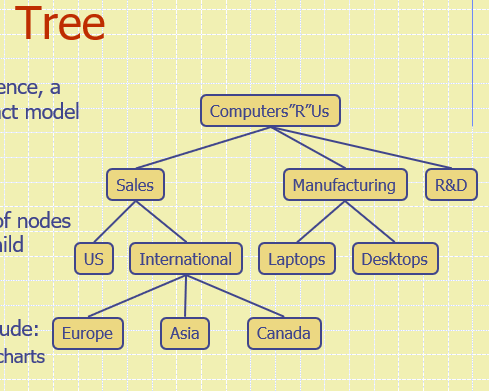
tree是一个抽象分层结构的模型

tree是由有父子关系的node组合而成的



Root :没有父辈的节点 (A)

internal node:有至少一个child 的node（A,B,C,F）

external node(leaf):没有children的node(E,I,J,K,G,H,D)

ancestor:一个node的parent,grandparent….

depth of a node:他所拥有ancestor的数量

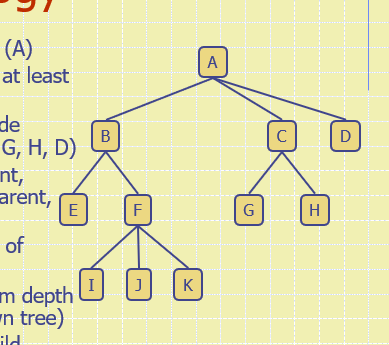
Height of a tree，任意node的最大depth，这个是3//因为只有三个ancestor，第四层是孙子

descendant of a node:他的子孙，儿子孙子啥的

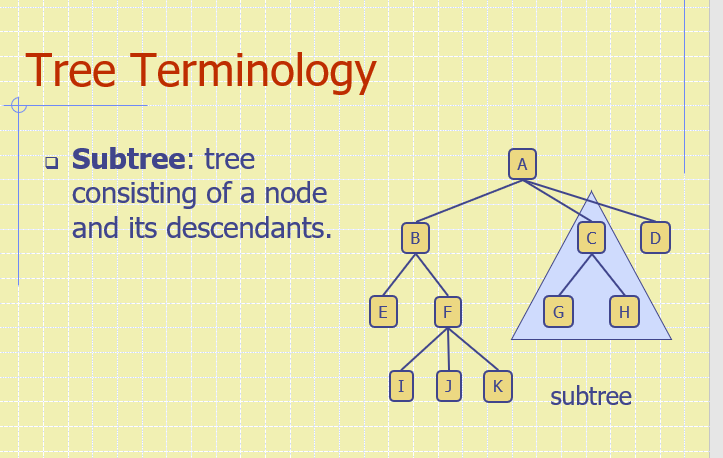
siblings:两个node parent相同

edge：一组node(u,v)，u是parent，v是child，或换过来。换句话来说是parent 与child之间的连接

path:一组node并且每两个node之间组成一条edge，例如A,B,F,J



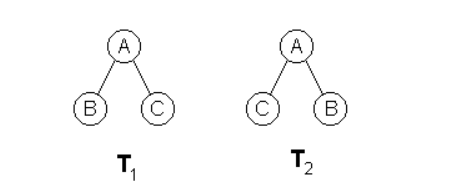
subtree:一个tree包含了一个Node和他的所有descendant



一个tree 是ordered如果给他的每个node的children都是linear ordering defined线性排序的

换句话来说,ordered tree可以分辨一个node的children们，通过first,second,third来区别是个那个child….

