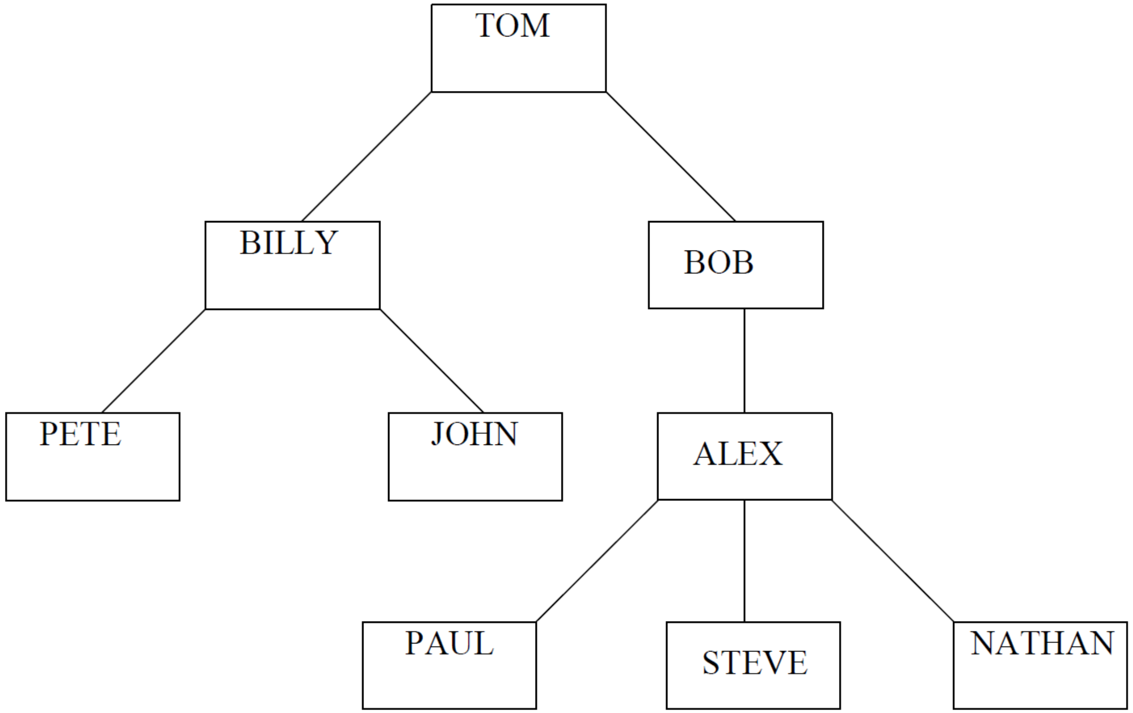
INT202
Complexity of Algorithms
Data Structures

