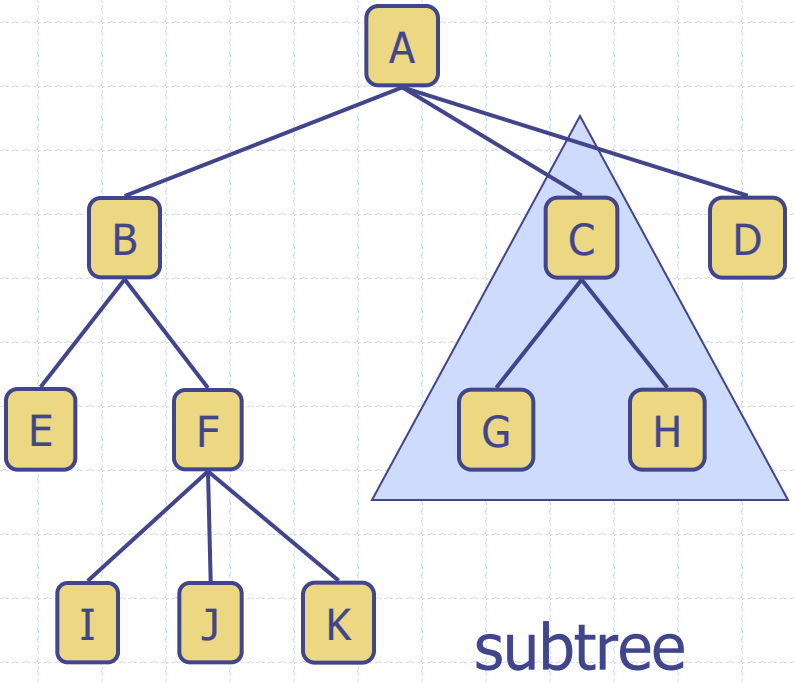
Tree Terminology

- Root: node without parent (A)
- Internal node: node with at least one child (A, B, C, F)
- External node (a.k.a. leaf): node without children (E, I, J, K, G, H, D)
- Subtree: tree consisting of a node and its descendants



Tree Terminology

- Ancestors of a node: parent, grandparent, grand-grandparent, etc.
- Depth of a node: number of ancestors
- Height of a tree: maximum depth of any node
- Descendant of a node: child, grandchild, grand-grandchild, etc.



Tree ADT

- We use positions to define the tree ADT
- The positions in a tree are its nodes and neighboring positions satisfy the parent-child relationships

method	description
root()	Return the tree's root; error if tree is empty
parent(v)	Return v's parent; error if v is a root
children(v)	Return v's children (an iterable collection of nodes)
isRoot(v)	Test whether v is a root
isExternal(v)	Test whether v is an external node
isInternal(v)	Test whether v is an internal node

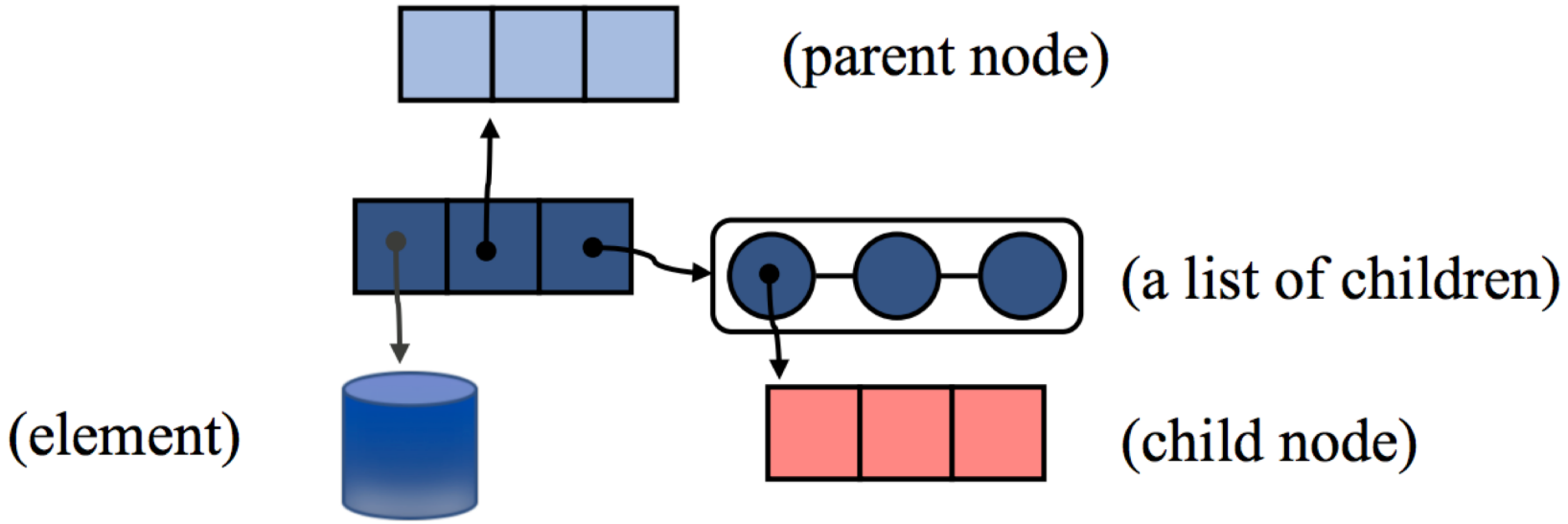
Tree ADT

- Generic methods (not necessarily related to a tree structure):

method	description
isEmpty()	Test whether the tree has any node or not
size()	Return the number of nodes in the tree
iterator()	Return an iterator of all the elements stored in the tree
positions()	Return an iterable collection of all the nodes of the tree
replace(v,e)	Replace with e and return the element stored at node v