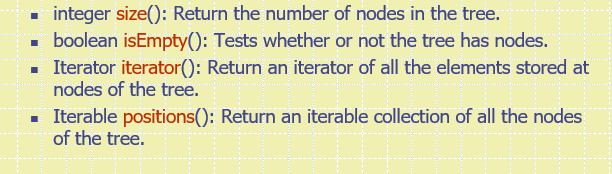
ordered tree 通常把siblings 按顺序线性排放来制定他们的order

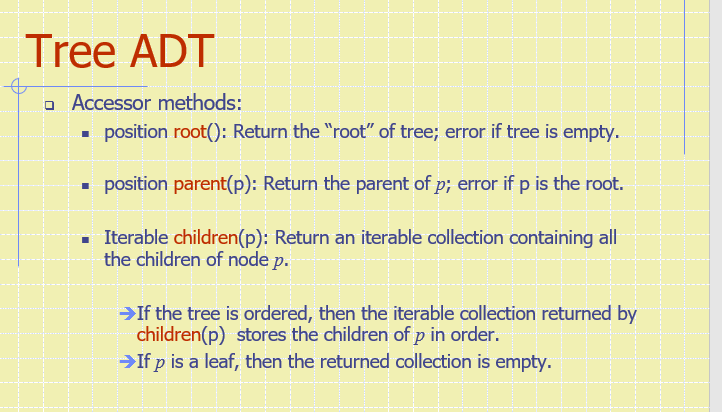


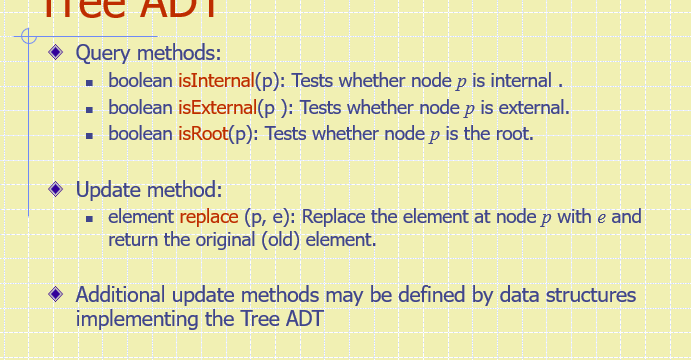
比如说这两个tree，如果是ordered tree，就不相等，第一个是B，第二个才是C,如果是unordered tree，就想等

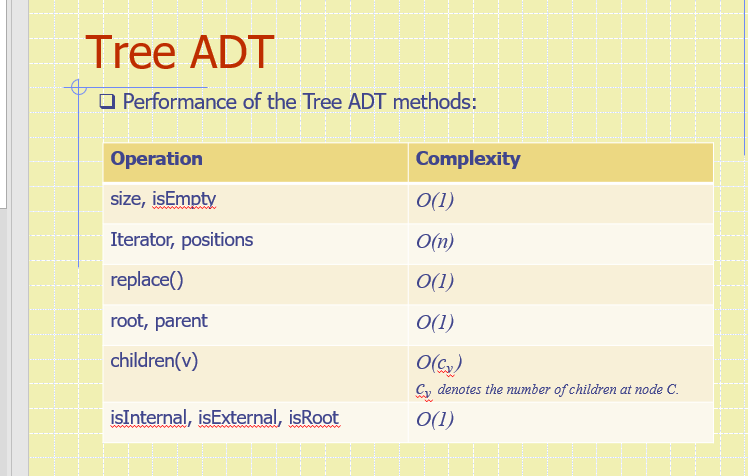
我们可以用positiion来抽象模仿node。一个tree的positions就是他的nodes.一个position object支持以下method







