第11章　二叉树和其他树

----一种具有分支层次关系的数据结构

计算机学院

主要内容

- **树的一般定义**
- 二叉树的定义和操作
- 二叉树的遍历

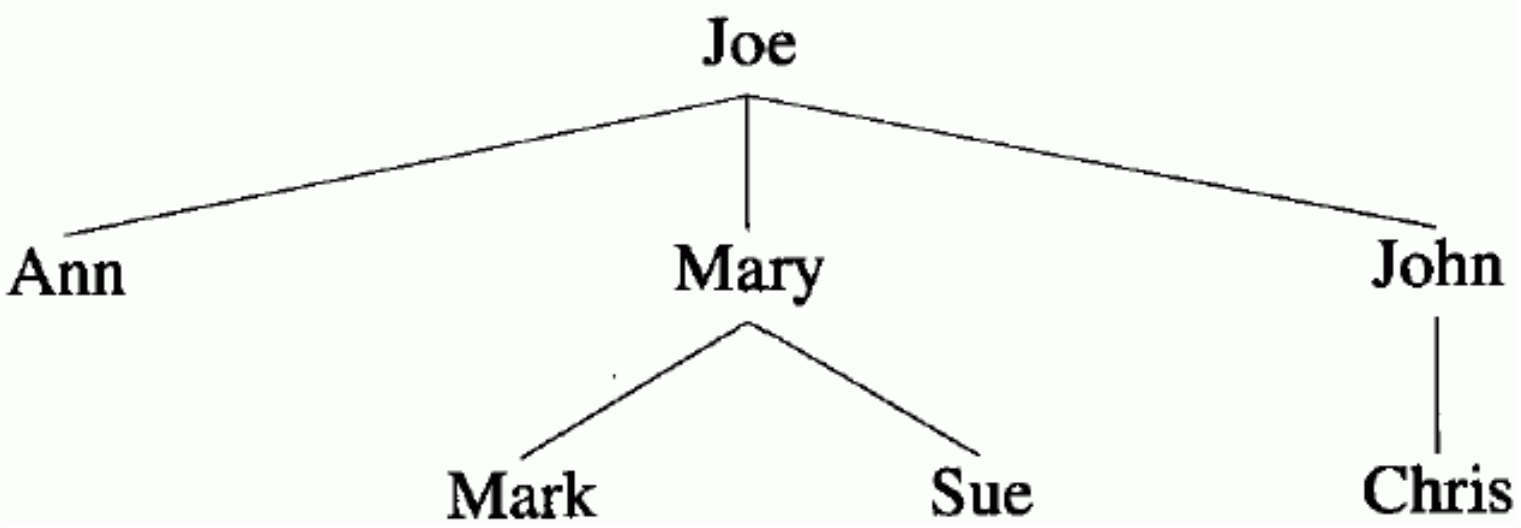
树

- 线性表、表：不适合描述**层次结构**数据
- 祖先－后代、上级－下属、整体－部分

很多事物具有非线性特征，如何描述？

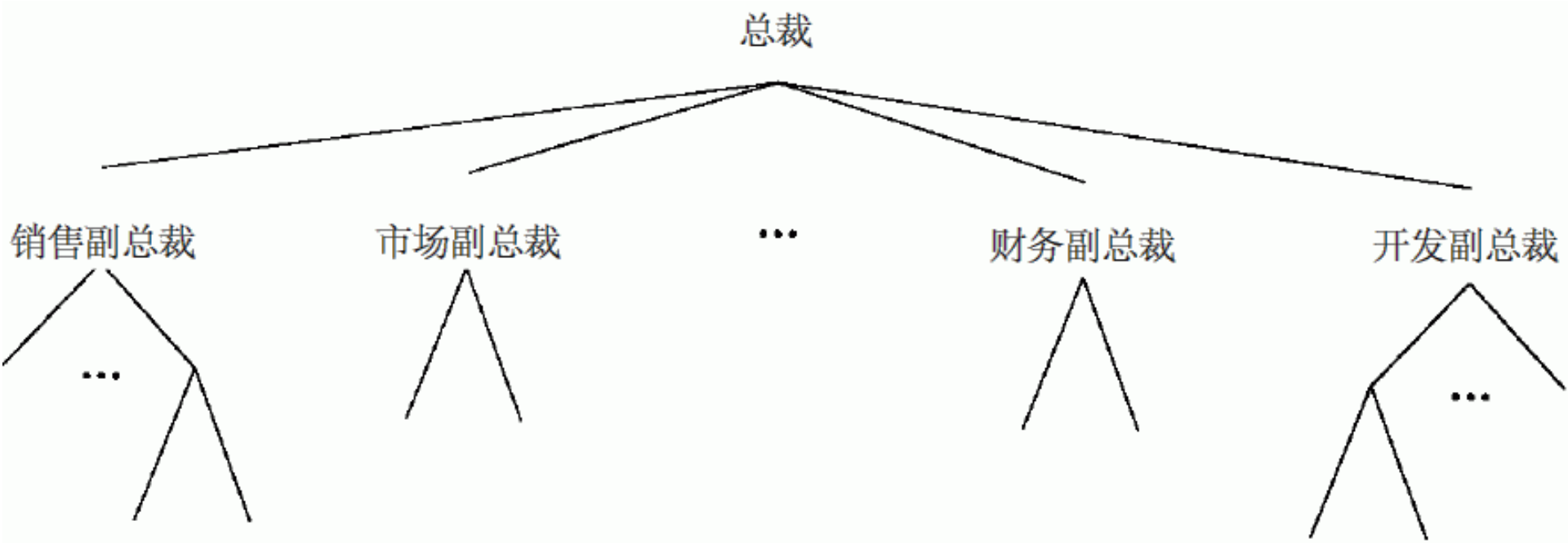
模仿自然界中的**树**！

例11-1：家庭关系



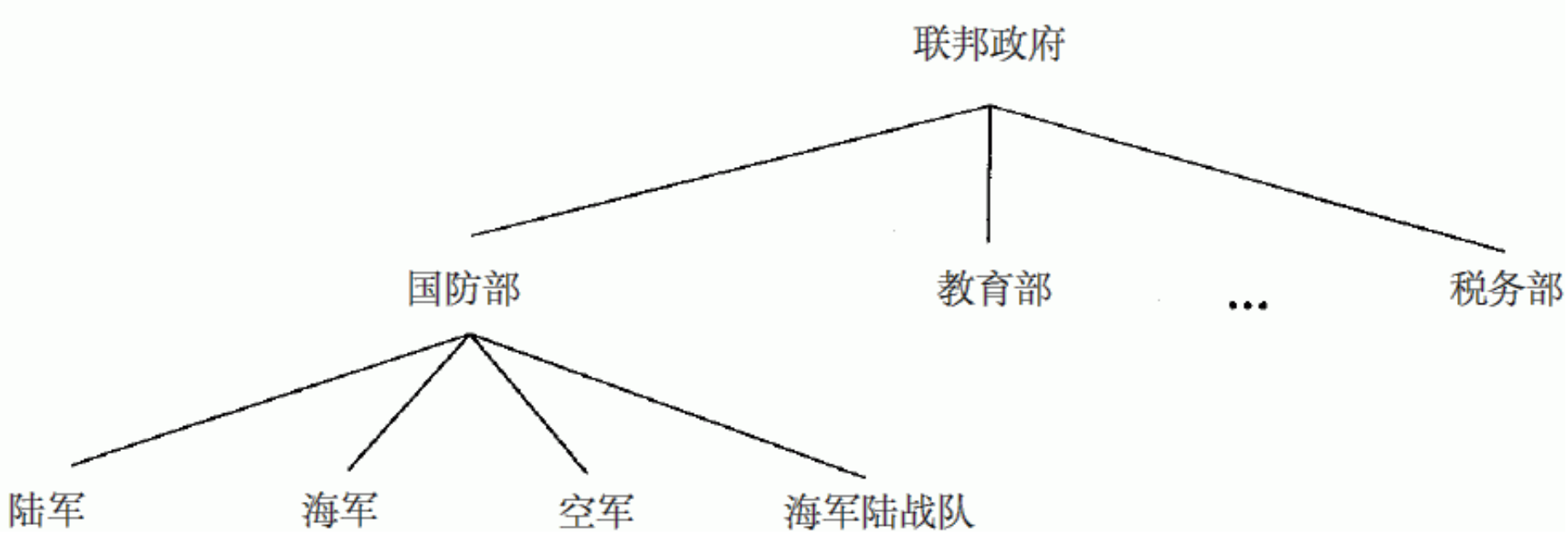
例11-2