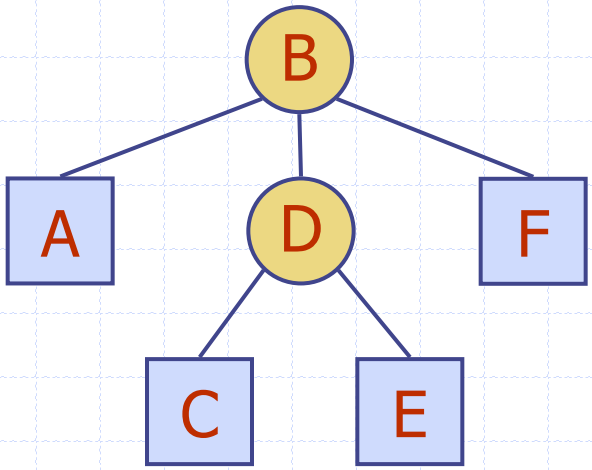
Linked Structure for Trees

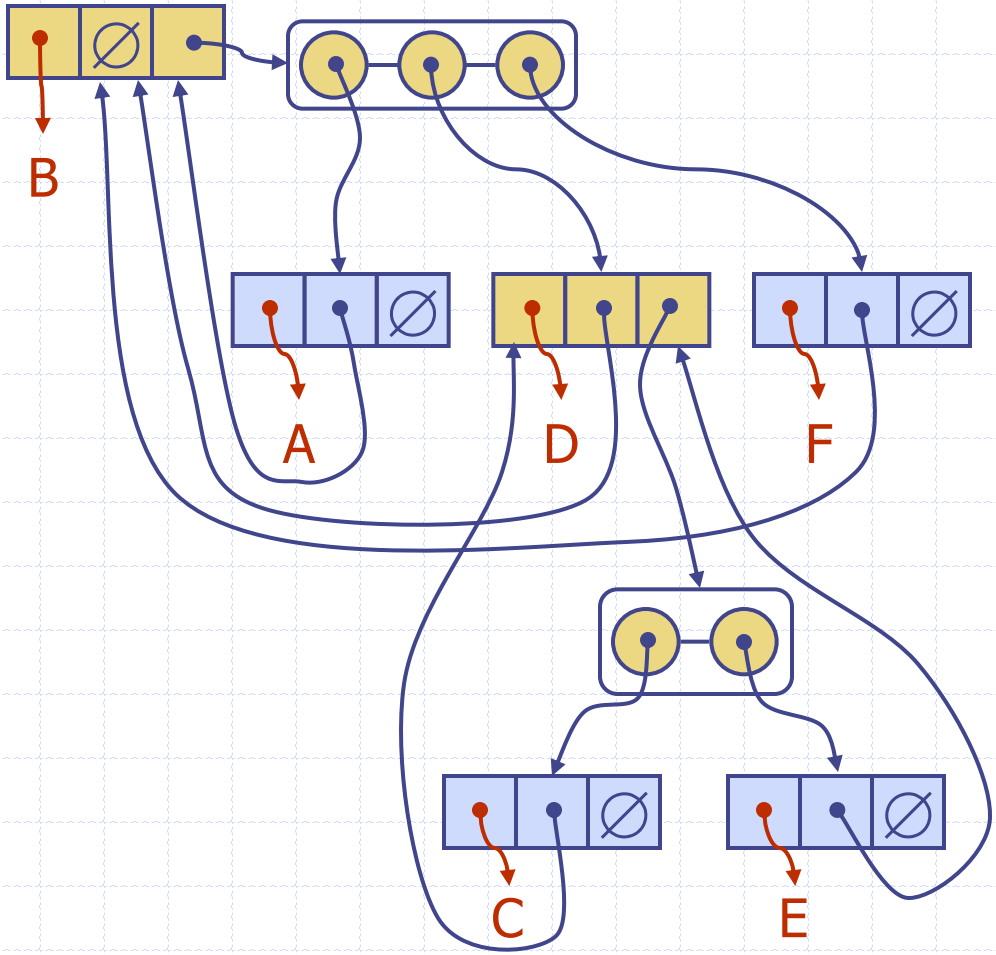
- A node is represented by an object storing
  - Element
  - Parent node
  - Sequence of children nodes



Linked Structure for Trees

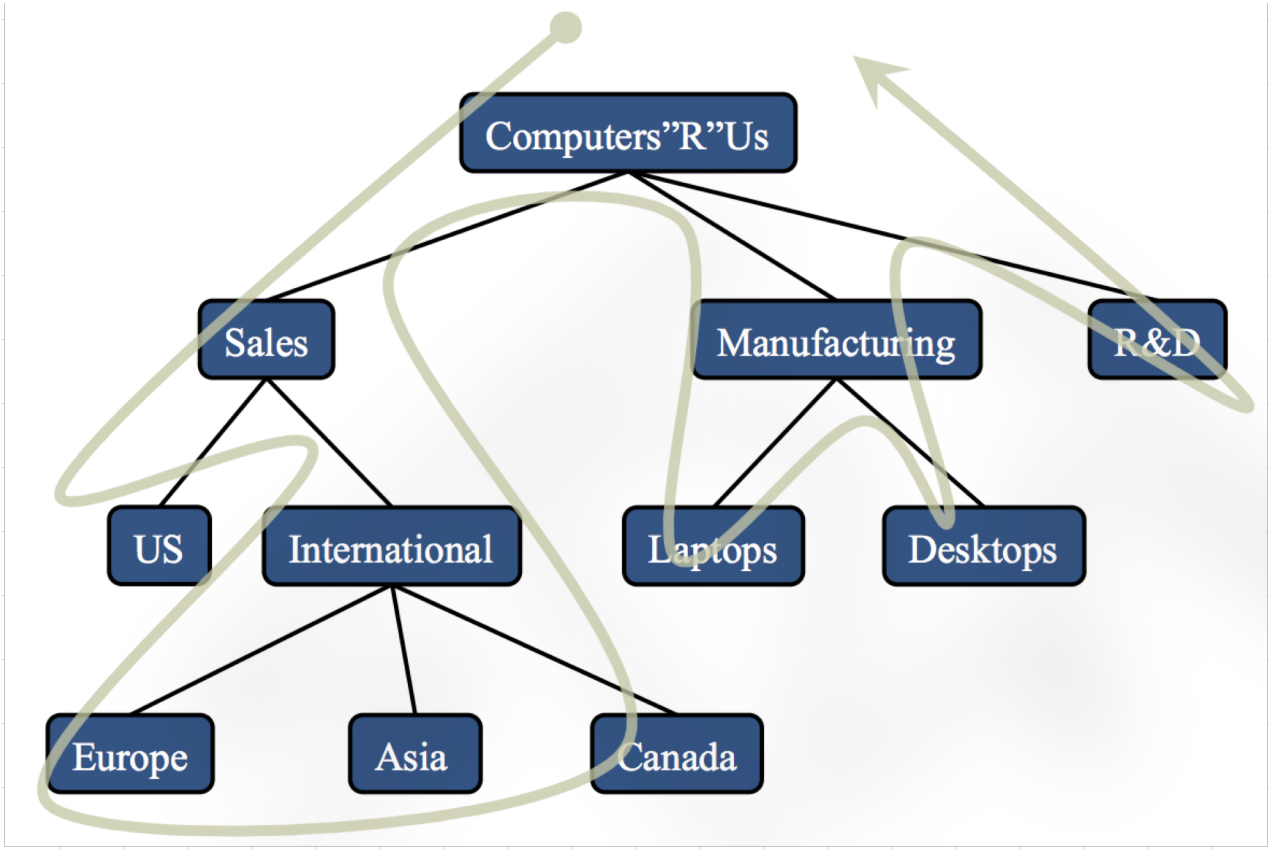
- Node objects implement the Position ADT