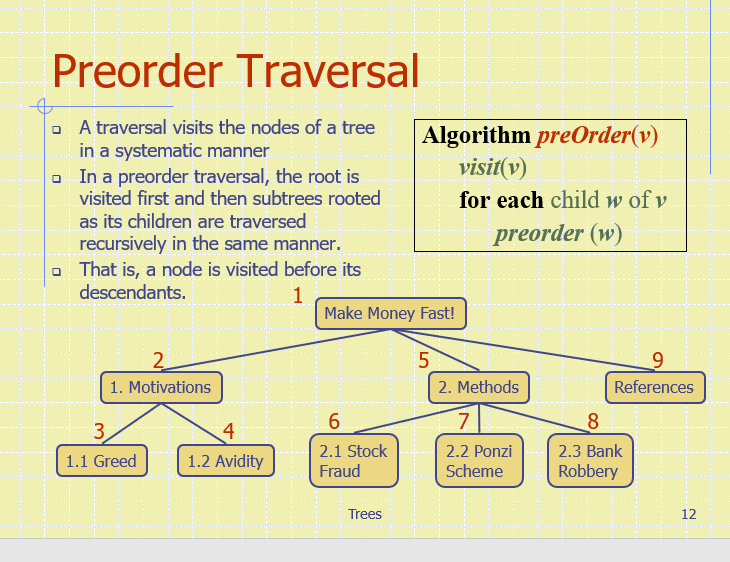




preorder traversal先序遍历

遍历有顺序的访问tree的nodes

先序遍历，先root，再subtree，subtree中先root再leaf

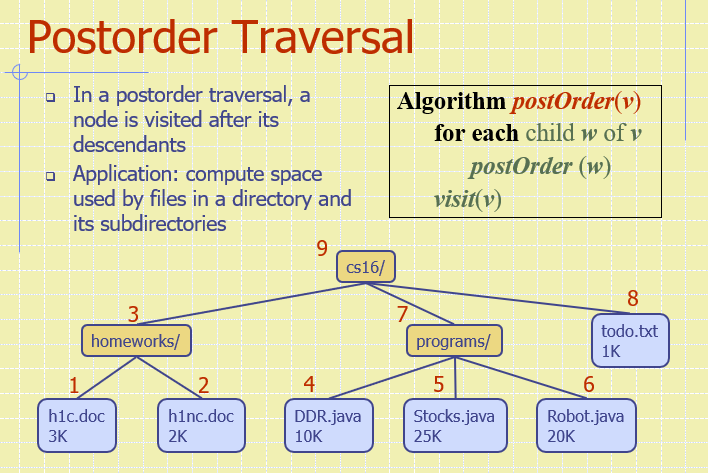


preorder会先visit当前Node，然后对他的每一个child进行preorder，

在child循环中，会先visit这个child root，然后preorder每个孙类，这时孙类已经是leaf了，preorder孙类相当于只有visit

Postorder traversal 后序遍历

先遍历他同级的，然后遍历他的父辈



post orderv 会Postorder cs16

对每个child of cs 16

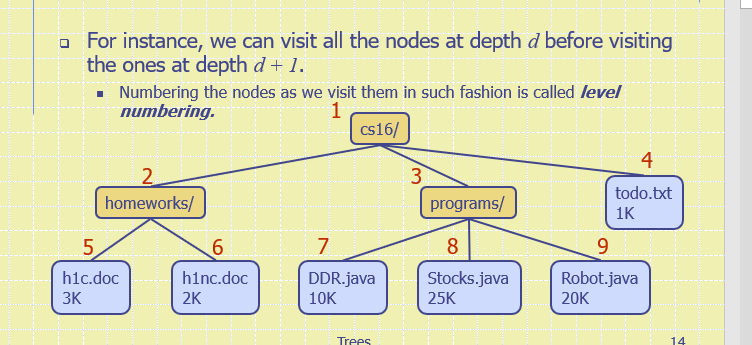
W1=HOMEWORKS 在W1循环中，会postorder 每个他的child也就是 1 2，然后visit w1也就是3

然后W2循环….

