1.Data Strucures：Rooted Trees（有根树）

定义：A rooted tree, T, is a set of nodes which store elements in a parent–child relationship. 节点间有明显的层次关系。

T has a special node, r , called the **root** of T. 树的根节点（没有父节点）。

Each node of T (excluding the root node r) has a parent node. 除根节点外所有节点都有父节点。

相关术语：

子节点与父节点：If node u is the parent of node v,then v is a child of u .

同级节点：拥有同一个父节点的子节点。Two nodes that are children ofthe same parent are called **siblings** .

外部节点与内部节点：A node is a **leaf (external)** if it has no children and **internal** otherwise.

有序树：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). 即一个内部节点的不同外部节点区分开来。

Binary Trees（二叉树）：

A binary tree is a rooted ordered tree in which every node has at most two children. 每个节点最多有两个子节点。

A binary tree is **proper** if each internal node has exactly two children. 合格的二叉树每个内部节点都恰好有两个子节点。

Each child in a binary tree is labeled as either **a left child** or **a right child**.

2.Tree ADT Methods

访问方法：

- root(): return the root of the tree. 返回根节点。
- parent(v): return parent of v. 返回v的父节点。
- children(v): return links to v's children. 返回v的子节点。

查询方法：

- isInternal(v): test whether v is internal node. 判断是否为内部节点。
- isExternal(v): test whether v is external node. 判断是否为外部节点。
- isRoot(v): test whether v is the root. 判断是否为根节点。

其他方法：

- size(): return the number of nodes in the tree. 返回节点数。
- elements(): return a list of all elements. 返回树中所有的元素。
- positions(): return a list of addresses of all elements. 返回树中所有元素的位置。
- swapElements(u,v): swap elements stored at positions u and v. 交换元素。
- replaceElements(v,e): replace element at address v with element e. 取代元素。

Depth（深度） of a node in a tree and Height（高度） of a tree：

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

高度：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.

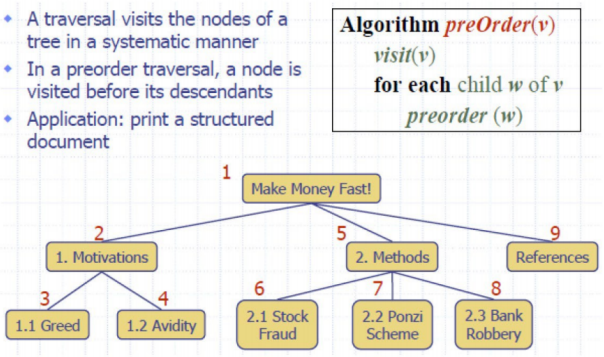
```
HEIGHT(T,v)
1 if ISEXTERNAL(v )
2    then return 0
3    else
4        h = 0
5        for each w ∈ T.CHILDREN(v)
6            do
7                h = MAX(h, HEIGHT(T,w))
8        return 1 + h
```

注意：深度是针对一个节点，高度是针对一个树，深度是从下往上，高度是从上往下。树的高度就是所有节点的深度中最大的那个。

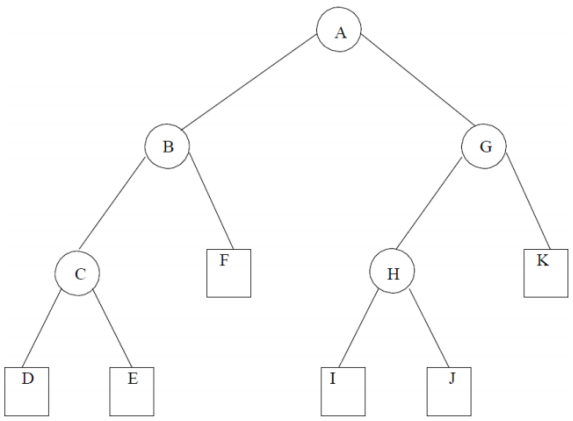
3.Tree Traversal（树的遍历）

遍历的目标：以某种特定的顺序访问树中的所有节点。二叉树有三种方式的遍历：

3.1 Preorder traversal in trees



类似深度优先搜索。首先访问根节点，然后递归的访问子节点。 例如：

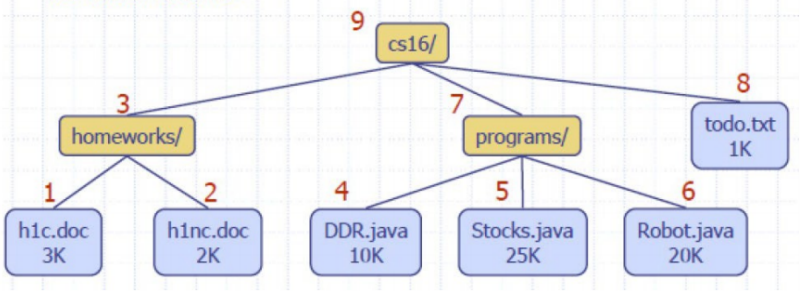


▶ A call to *preorder(T,A)* would produce: A,B,C,D,E,F,G,H,I,J,K.