XJTLU/SAT/INT

SEM2 AY2020-2021

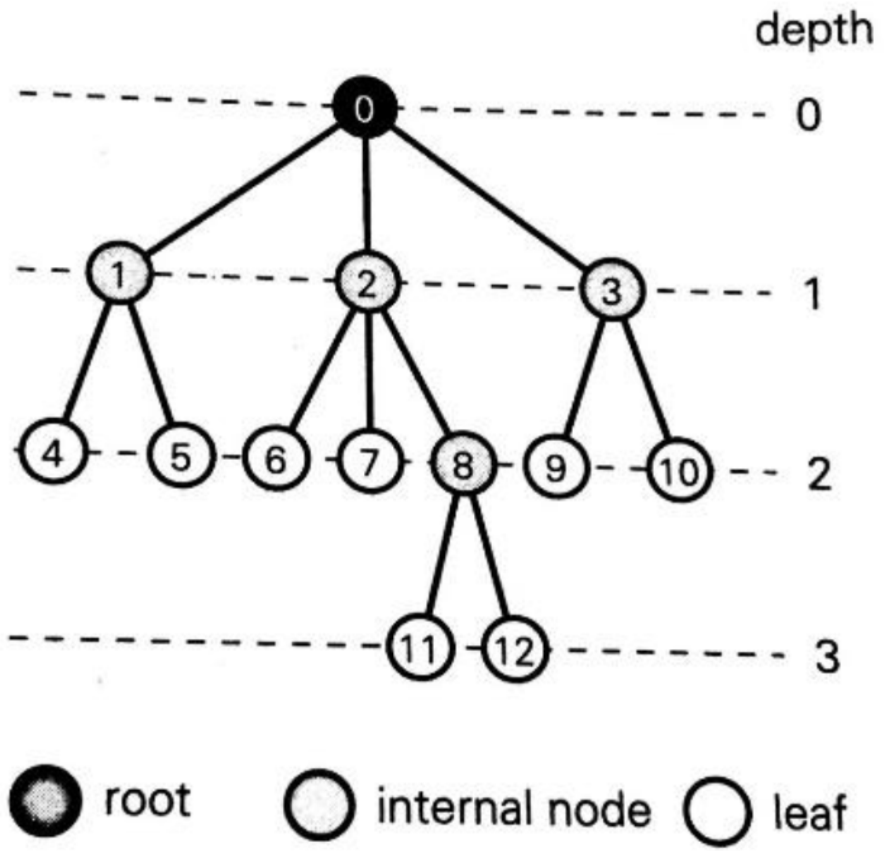
Data Structures: Rooted Trees

- A rooted tree, *T*, is a set of nodes which store elements in a parent-child relationship.
- *T* has a special node, *r* , called the *root* of *T*.
- Each node of *T* (excluding the root node *r*) has a *parent* node.



Rooted trees: terminology

- If node *u* is the *parent* of node *v*, then *v* is a *child* of *u*.
- Two nodes that are *children* of the same *parent* are called *siblings*.
- A node is a *leaf* (external) if it has no *children* and *internal otherwise*
- A tree is *ordered* if there is a *linear* ordering defined for the *children* of each internal node (i.e. an internal node has a distinguished first child, second child, etc).



Binary Trees

- A *binary tree* is a rooted ordered tree in which every node has *at most two children*.
- A binary tree is *proper* if each internal node has *exactly two children*.
- Each *child* in a binary tree is labeled as either a *left child* or a *right child*.

Tree ADT Methods

Tree ADT access methods:

- *root()*: return the root of the tree.
- *parent(v)*: return parent of *v*.
- *children(v)*: return links to *v*'s children.

Tree ADT query methods:

- *isInternal(v)*: test whether *v* is internal node.
- *isExternal(v)*: test whether *v* is external node.
- *isRoot(v)*: test whether *v* is the root.

Tree ADT Methods (cont.)

Tree ADT generic methods:

- *size()*: return the number of nodes in the tree.
- *elements()*: return a list of all elements.
- *positions()*: return a list of addresses of all elements.
- *swapElements(u,v)*: swap elements stored at positions *u* and *v*.
- *replaceElements(v,e)*: replace element at address *v* with element *e*.

Depth of a node in a tree

The *depth* of a node, *v*, is number of ancestors of *v*, excluding *v* itself. This is easily computed by a recursive function.

DEPTH(*T*,*v*)

if *T.isRoot*(*v*)

then return 0

else return 1 + DEPTH(*T*, *T.parent*(*v*))

Depth of a node in a tree

The *depth* of a node, *v*, is number of ancestors of *v*, excluding *v* itself. This is easily computed by a recursive function.

```
DEPTH(T,v)
    if T.isRoot(v)
    then return 0
    else return 1 + DEPTH(T, T.parent(v))
```

Height of a tree

The *height* of a tree is equal to the maximum depth of an external node in it. The following pseudo-code computes the height of the subtree rooted at *v*.

HEIGHT(*T*,*v*)

if ISEXTERNAL(*v*)
    then return 0
    else
        h = 0
        for each $w \in$ T.CHILDREN(*v*)
            do
                h = MAX(*h*, HEIGHT(*T*,*w*))
        return 1 + *h*

Tree Traversal

▶ In a *traversal*, the goal is for the algorithm to visit all the nodes in the tree in some order and perform an operation on them.