其余遍历法：

最常用的就是Preorder和Postorder

但也有其他traversal，比如一级一级depth遍历下来



binary trees 二叉树

每一个Internal node最多有两个children（正好两个叫做proper binary trees,不然叫做improper trees）

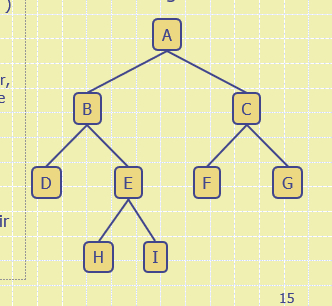
左边的叫left child右边的叫right child

一个node的children是ordered pair，左边的child优先于右边的child

一个binary tree可以被recursively 定义

base ：一个tree只有一个single node

recursive:一个tree的root有一组children，每一个都是一个binary tree

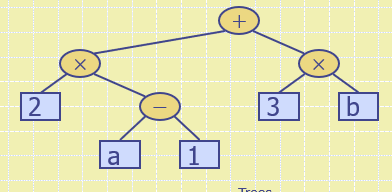


arithmetic expression tree，

binary tree 与数学四则运算相关联是一个proper binary tree

分作internal nodes:符号

external nodes:数字//最外端数字

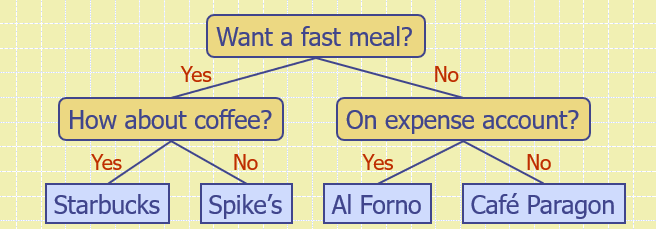


decision tree

binary tree与决定过程相关联

internal nodes:有着yes or no的问句

external nodes:所作出的决定



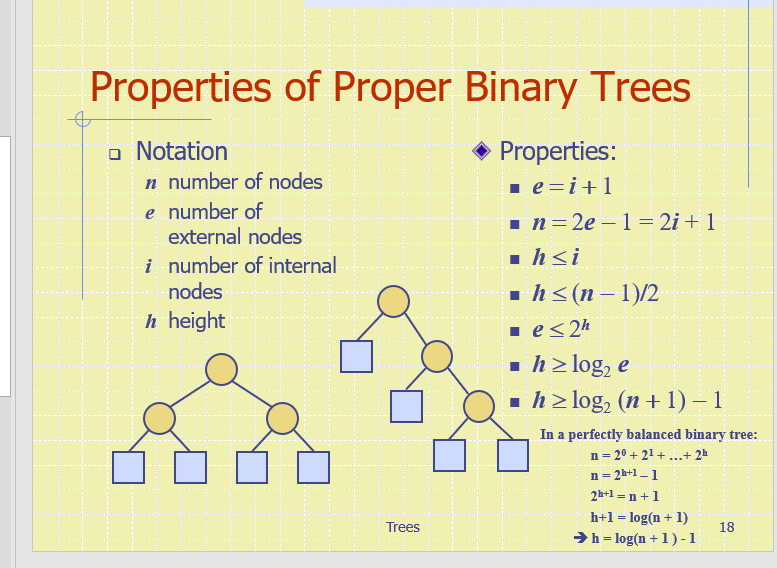
proper binary tree的性质

n代表node的总数

e代表external nodes

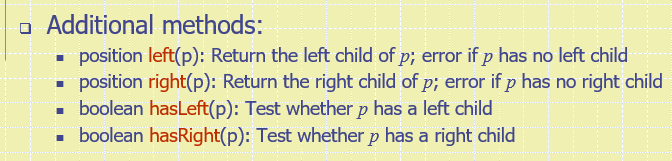
i代表internal nodes

h 代表Height



BinaryTree ADT

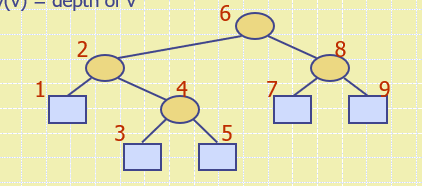
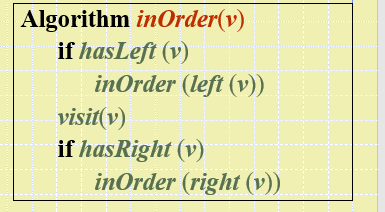
BinaryTree ADT extends Tree ADT, 所以，他继承了Tree ADT的所有methods