Tree Traversal

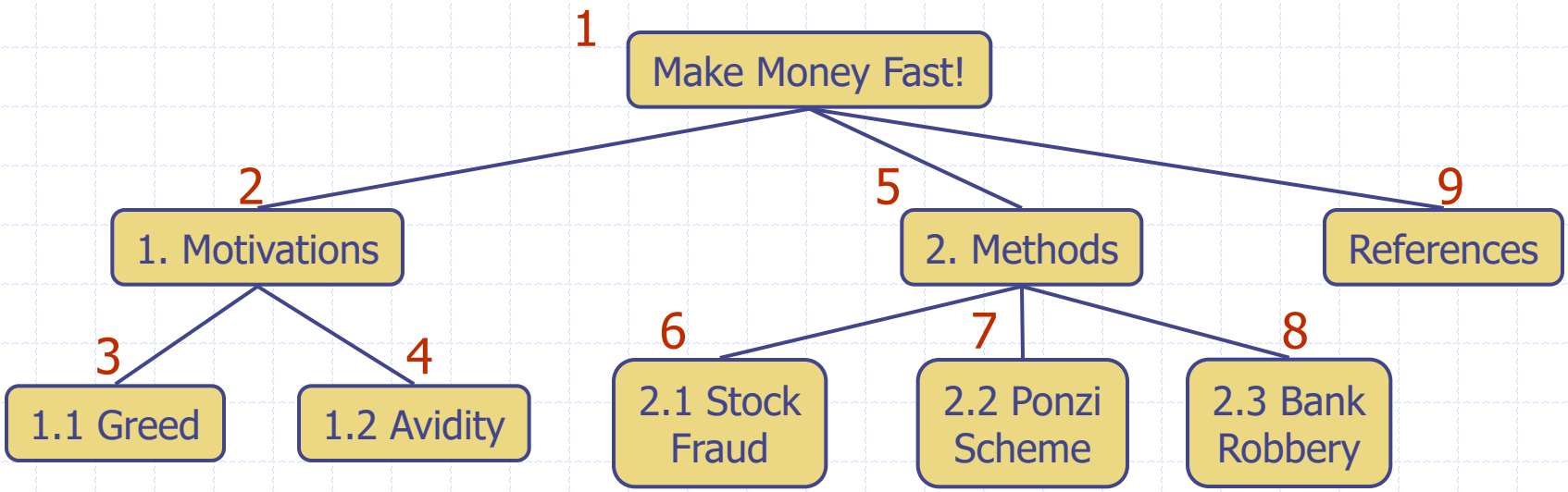
- Visit all nodes in a tree
- Perform some operations during the visit



Preorder Traversal

- In a preorder traversal, a node is visited before its descendants
- Application: print a structured document

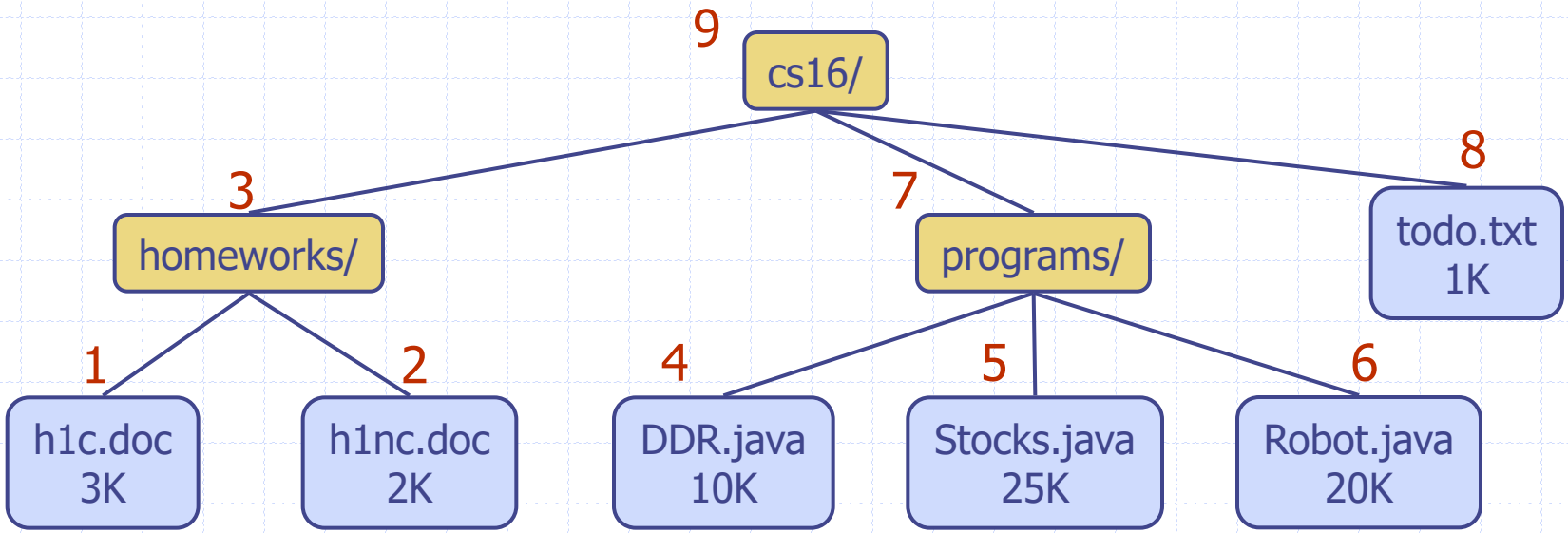
Algorithm *preOrder*(*v*)
    visit(*v*)
    for each child *w* of *v*
        preorder (*w*)



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

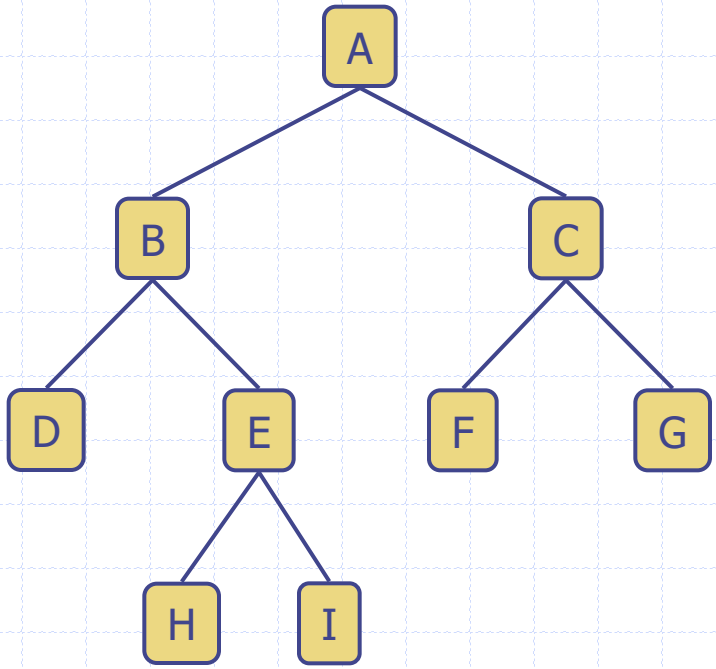
Algorithm *postOrder*(*v*)
  for each child *w* of *v*
    postOrder (*w*)
  visit(*v*)