- 公司结构



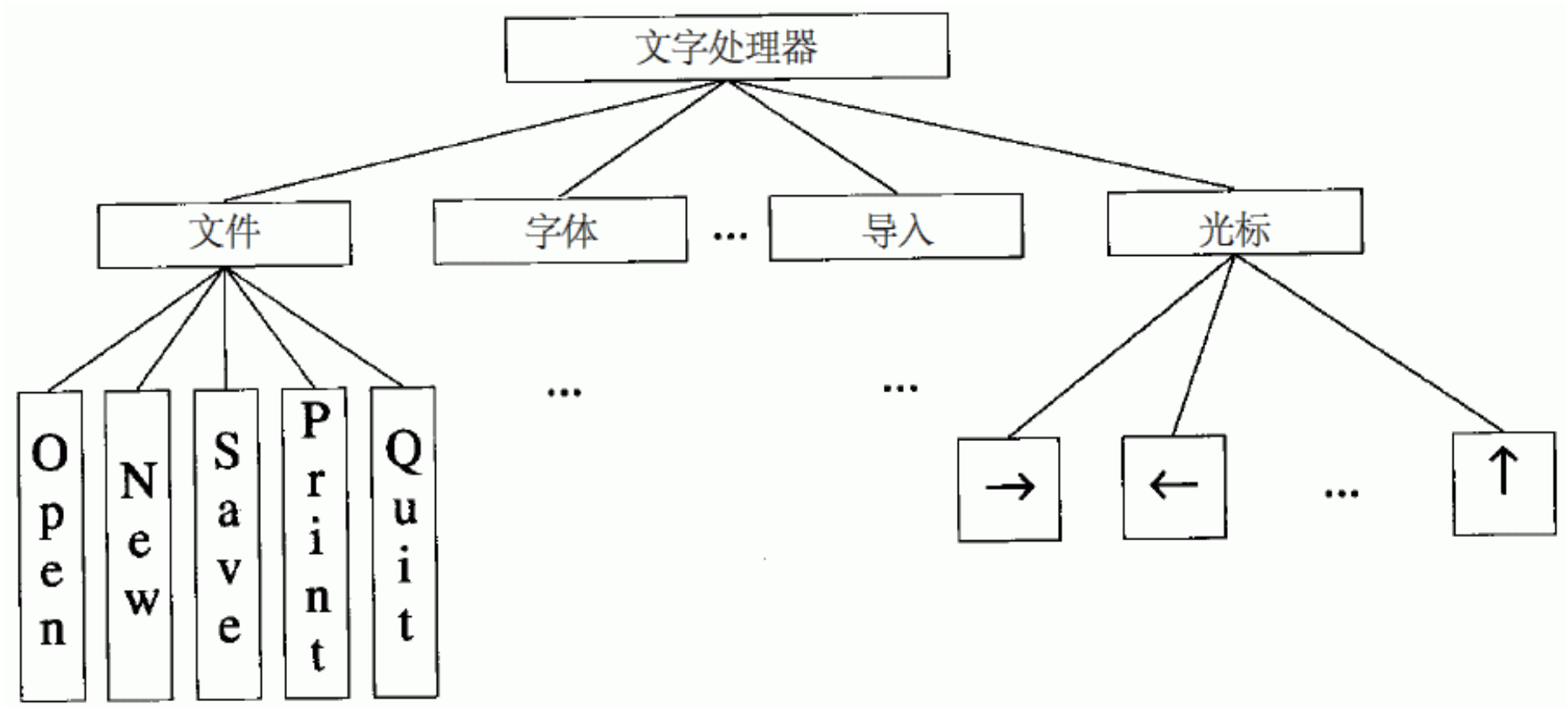
例11-3

- 政府机构



例11-4

- 软件工程



南开大学研究生院
Graduate School of Nankai University

设为首页
加入收藏
联系我们

首页 home | 本院简介 introduction | 新闻与公告 news | 招生工作 admissions | 培养工作 cultivation | 教育管理 administration | 学科学位 degrees | 211工程 211project | “985”计划 985project | 校园生活 campus