inorder traversal of bineary tree 中序遍历

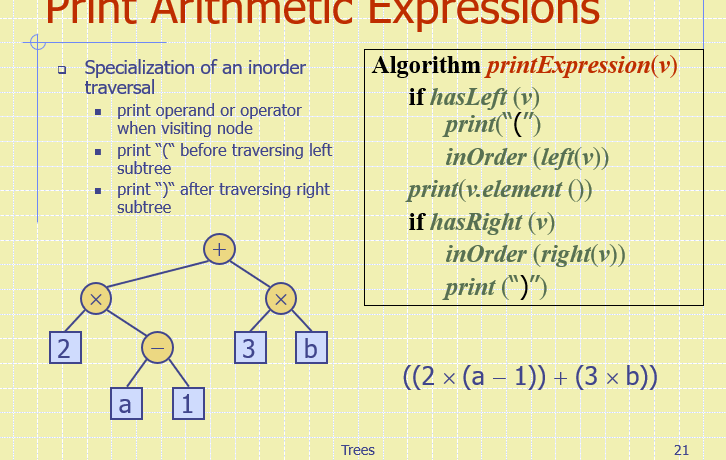
先visit他的left subtree，再visit这个node本身，在visit他的right subtree

注意先从最左下角开始，如果一个node有left，就left-node-right

其中一个用法就是画一个binary tree

print 一个arithmetic expression



在visit的时候print符号或数字

在遍历左subtree之前print”(“

在便利右subtree之后 print”)”

如果v有left，print“（”

然后遍历左边，写下V的element，

如果有right，遍历右边

print“）”

换句话说inOrder当做没解决的部分

就是(LEFT ELEMENT RIGHT)