Binary Trees

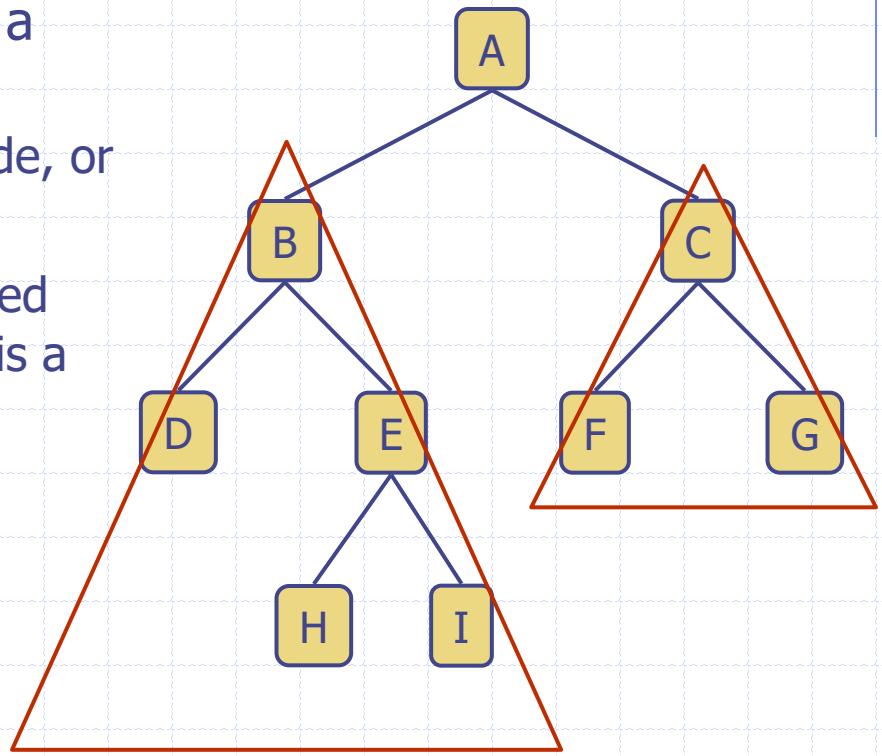
- A binary tree is a tree with the following properties:
  - Each internal node has at most two children.
  - The children of a node are an ordered pair
- We call the children of an internal node left child and right child