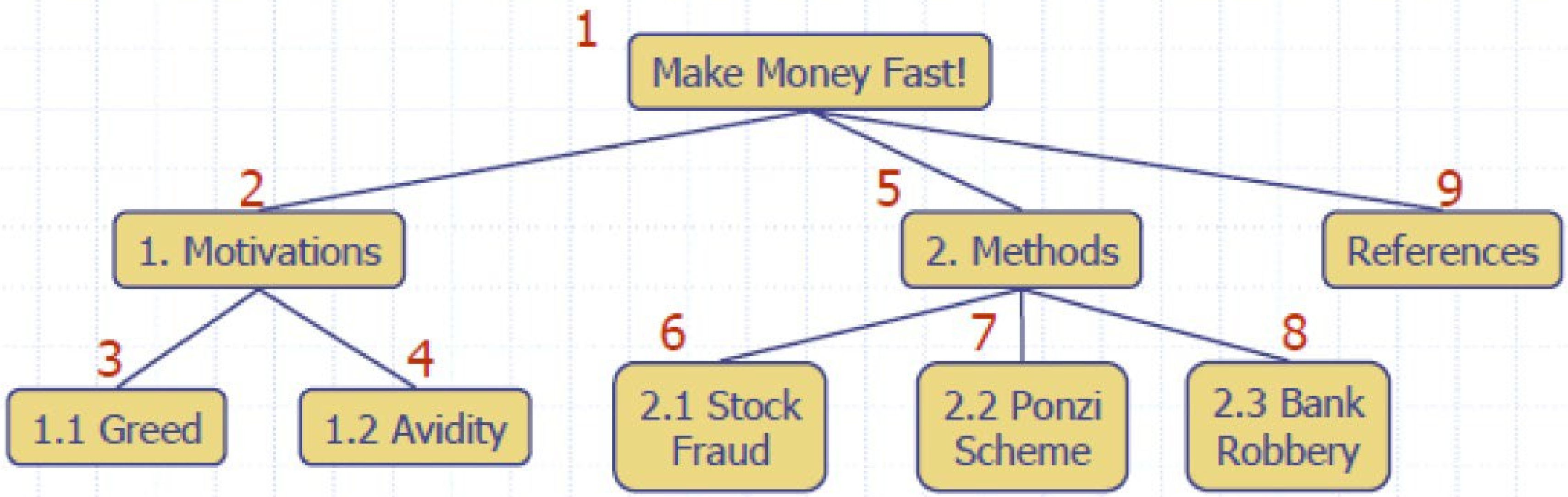
▶ Traversal and Searching

▶ Binary trees have three kinds of traversals: preorder, postorder, and inorder.

Preorder traversal in trees

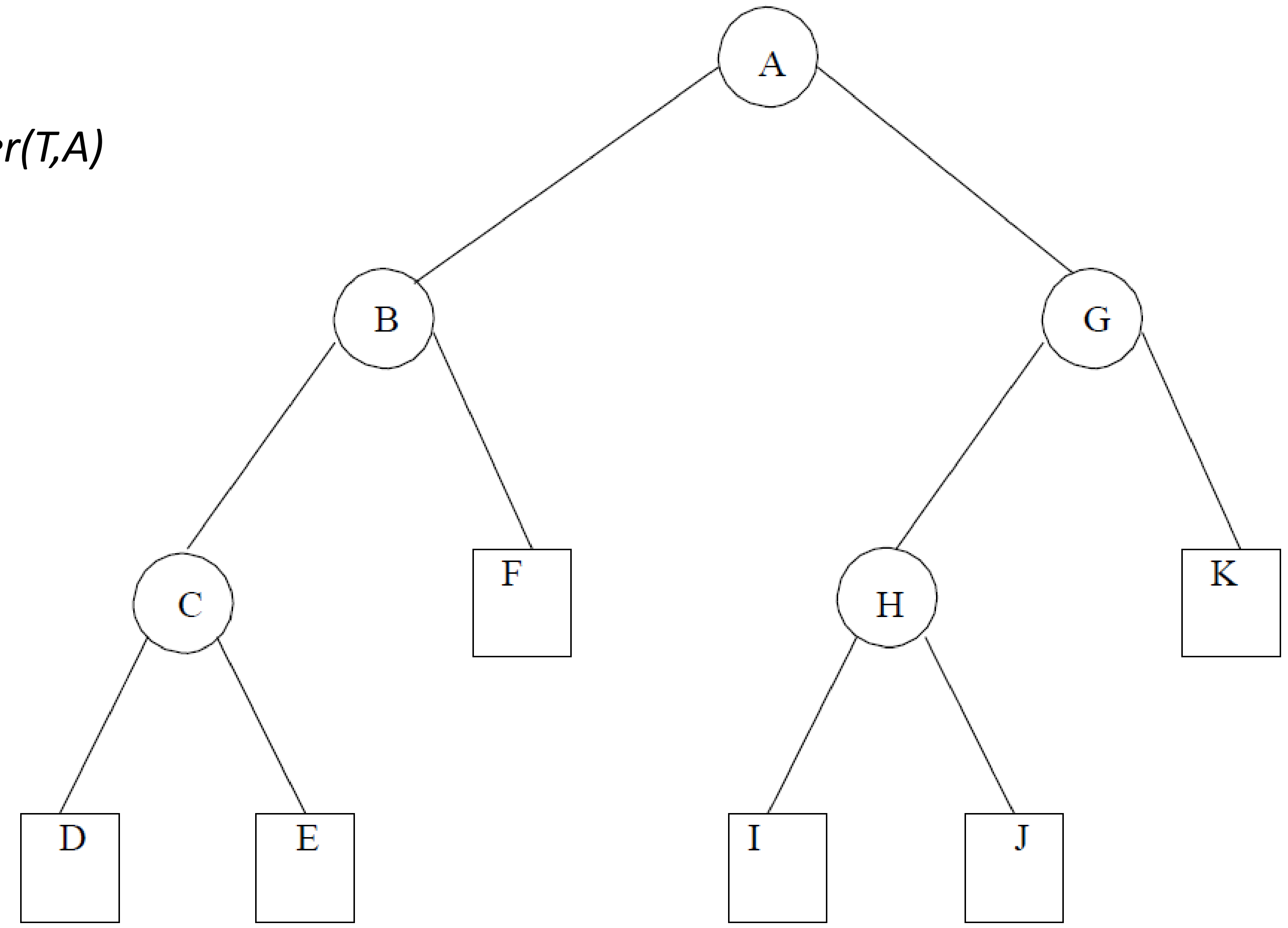
- A traversal visits the nodes of a tree in a systematic manner
- In a preorder traversal, a node is visited before its descendants
- Application: print a structured document