招生工作 admissions

- 学历教育招生
  - 招生信息
  - 各院系所联系方式
  - 博士生招生
  - 硕士生招生
  - 港澳台及留学招生
- 非学历教育招生
- 导师信息

招生信息

- 2012年南开大学博士研究生入学考试复试确认系统已经开通 [2012-4-12]
- 南开大学2012年博士研究生入学考试复试分数线 [2012-4-11]
- 2012年博士复试相关通知 [2012-4-9]
- 研招办已经收到的硕士研究生调剂申请表传真件 [2012-4-16]
- 2012年南开大学少数民族高层次骨干人才计划硕士复试说明 [2012-3-31]
- 2012年南开大学少数民族高层次骨干计划和西部计划硕士研究生复试分数线 [2012-3-31]
- 南开大学2012年博士研究生初试成绩及排名查询网址 [2012-3-30]
- 2012年硕士入学考试考生调剂（调出）流程 [2012-3-30]
- 2012年全国硕士研究生统一入学考试复试分数线划定 [2012-3-30]
- 关于领取2012年博士研究生入学考试报名费收据的通知 [2012-3-15]
- 南开大学2012年博士研究生报名现场确认安排 [2012-3-13]
- 南开大学2012年硕士研究生复试办法 [2012-3-9]
- 关于少数民族高层次骨干计划的说明 [2012-3-9]
- 2012年南开大学硕士研究生入学考试复试网上确认系统开通 [2012-3-8]
- 关于2012年硕士研究生入学考试复试网上确认的说明 [2012-3-6]
- 有关复试的相关安排和要求待定 [2012-3-5]
- 2012年南开大学硕士研究生复试分数线 [2012-3-5]

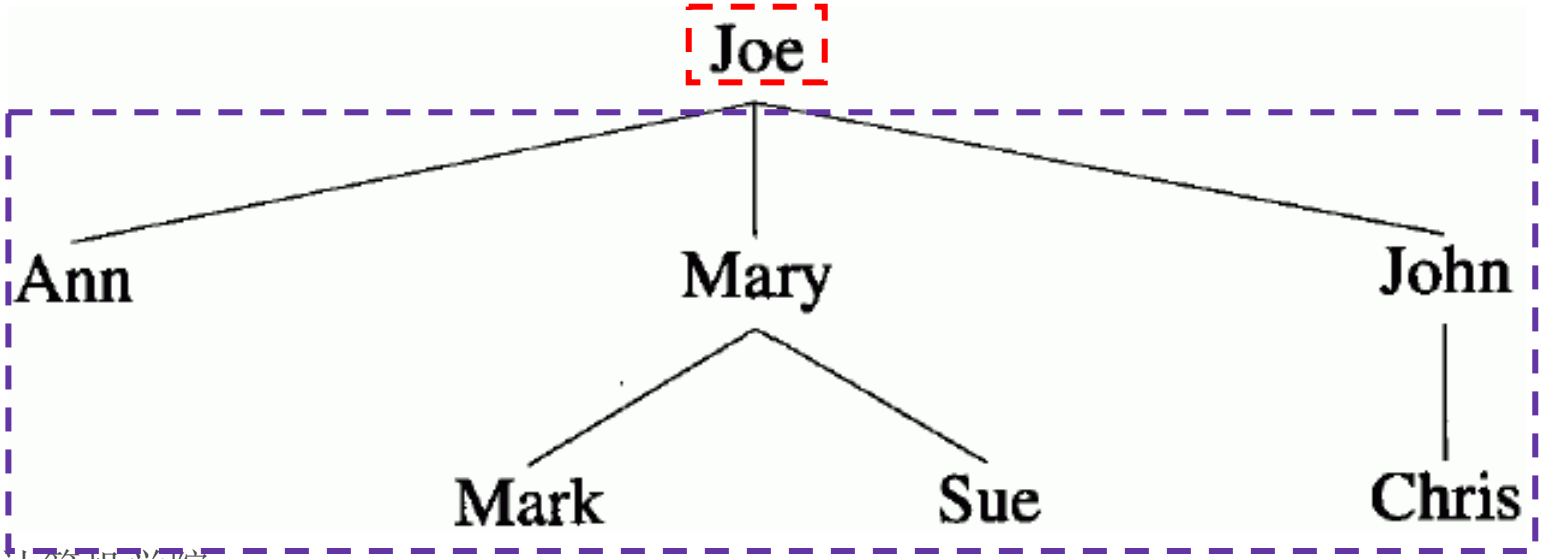
数据结构“树”

- **定义**：
  树（tree）t是一个非空的有限元素的集合，
  一个特殊的元素称为**根**（root），
  余下的元素（如果有的话）组成t的若干**子树**（subtree）