- Applications:
  - arithmetic expressions
  - decision processes
  - searching



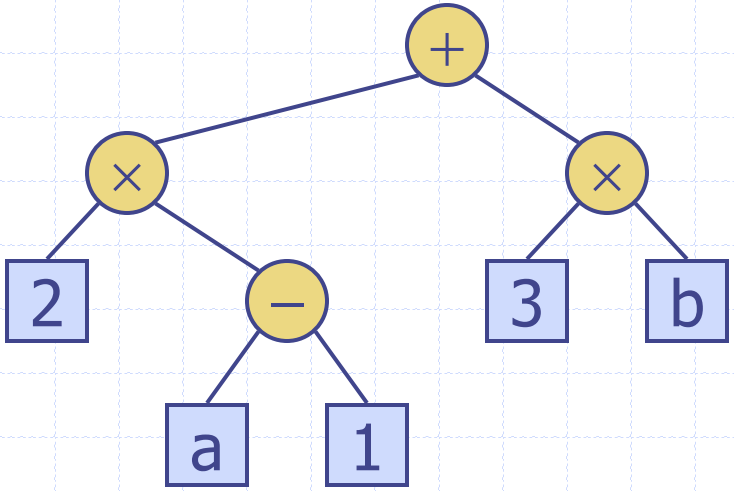
Binary Trees

- 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



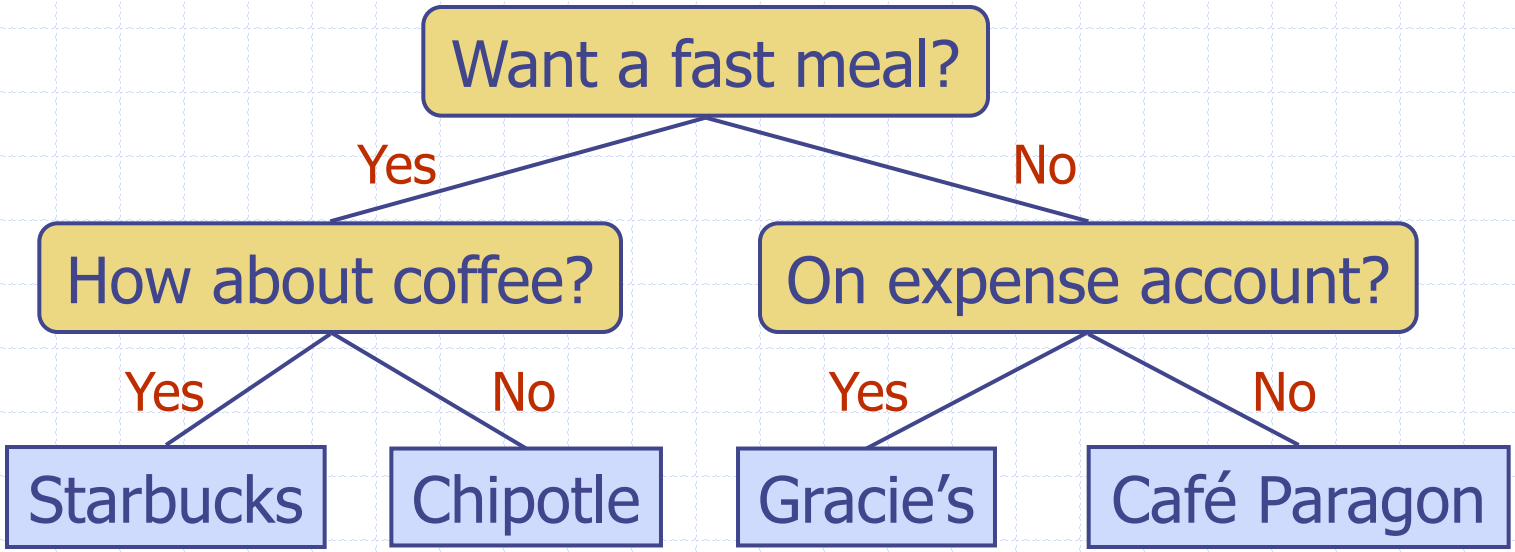
Arithmetic Expression Tree

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



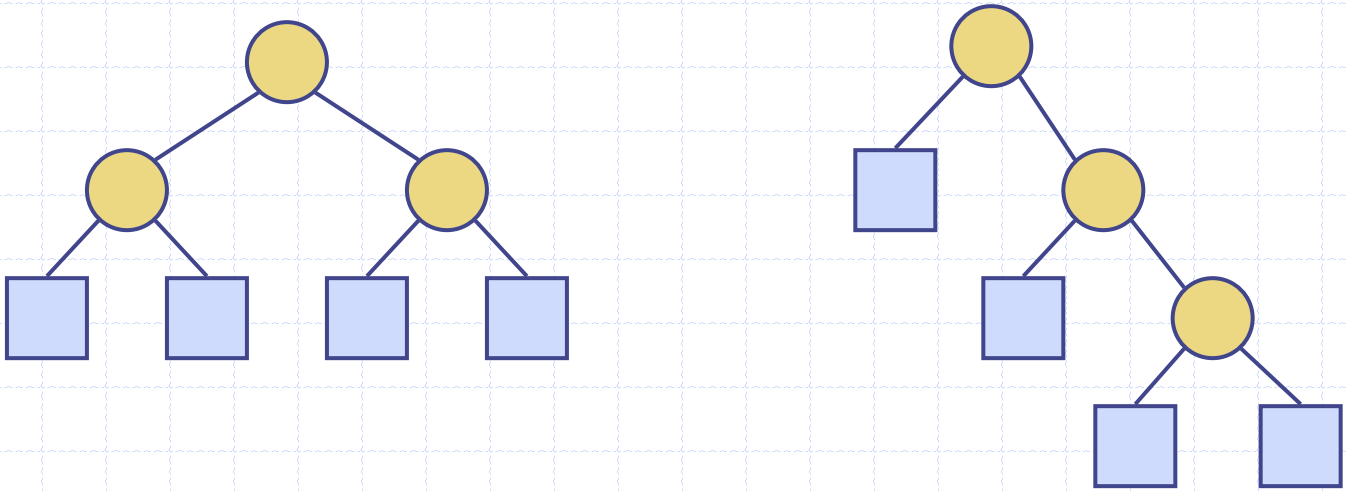
Decision Tree

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



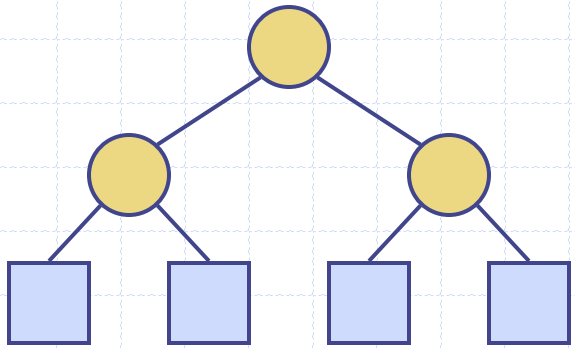
Proper Binary Trees

- Each internal node has exactly 2 children



Properties of Proper Binary Trees

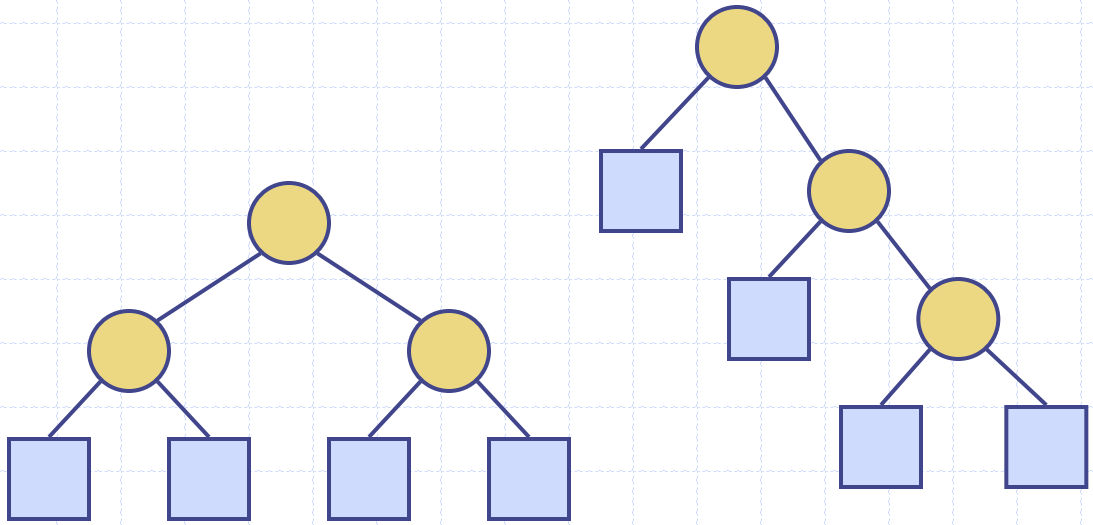
- Notation
  - n number of nodes
  - e number of external nodes
  - i number of internal nodes
  - h height(maximum depth of a node)



- Properties:
  - $e = i + 1$
  - $n = 2e - 1$
  - $h \leq i$
  - $h \leq (n - 1)/2$
  - $e \leq 2^h$
  - $h \geq \log_2 e$
  - $h \geq \log_2 (n + 1) - 1$

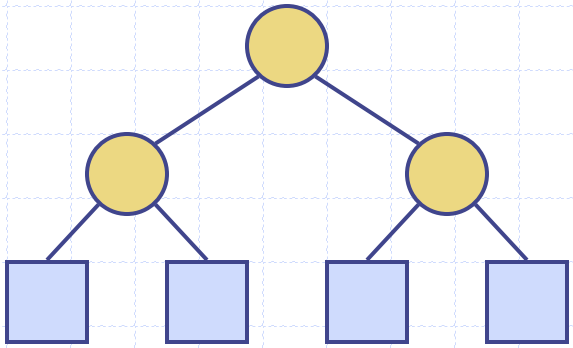
Properties of Proper Binary Trees

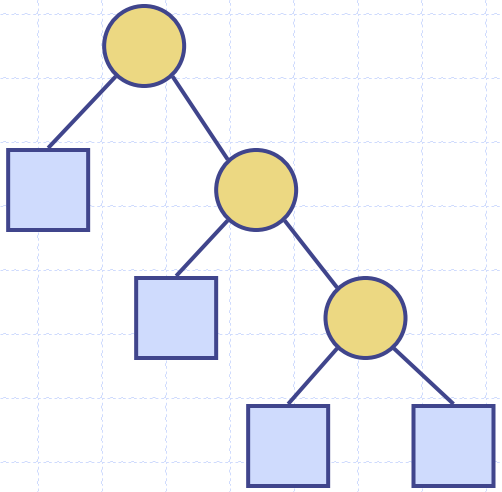
- Properties:
  - $e = i + 1$
  - $n = e + i = 2e - 1 = 2i + 1$