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

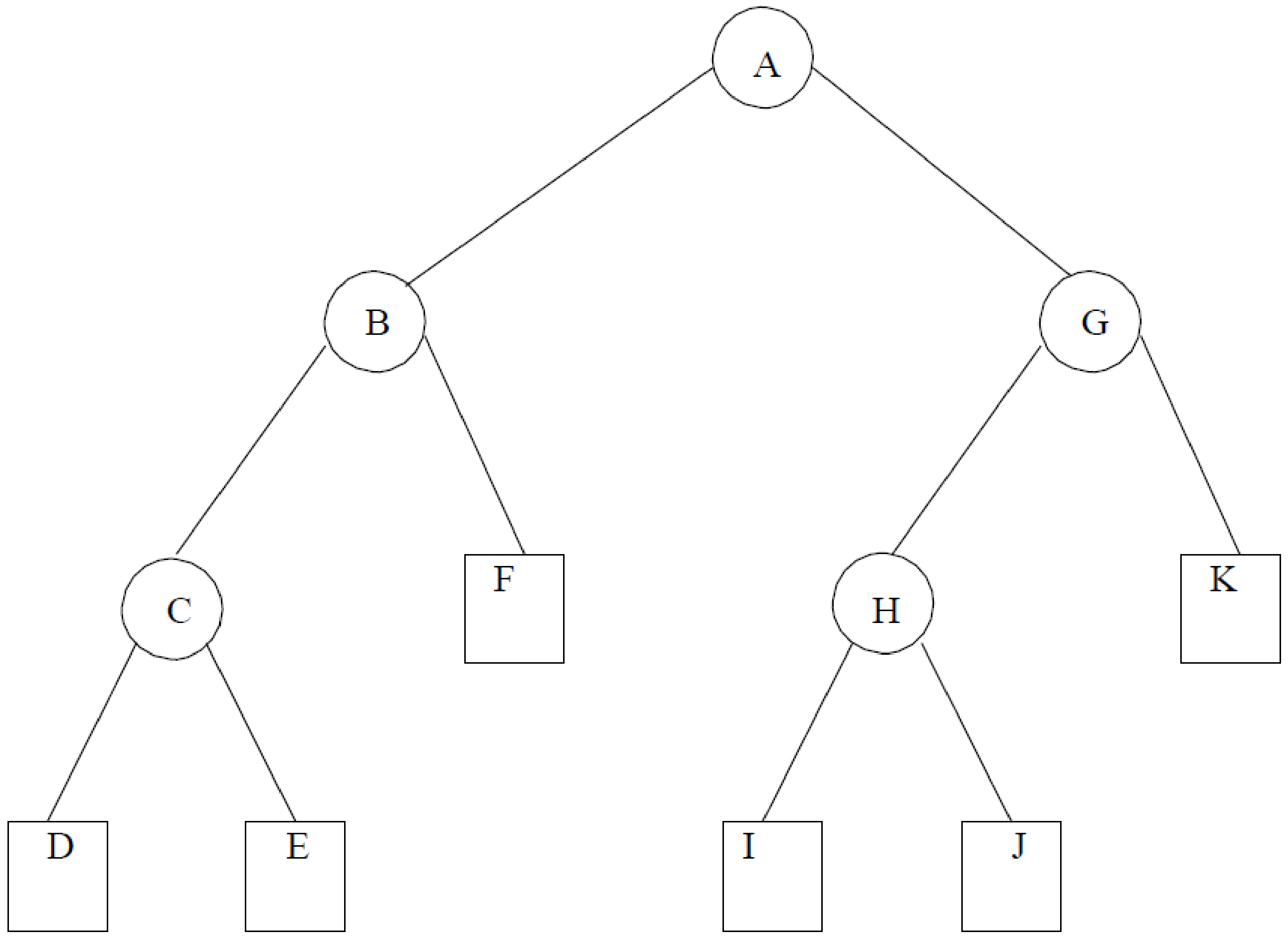


Preorder traversal in trees (cont.)

A call to *preorder(T,A)*



Preorder traversal in trees (cont.)