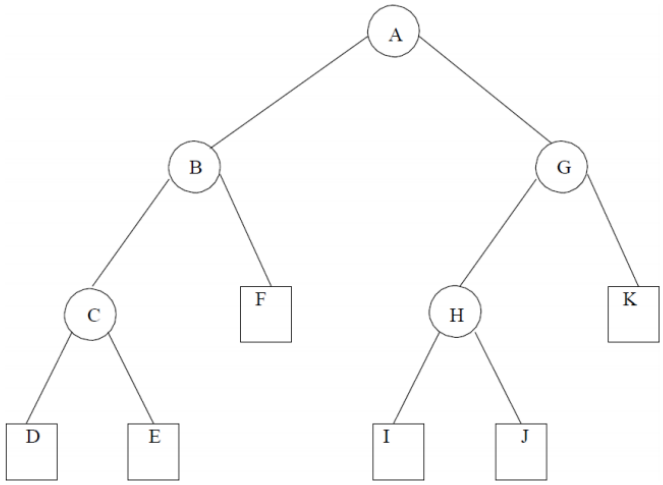
3.2 Postorder traversal of trees

- 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)
```



对于一个父节点的子节点来说，先遍历完所有的子节点再浏览父节点，然后去遍历下一个父节点的子节点，直到遍历完根节点为止。 例如：

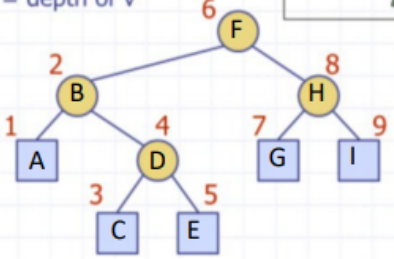


▶ A call to postorder(T,A) would produce: D,E,C,F,B,I,J,H,K,G,A.

3.3 Inorder traversal in trees

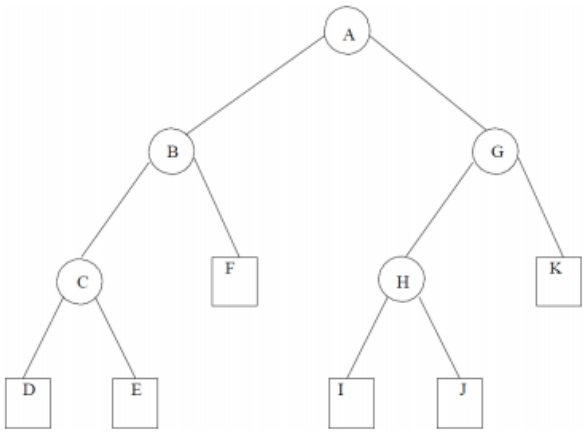
- In an inorder traversal a node is visited after its left subtree and before its right subtree
- Application: draw a binary tree
  - x(v) = inorder rank of v
  - y(v) = depth of v

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



这种方式的遍历按左子节点–根节点–右子节点的大致顺序。

首先遍历一个父节点是左子节点的节点，遍历完该节点后遍历该左子节点的父节点，然后转移到父节点的右子节点（A），然后依次遍历A的左子节点，A以及A的右子节点，重复上述步骤，或者从根节点开始建立子树来进行遍历。 例如：

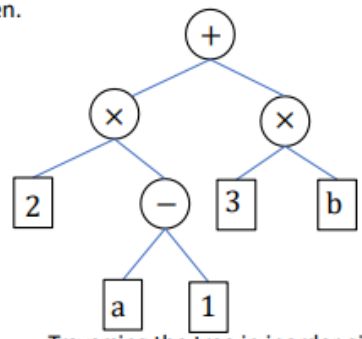


- A call to *inorder*(T,A) would produce: D,C,E,B,F,A,I,H,J,G,K.

实际应用：用于解析算数表达式（Parsing arithmetic expressions）

用二叉树的外部节点表示变量或者常数，内部节点表示操作符号。

- 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))

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