Properties of Proper Binary Trees

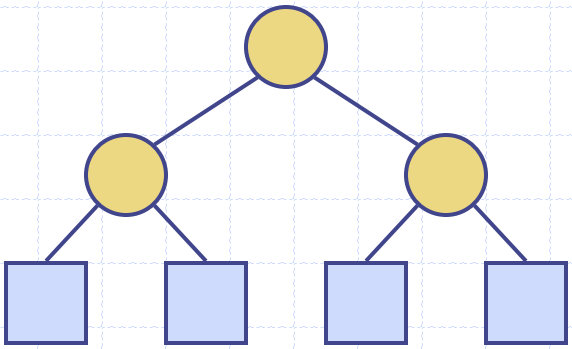
- Properties:
  - $h \leq i$
  - $h \leq (n-1)/2$





Properties of Proper Binary Trees

- Properties:
  - $e \leq 2^h$
  - $h \geq \log_2 e$
  - $h \geq \log_2 (n + 1) - 1$



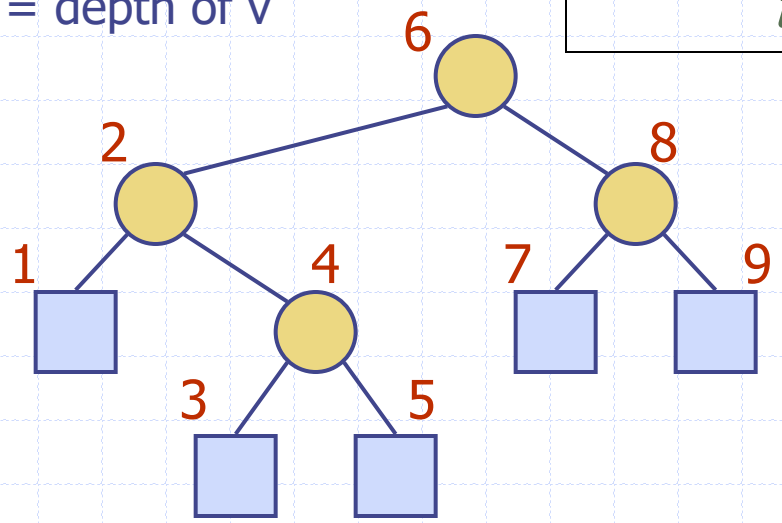
BinaryTree ADT

- 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)
  - position sibling(p)
- The above methods return null when there is no left, right, or sibling of p, respectively
- Update methods may be defined by data structures implementing the BinaryTree ADT

Inorder Traversal of a Binary Tree

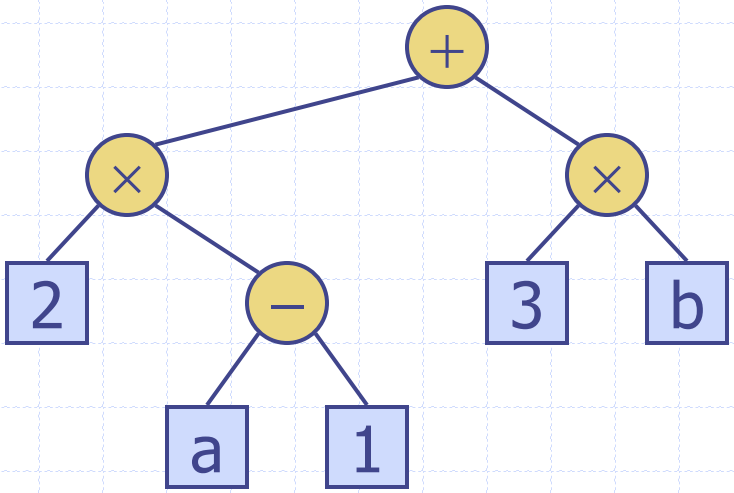
- In an inorder traversal a node is visited after its left subtree and before its right subtree
- Application: draw a binary tree
  - x(v) = inorder rank of v
  - y(v) = depth of v

Algorithm *inOrder*(*v*)
  if *left* (*v*) ≠ **null**
    inOrder (*left* (*v*))
  visit(*v*)
  if *right*(*v*) ≠ **null**
    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)

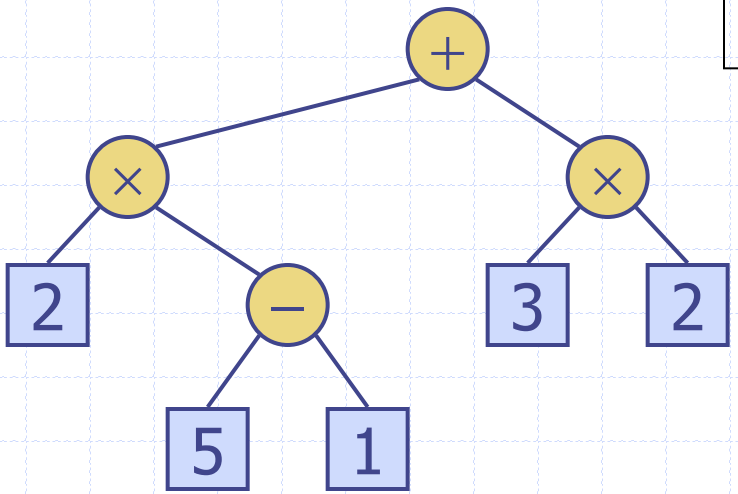
```
if left (v) ≠ null
    print("(" )
    printExpression (left(v))
print(v.element ())
if right(v) ≠ null
    printExpression(right(v))
    print (")" )
```