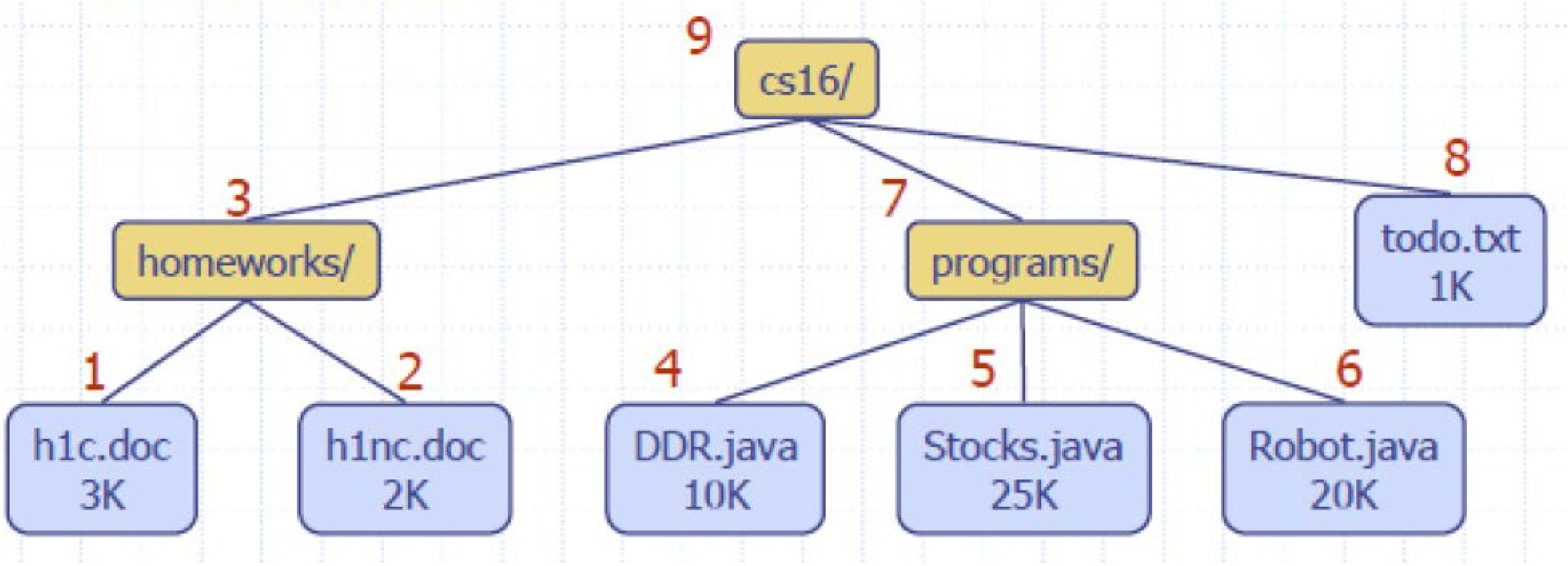


- A call to *preorder(T,A)* would produce: A,B,C,D,E,F,G,H,I,J,K.