- 递归！

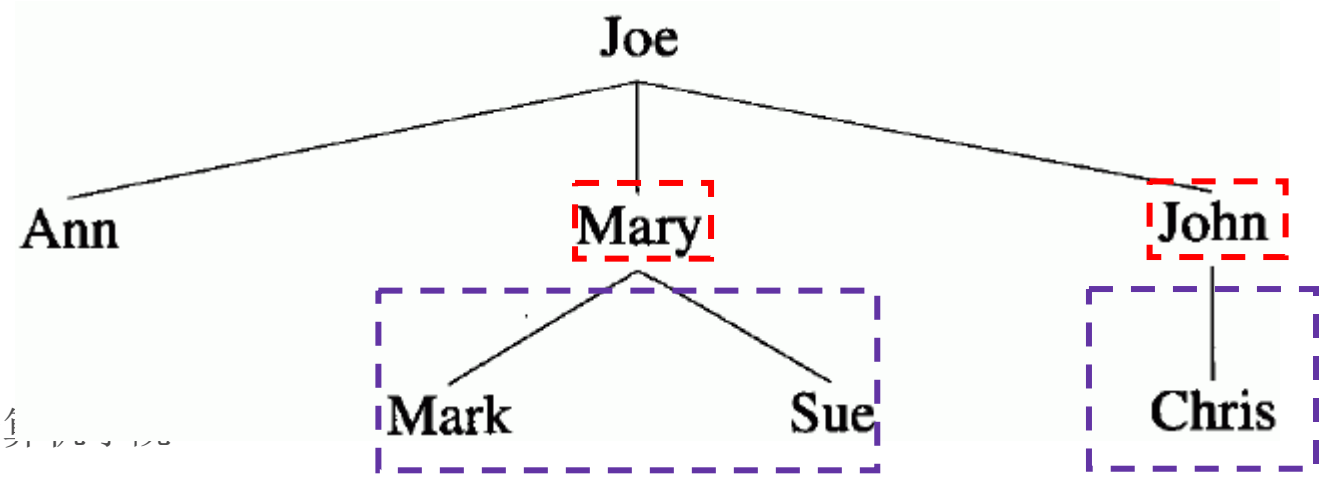
例11-5 家庭关系例树的描述

- 数据集合
  {Joe, Ann, Mary, Mark, Sue, John, Chris}
- n=7，根为Joe

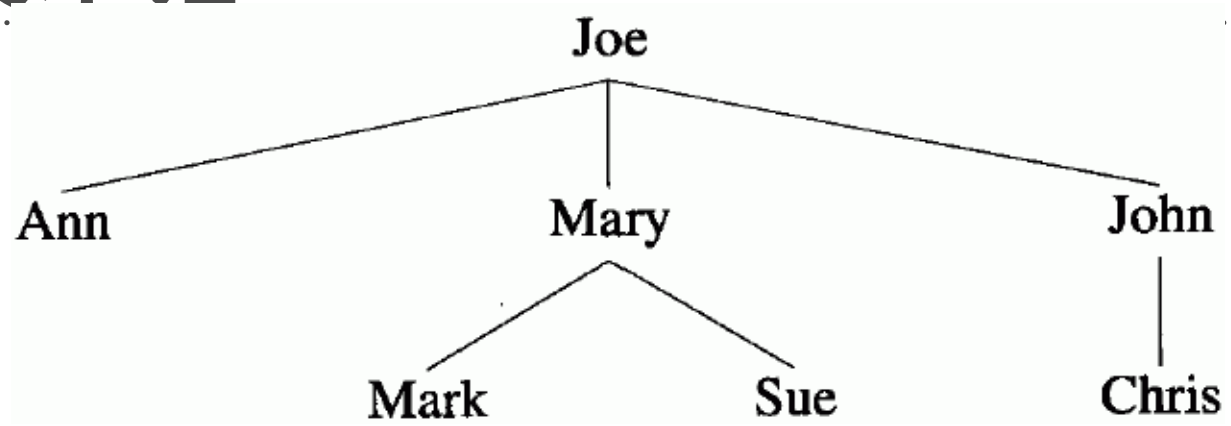


家庭关系例树的描述（续）

- 余下的元素
  - {Ann}——单一元素子树
  - {Mary，Mark，Sue}，根为Mary的子树
    - {Mark}和{Sue}，均为单元素的子树
  - {John，Chris}，根为John的子树
    - {Chris } 也为单元素子树

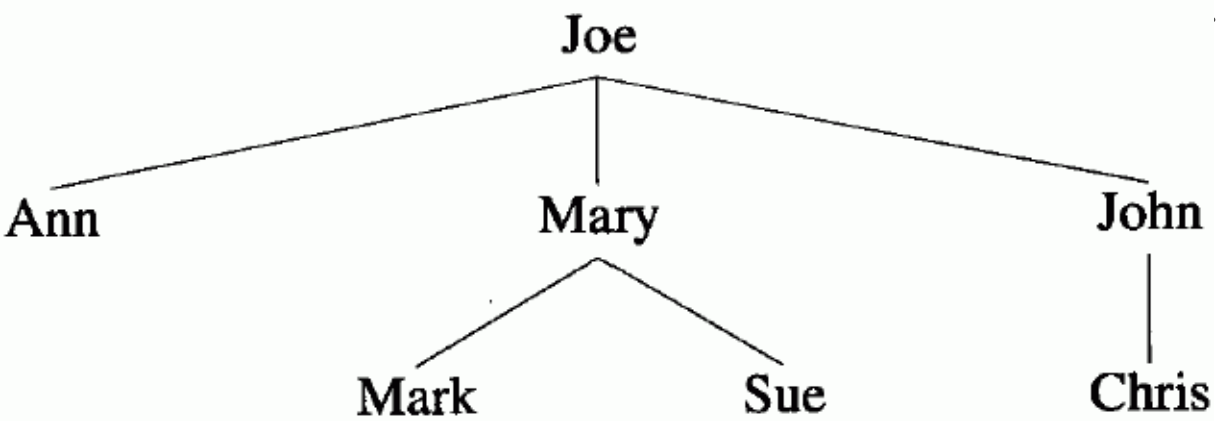


相关术语



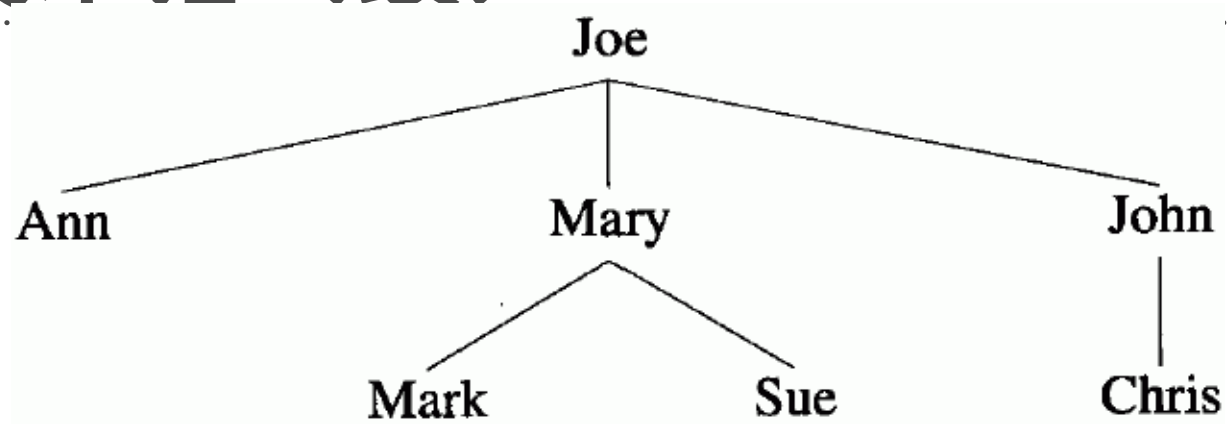
- 元素——节点
- 根节点与子树的根节点的关系——边
- 边的两端——父母（parent）和孩子（children）