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

Evaluate Arithmetic Expressions

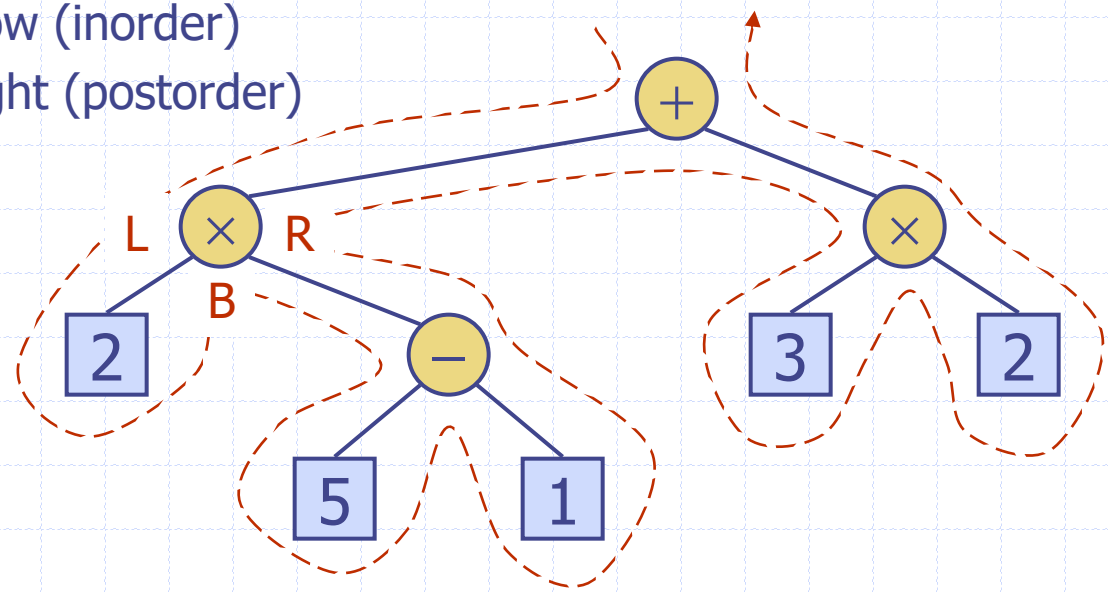
- Specialization of a postorder traversal
  - recursive method returning the value of a subtree
  - when visiting an internal node, combine the values of the subtrees

Algorithm *evalExpr*(*v*)
  if *isExternal* (*v*)
    return *v.element* ()
  else
    $x \leftarrow$ *evalExpr*(*left*(*v*))
    $y \leftarrow$ *evalExpr*(*right*(*v*))
    $\lozenge \leftarrow$ operator stored at *v*
    return $x \lozenge y$



Euler Tour Traversal

- Generic traversal of a binary tree
- Includes special cases the preorder, postorder and inorder traversals
- Walk around the tree and visit each edge twice (visit each internal node three times):
  - on the left (preorder)
  - from below (inorder)
  - on the right (postorder)



A template method pattern

- A generic computation mechanism for Binary Trees
- Specialized for an application by redefining the visit actions

```
Algorithm eularTour(T,v)
Perform the action for visiting node v on the left
If v has a left child u in T then
        eularTour(T, u)
Perform the action for visiting node v from below
If v has a right child w in T then
        eularTour(T, w)
Perform the action for visiting node v on the right
```

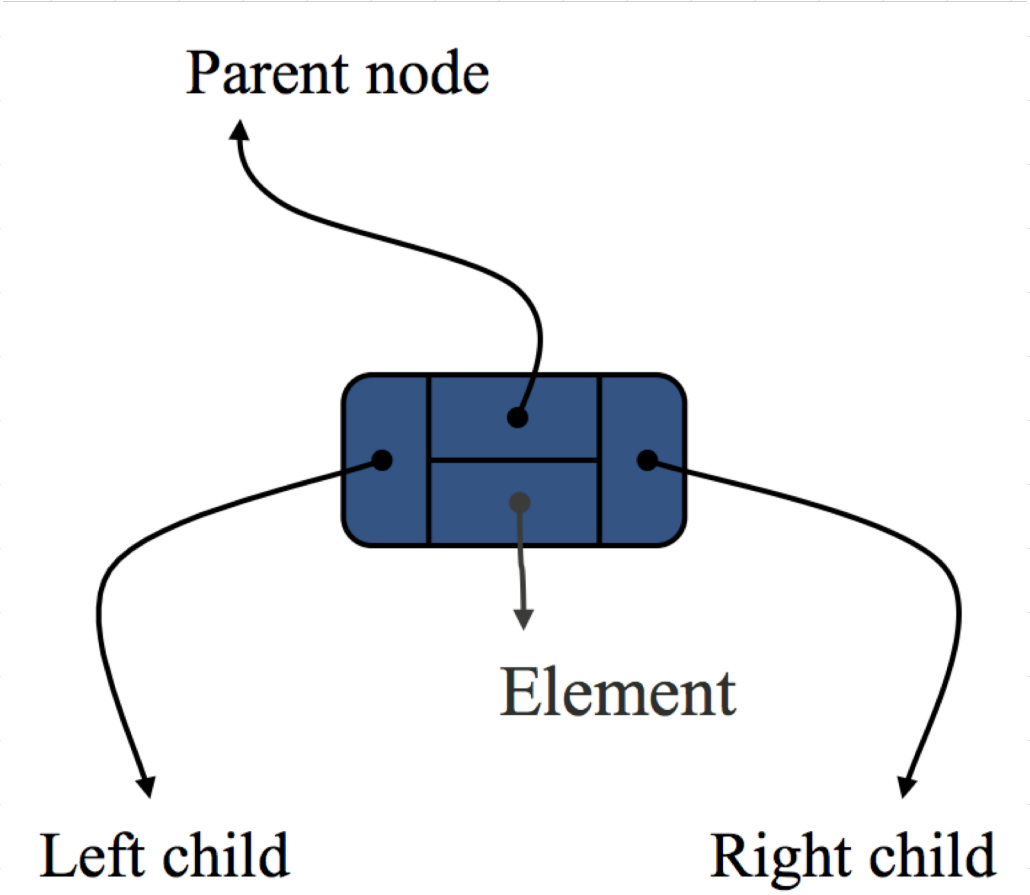
An application of EularTour

- printExpression
  - On the left action: print (
  - From below action: print v
  - On the right action: print)

```
Algorithm printExpression(T,v)
if T.isInternal(v) then print "("
If v has a left child u in T then
        printExpression(T, u)
print(v)
If v has a right child w in T then
        printExpression(T, w)
if T.isInternal(v) then print ")"
```