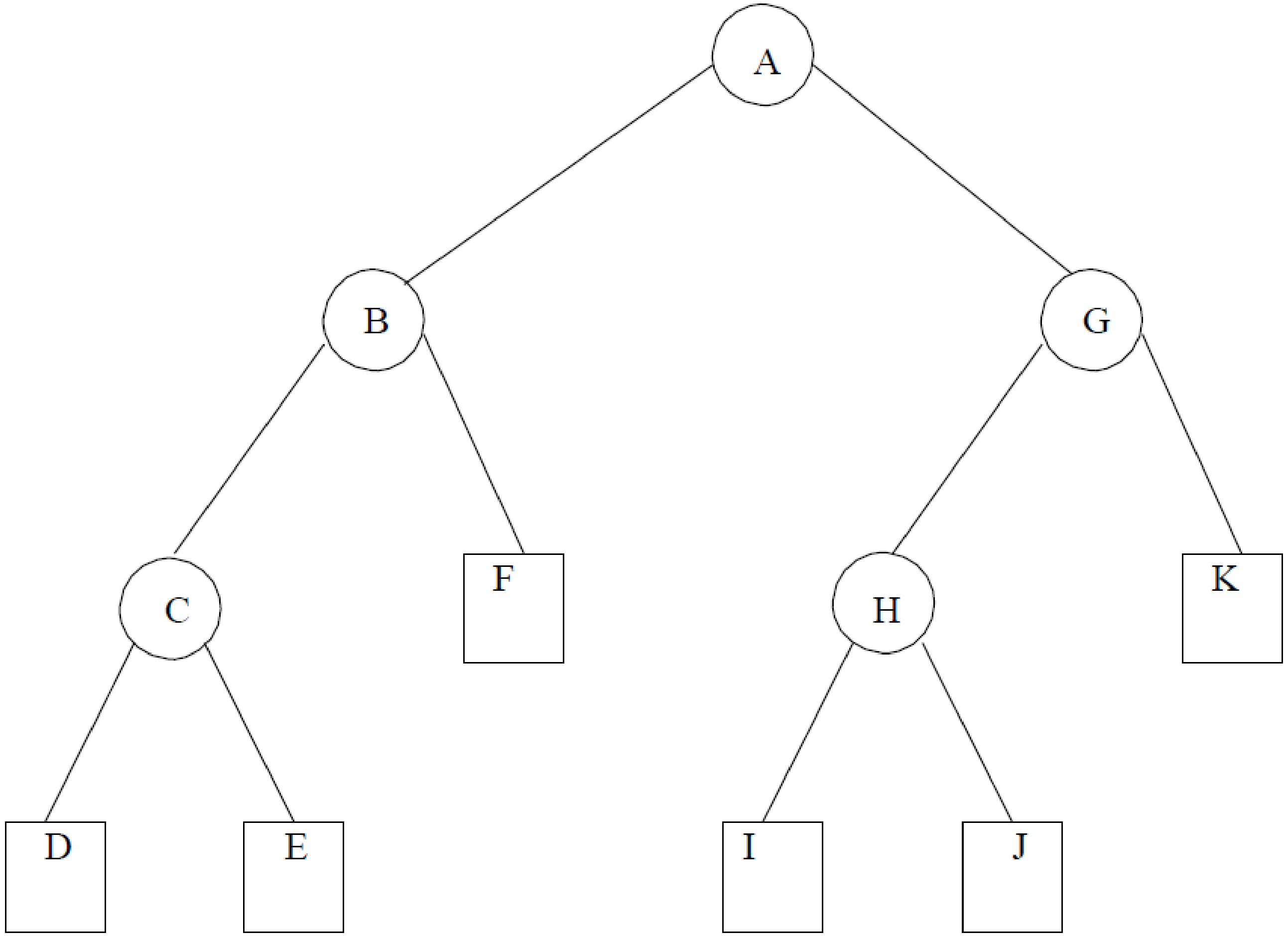
Postorder traversal of trees

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

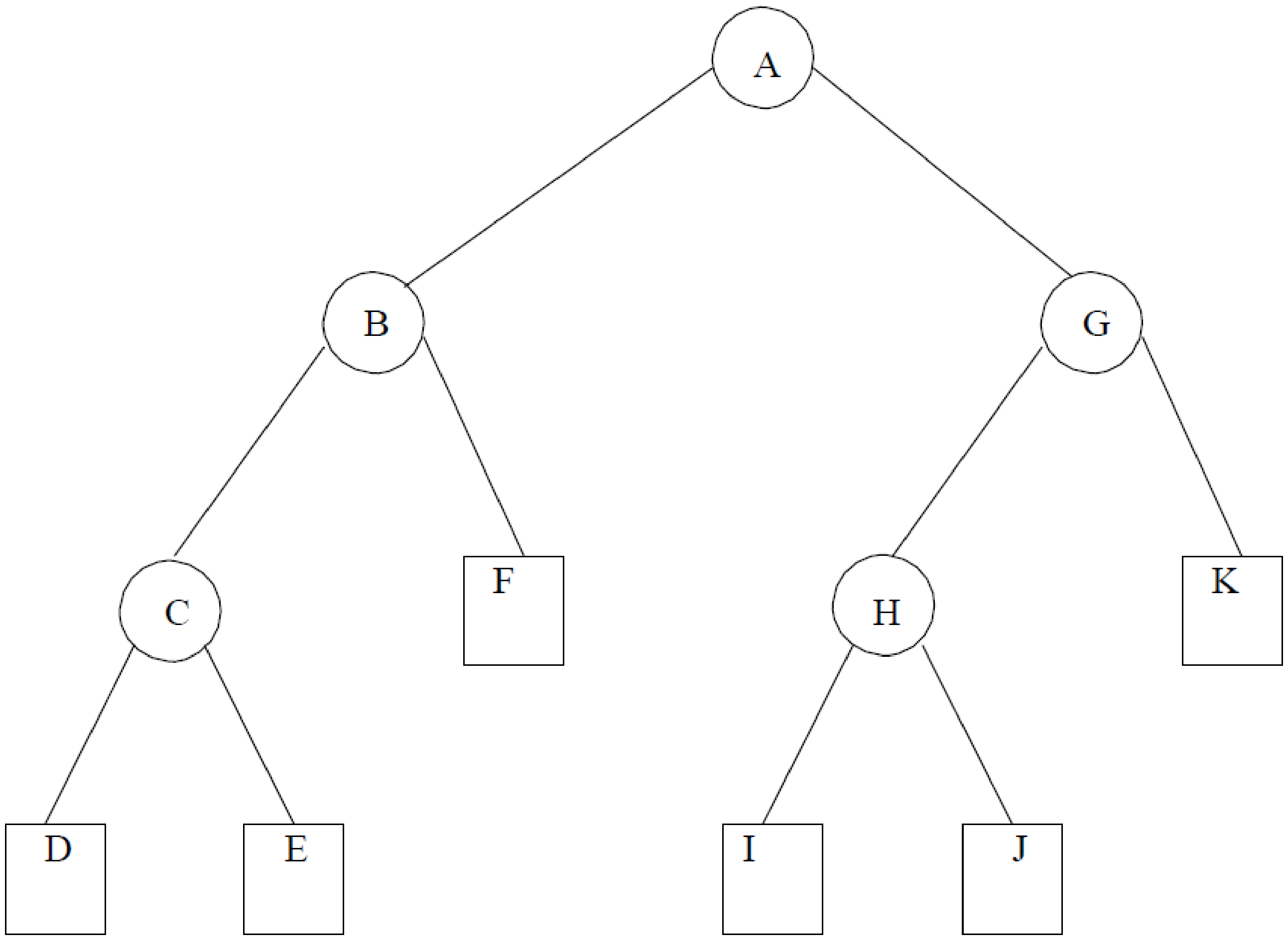
Algorithm *postOrder*(v)
  for each child w of v
    postOrder (w)
  visit(v)