相关术语（续）



- 相同父母的节点——兄弟（sibling）
- 祖父——孙子，祖先——后代
- 叶节点——没有孩子的节点，非叶节点——非终端节点

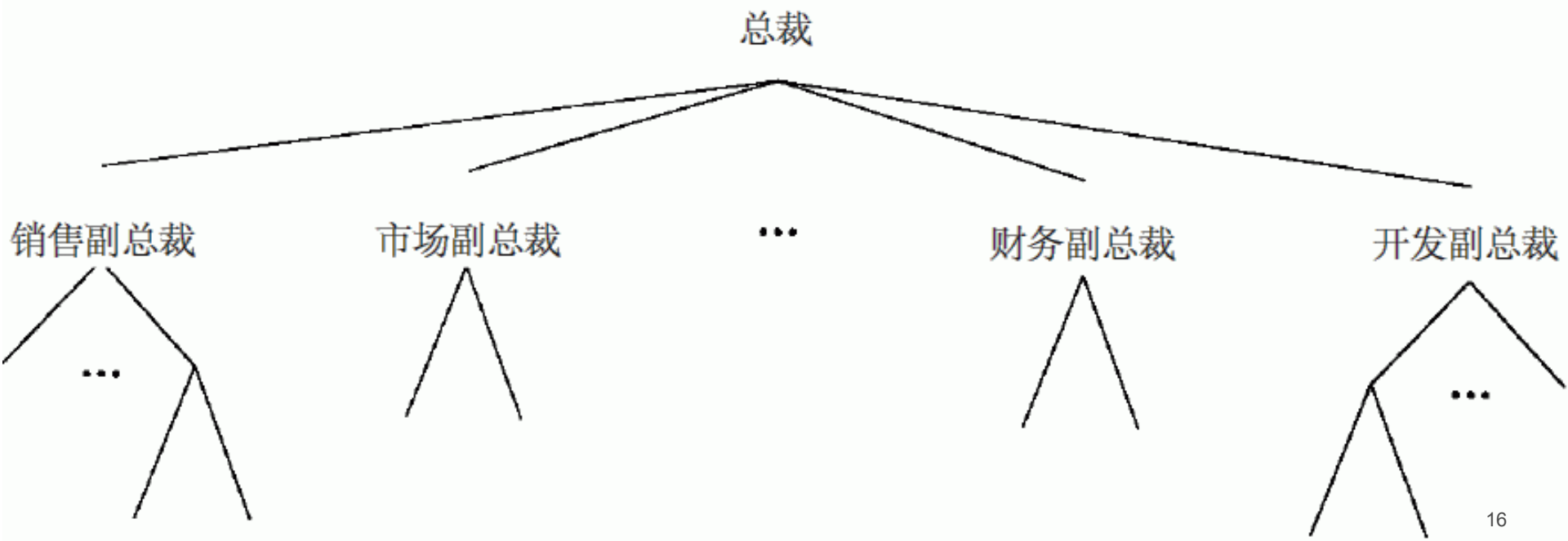
相关术语（续）



- 根——没有父节点的节点
- 层（level）：根—1，根的孩子—2，...
- 节点的度（degree）：孩子数目，叶—0

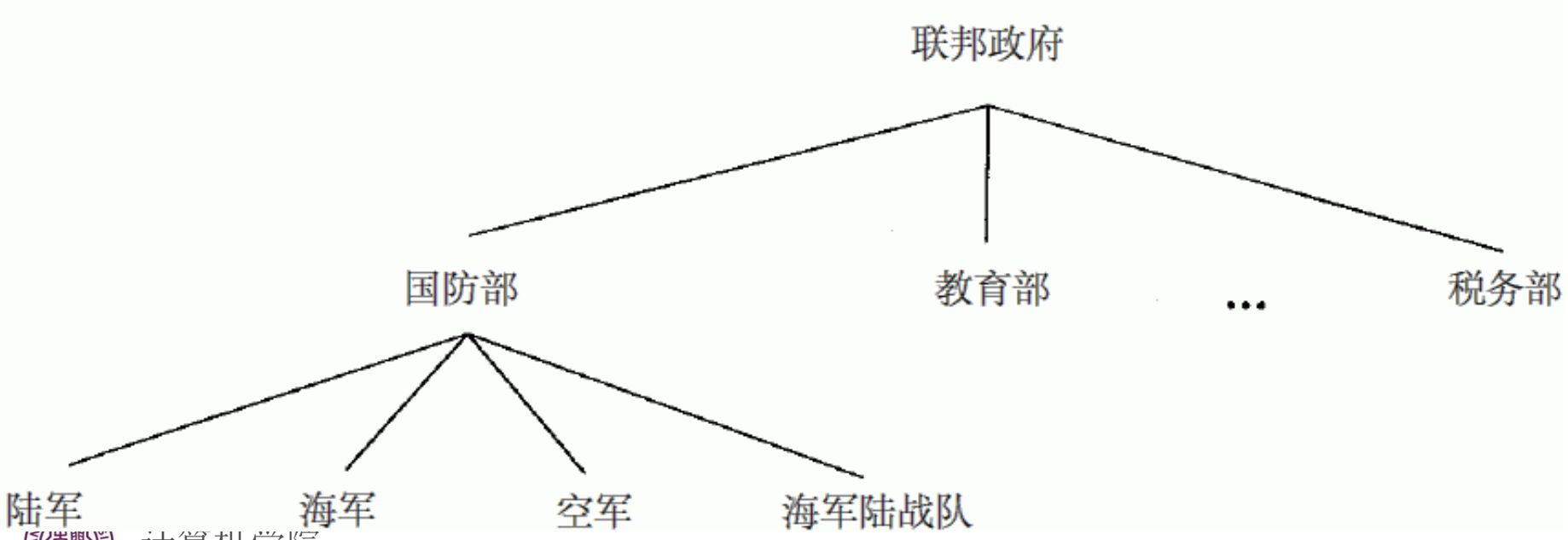
例11-6

- 公司机构例
  - 公司雇员——节点，总裁——根
  - 副总裁——总裁的子节点，总裁——副总裁的父节点