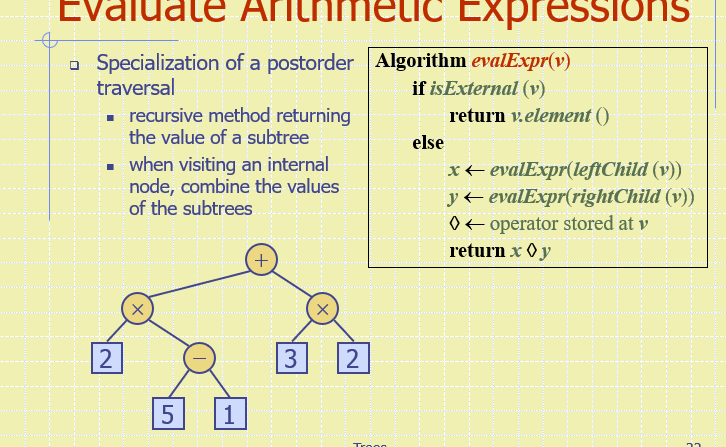
运行加减法：

如果已经是数字了，就return 这个数字

如果不识数自己，就把X设置成他左边的evalexpre，Y设置成右边的EVALEXPRE，符号设置成v的符号

return x<>y，

有时候左边的仍然不是数字，就会循环，直到is external，最后总会成为两个external加一个数字的组合

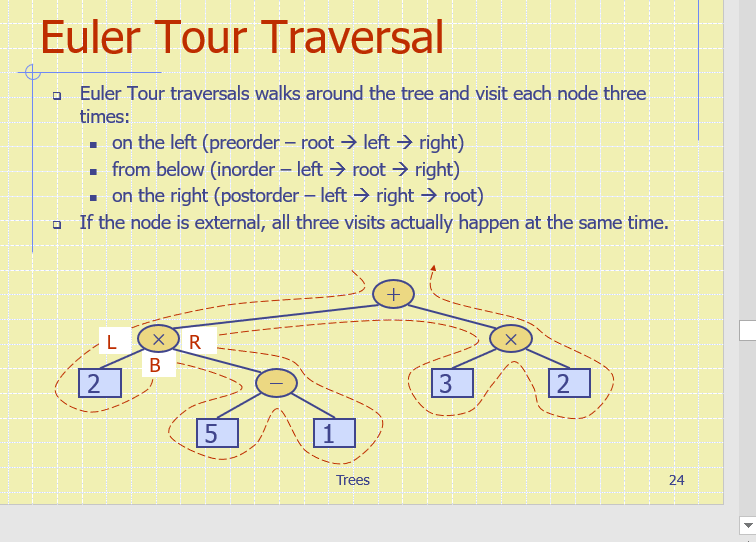


euler tour traversal 欧拉遍历

之前的traversal算法都是iterator的形式，每一次遍历都是visit一个具体node以一个固定顺序，并且只visit一次

euler 放宽了对每个node只visit一次的要求//有可能visit一个node 多次

欧拉遍历给所有二叉树提供了一个通用遍历，不管是preorder,postorder还是inorder



节点左边，就preorder， root left right，

节点下面 就 inorder left root right

节点右边就Postorder left right root

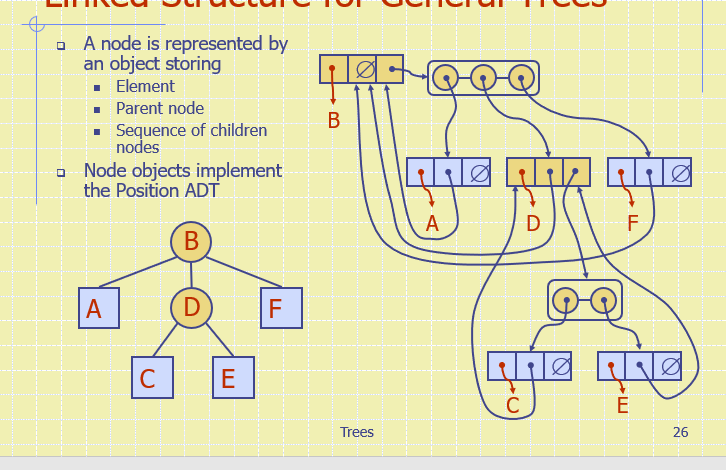
Euler tour 每一个visit用时为 O(1)