Postorder traversal in trees (cont.)



Postorder traversal in trees (cont.)

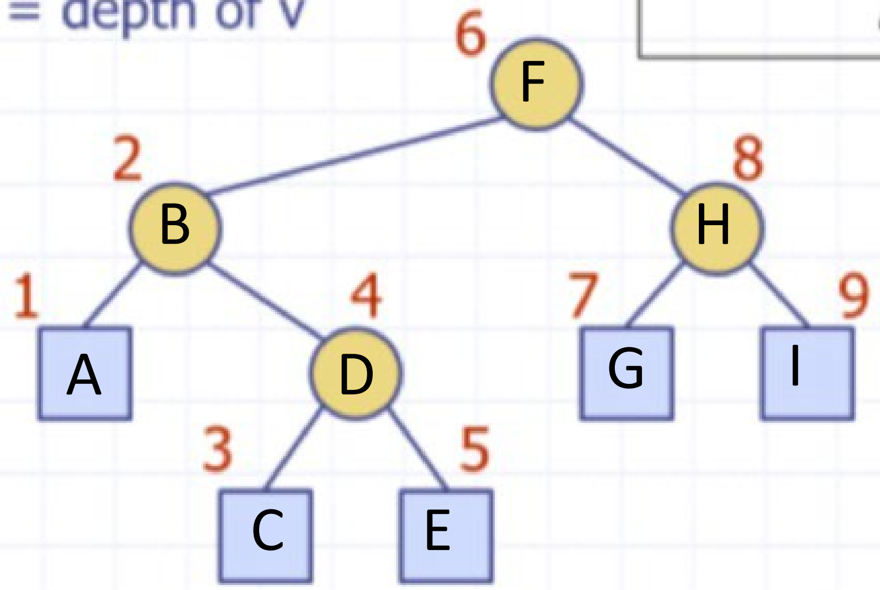


▶ A call to postorder(T,A) would produce: D,E,C,F,B,I,J,H,K,G,A.