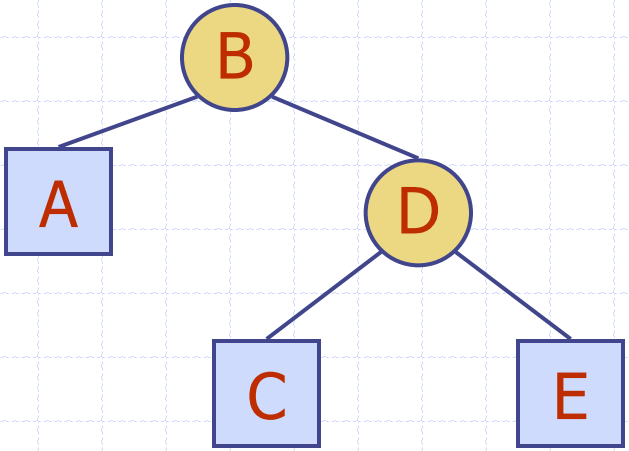
Linked Structure for Binary Trees

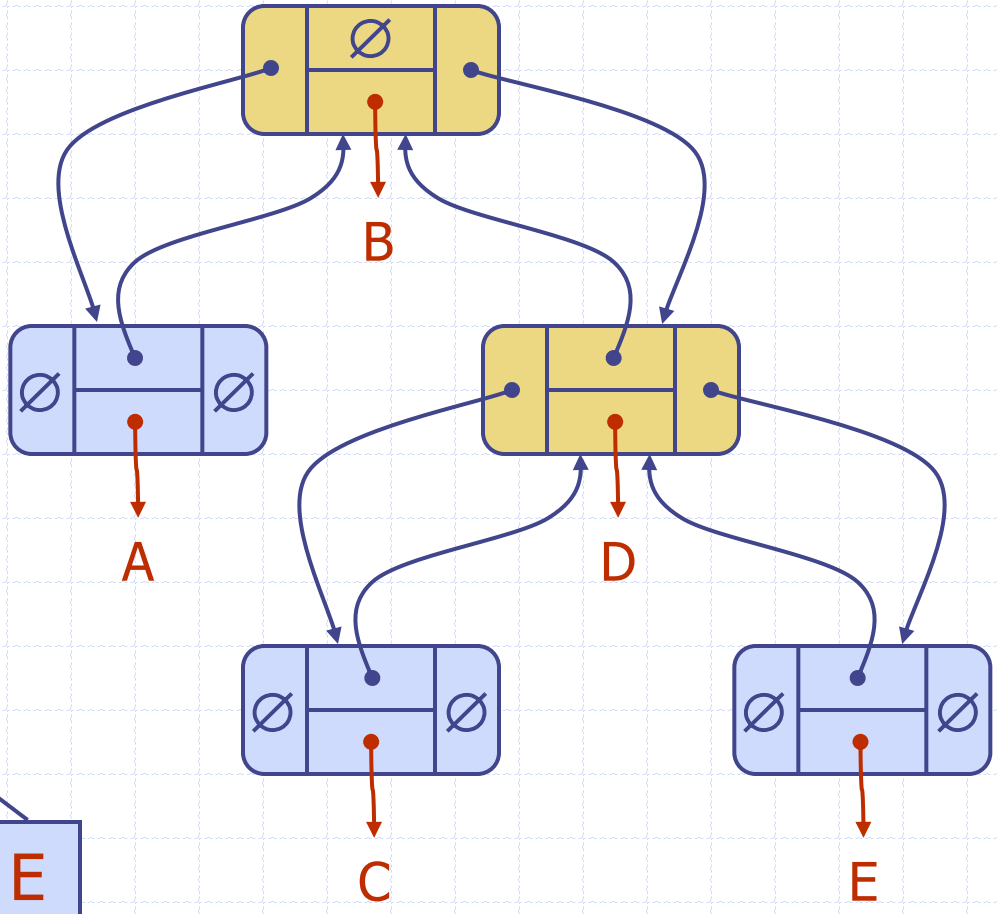
- A node is represented by an object storing
  - Element
  - Parent node
  - Left child node
  - Right child node



Linked Structure for Binary Trees

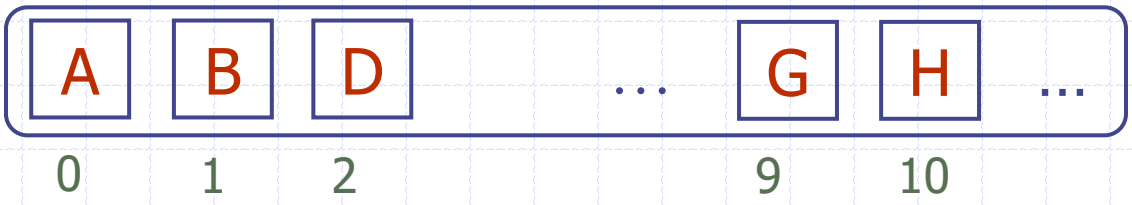
- Node objects implement the Position ADT

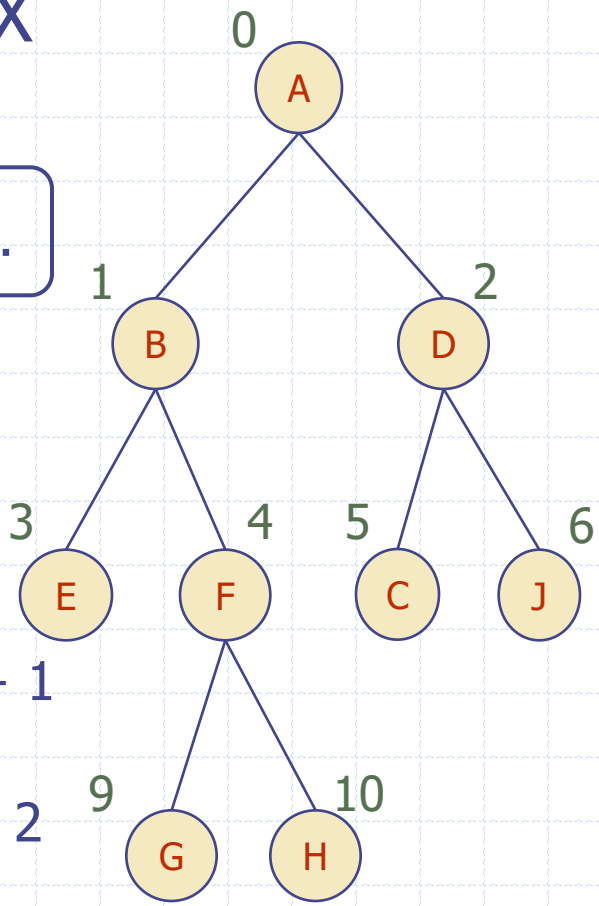




Array-Based Representation of Binary Trees

- Nodes are stored in an array X





Node v is stored at X[rank(v)]

- rank(root) = 0
- if node is the left child of parent(node),
  rank(node) = 2 · rank(parent(node)) + 1
- if node is the right child of parent(node),
  rank(node) = 2 · rank(parent(node)) + 2