因为我们每个NODE所用时间相等，总共时间O（n）

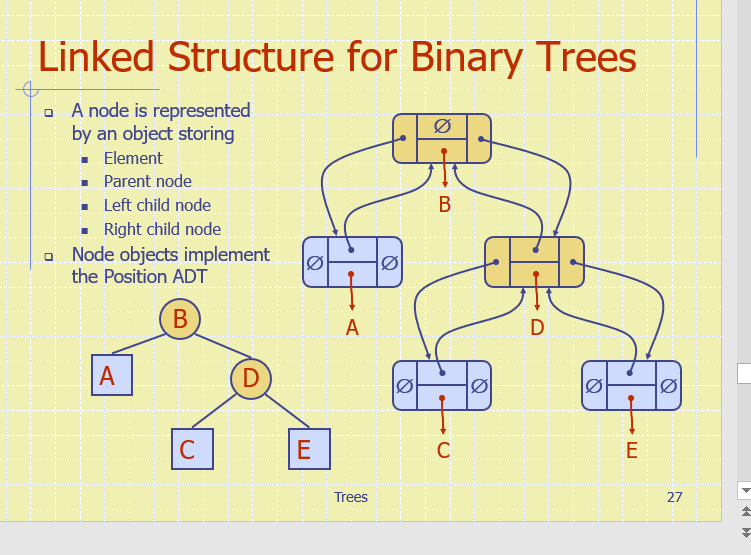
一个标准tree的Linked Structure

一个Node通过存储着element,parent node, sequence of children node 的object来表示

第一个格子存储着他是谁，第二个格子存储着他的parent，第三个格子存储的是他的children sequence



Binayrr tree的 linked structure

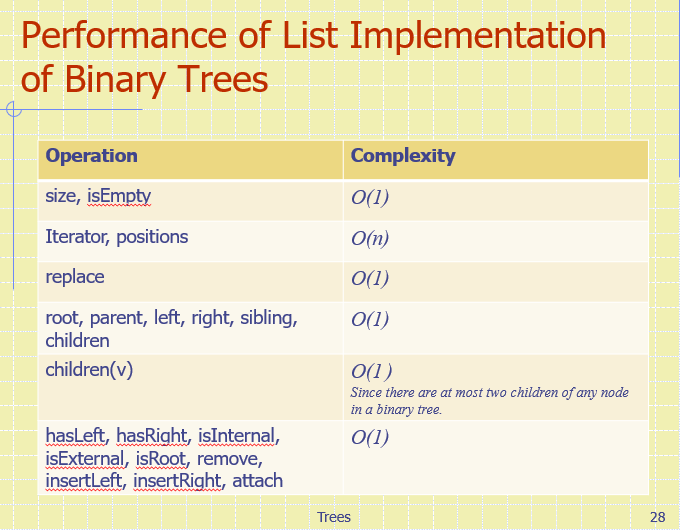


一个Node由一个存储着element,parent node，left child node, right child node的object表示

由Position ADT构建node objects

最上面的存着爹link，下面的存着自己position

左边left node，右边right node



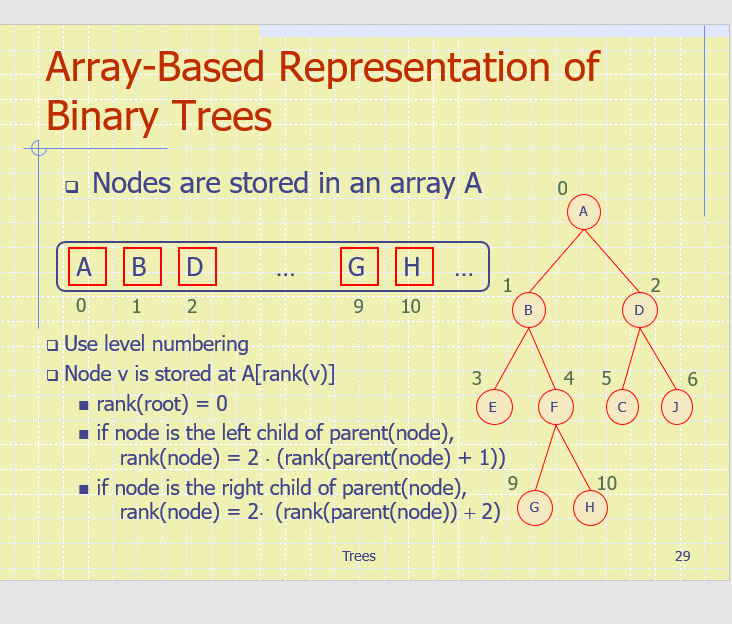
binary tree也可以是array based

node V被存储在A[rank(v)]里

root的rank=0

node是左边，他的rank=parent的rank\*2+1

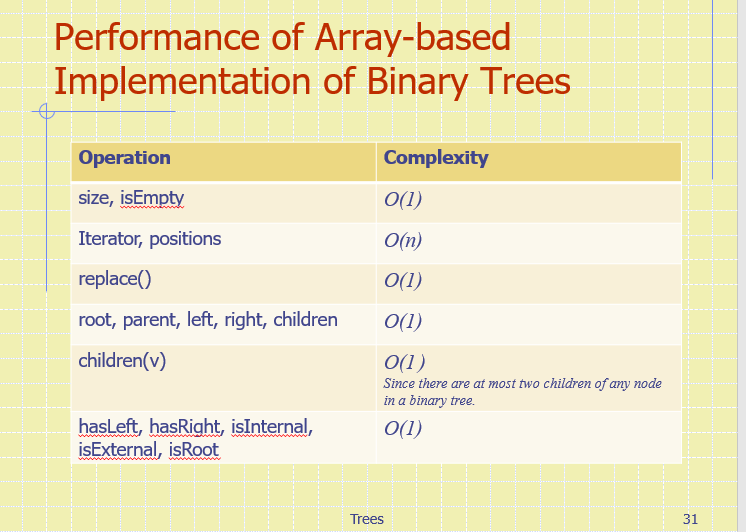
node是右边，他的rank=parent的rank\*2+2



可以看见左边的永远是父辈的2\*+1，右边的永远是父辈的\*2+2

没有78，那就跳过

node之间的Link没有被存储，因为默认了存储方式‘



’