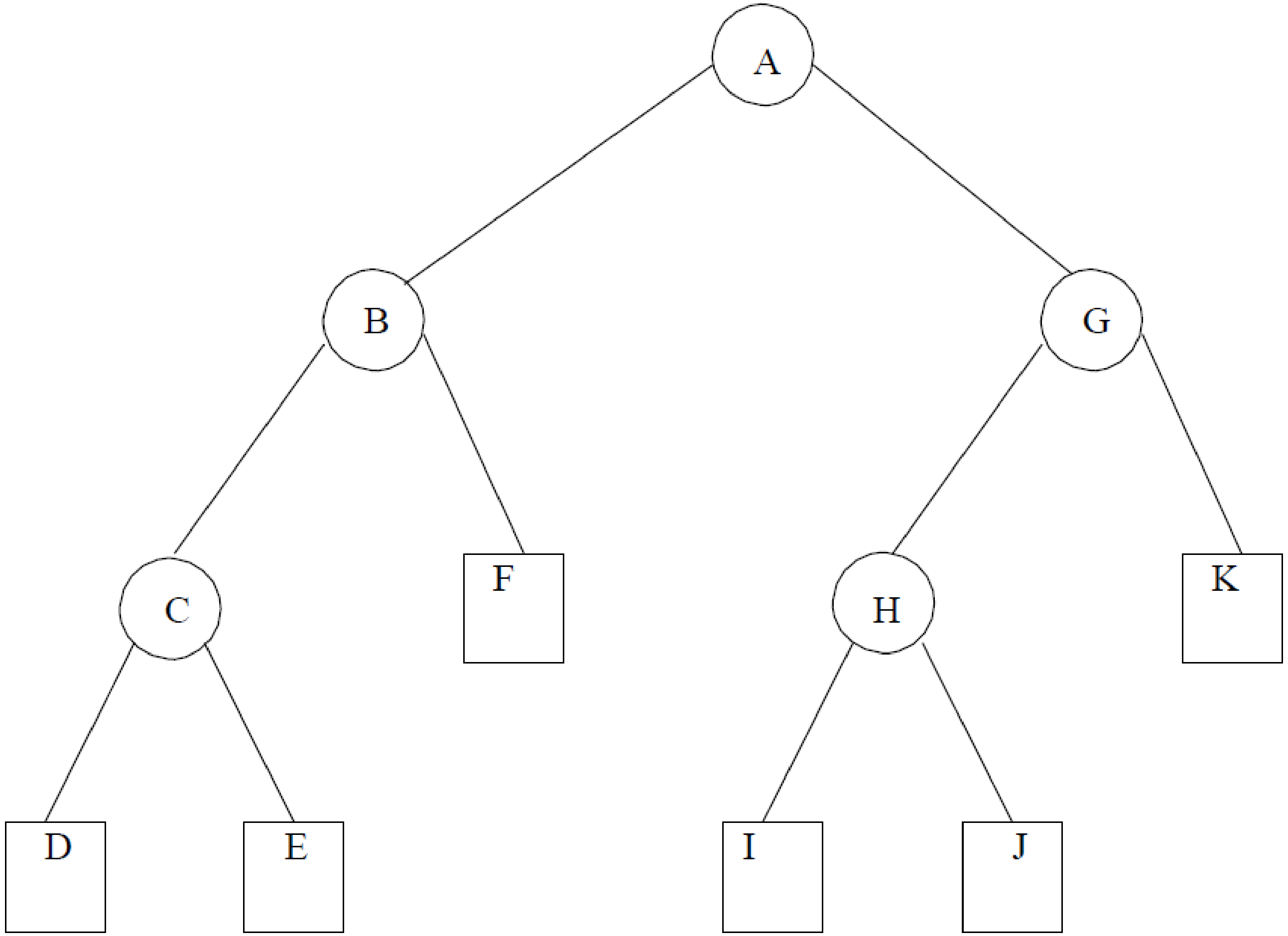
Inorder traversal in trees

- 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*))



Inorder traversal in trees (cont.)

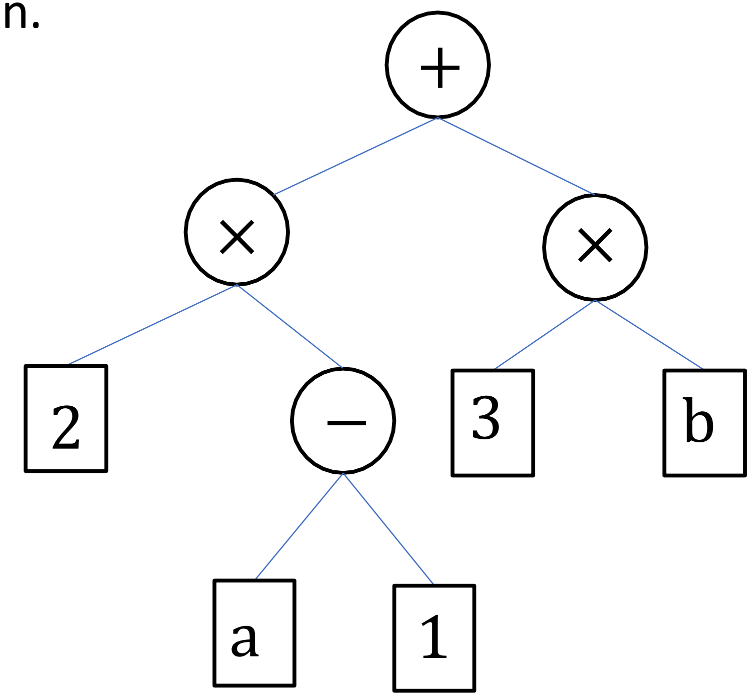


- A call to *inorder*(T,A) would produce: D,C,E,B,F,A,I,H,J,G,K.

Example: Parsing arithmetic expressions

Binary tree associated with an arithmetic expression

- Each *external node* is a *variable* or a *constant*.
- Each *internal node* defines an *arithmetic operation* on its two children.



Traversing the tree in inorder gives the valid *postfix* expression that represents this arithmetic calculation:

(2*(a-1)+(3*b))

Example: Parsing arithmetic expressions

Y=[(4+7)*6+(11-5)+3]*[(4+6)*8-3]