  - 不同副总裁——兄弟节点



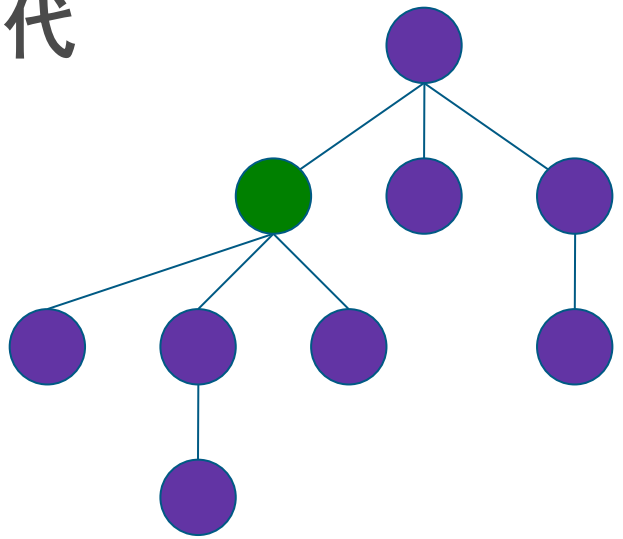
例11-6 （续）

- 政府机构例
  - 联邦政府——国防部，教育部，. . .，税务部的父节点
  - 国防部的子节点——陆军、海军、空军、海军陆战队——兄弟节点，叶节点

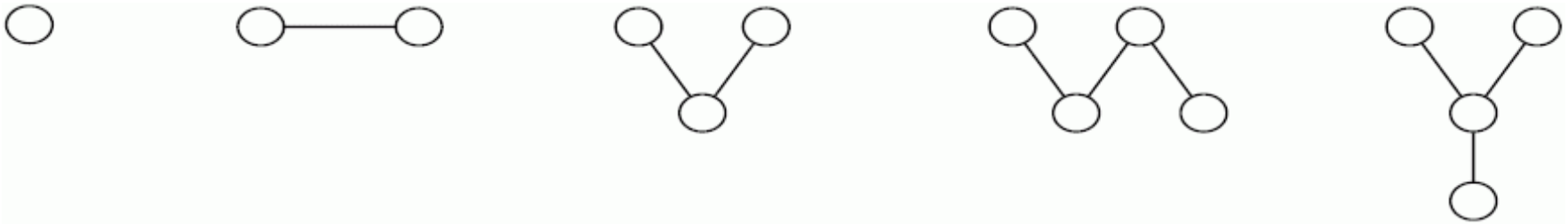


树的相关概念

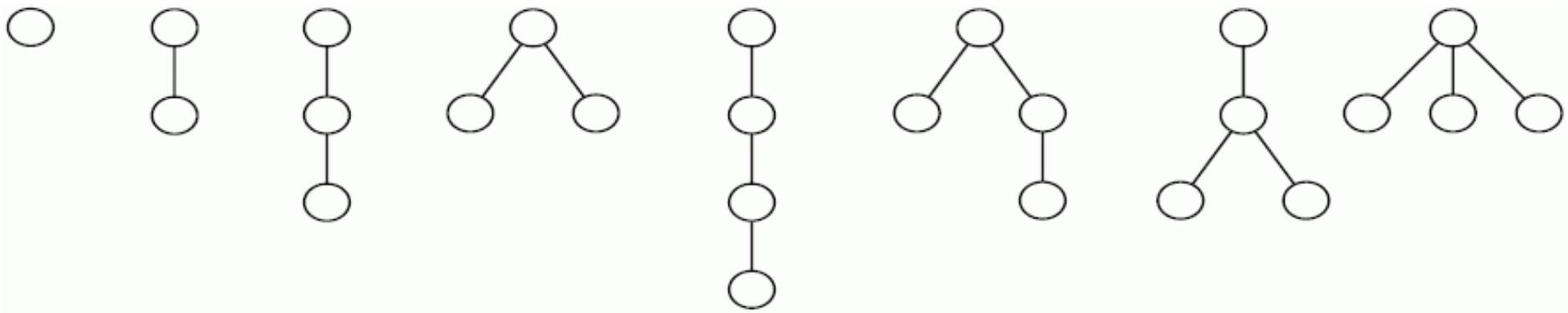
- 树、子树
- 根、分支节点、叶子节点
- 父节点、孩子节点、兄弟节点
- 祖父、孙子、祖先、后代
- 边
- 级、深度、高度
- 度



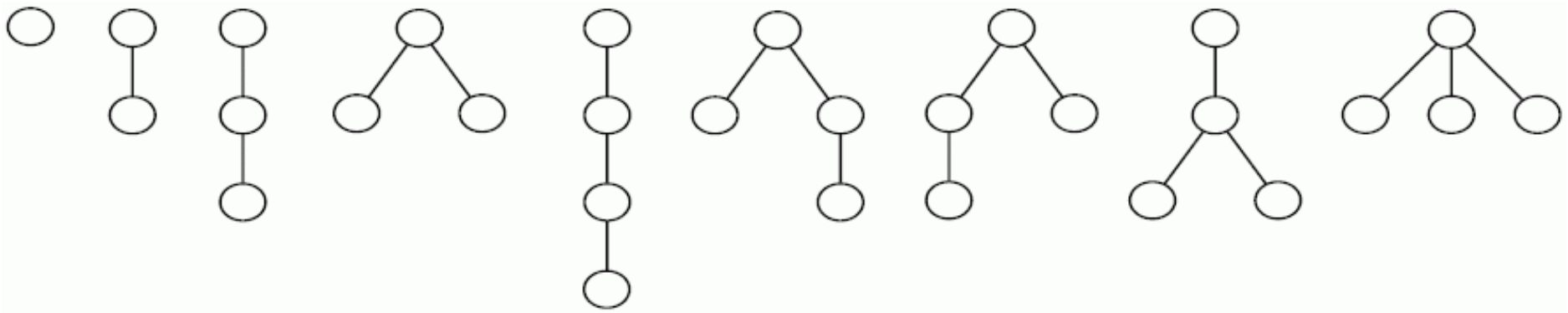
自由树



有根树



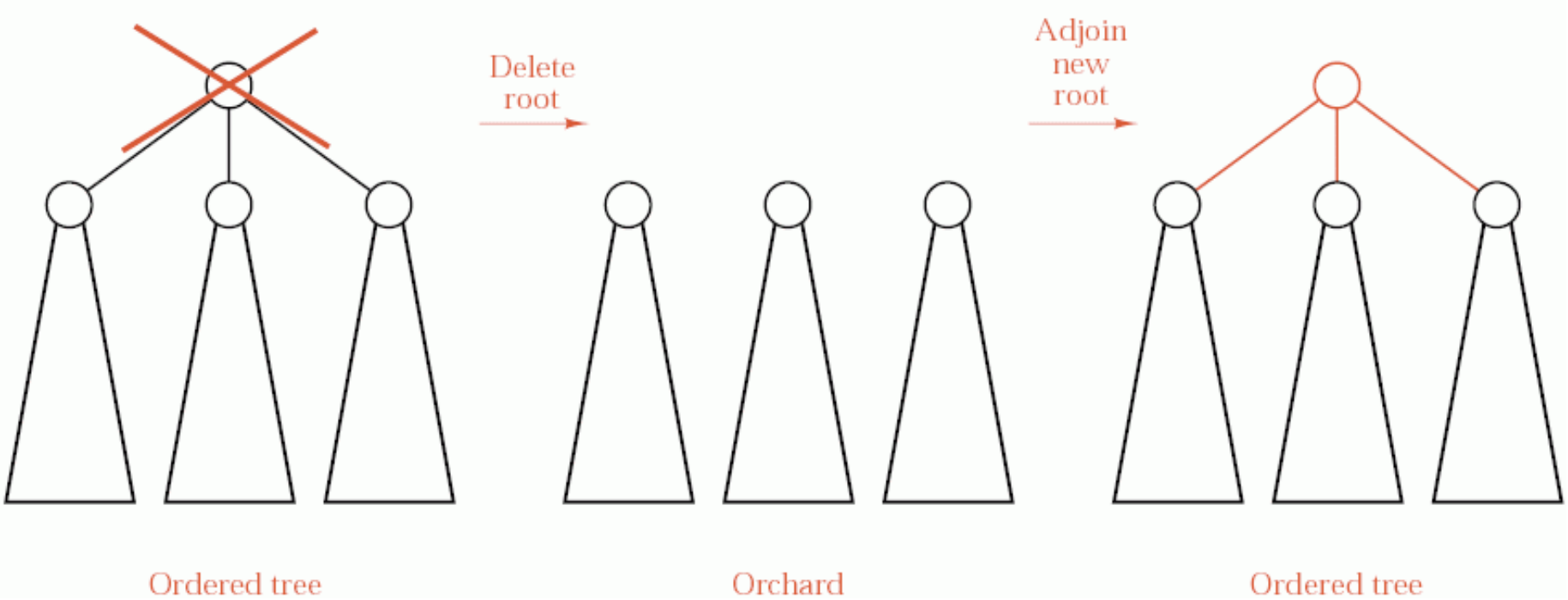
有序树



森林和有序森林

- **森林**（*forest*）：树的集合，通常认为是有根树的集合
- **有序森林**（*orchard, ordered forest*）：有序树的有序集合
- 有根（有序）树去掉根节点➔（有序）森林
- （有序）森林添加父节点➔有根（有序）树

森林和有序森林（续）


Delete
root
Adjoin
new
root
Ordered tree
Orchard
Ordered tree

有根树的递归定义

- **有根树**：
  包含单一顶点v，称为树的**根**，
  和一个森林F，F的树称为**根的子树**
  而**森林**F（可为空）是一个有根树的集合

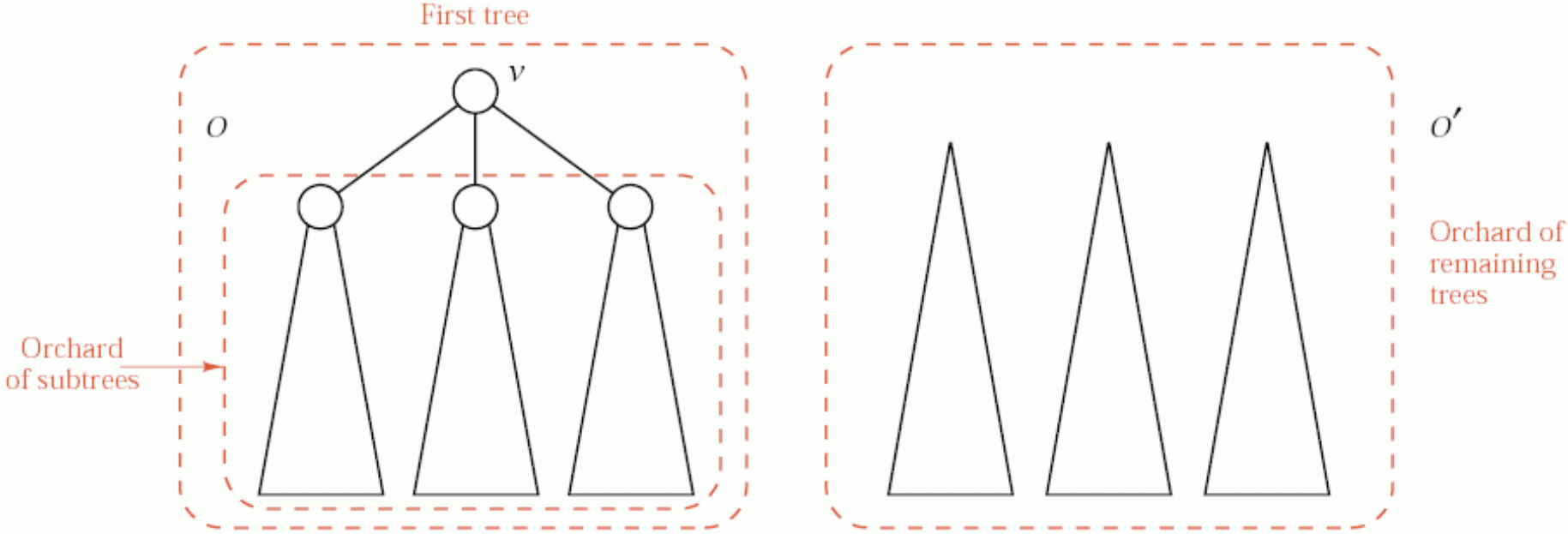
有序树的定义

- 一个**有序树**T：
  包含一个单一节点v，称为树的**根**，
  和一个**有序森林**0，0的树称为v的子树，表示为
  T={v，0}

有序森林的定义

- 一个**有序森林**0：
  或者为空集Φ，
  或包含一个有序树T，称为有序森林的**第一树**，和另一个有序森林0’（包含有序森林的其它树），可表示为0={T，0’}
- 有序树－有序森林：间接递归定义

有序森林


First tree
v
O
O′
Orchard of remaining trees
Orchard of subtrees

课堂练习

- 一棵有n个结点的树的所有结点的度数之和是多少？

课堂练习

- 如果一棵树有n_1个度为1的节点，有n_2个度为2的节点，……，n_m个度为m的节点，试问有多少个度为0的节点？

$$n_0 = \left(\sum_{i=1}^{m} (i-1) n_i \right) + 1$$

主要内容

- 树的一般定义
- 二叉树的定义和操作
- 二叉树的遍历

二叉树

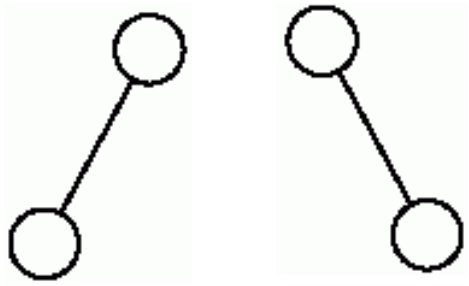
- **定义：**
  二叉树（binary tree）t是有限元素集合：
  或者为空；
  或者，有一个特殊元素根，余下的元素构成2个二叉树（可以为空）——t的左子树和右子树

二叉树和树的根本区别

- 二叉树每个节点都恰好有两棵子树（可以为空）
  树中每个节点可有若干子树
- 二叉树每个节点的子树是有序的——“左”、“右”
  树的子树间是无序的

二叉树例——递归构造

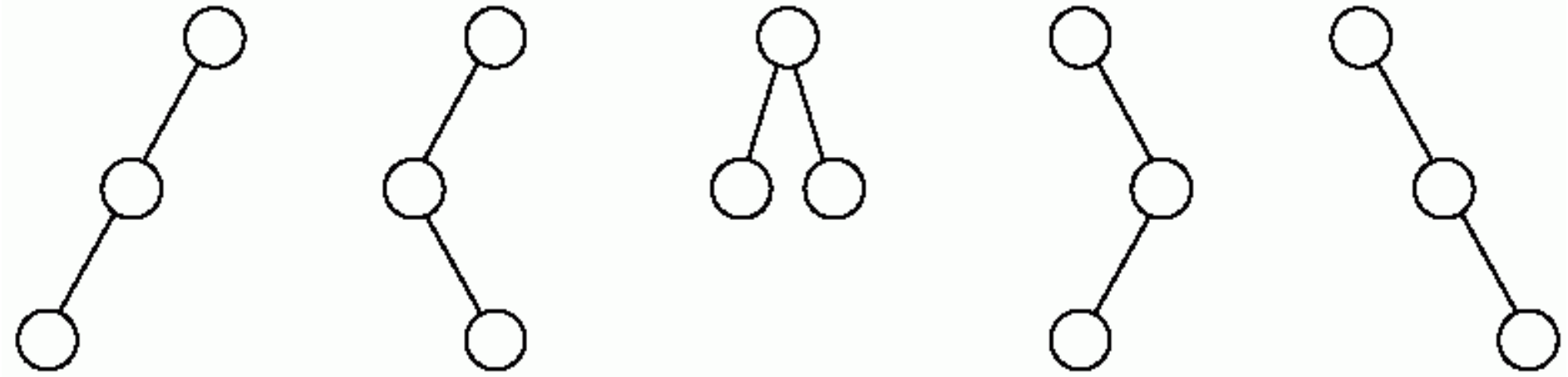
- 空二叉树——递归的停止
- 单一节点二叉树
- 两个节点：



不同！不能表示为

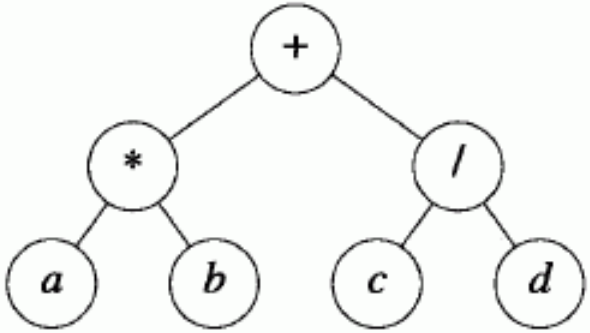
二叉树例——递归构造（续）

- 三个节点
  - 根＋左子树(2)＋右子树(0)、1＋1、0＋2

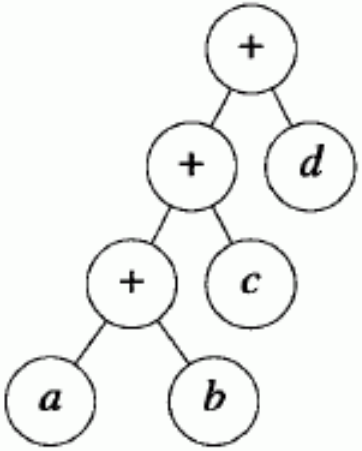


- 类似方式，可构造更大的二叉树

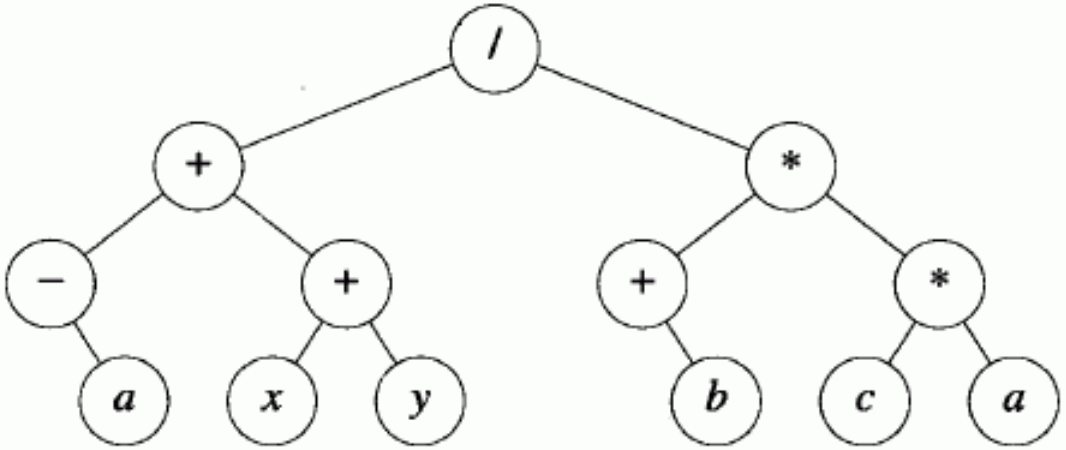
二叉树例：表达式树



a)



b)



c)

a) a*b + c/d

b) a+b+c+d

c) (−a+(x+y))/(+b*(c*a))

二叉树的特性

- **特性1** 包含n（n>0）个节点的二叉树边数为$n-1$

证明

二叉树中每个节点（除根节点，共n-1个节点） 有且只有一个父节点

每个子节点与父节点间有且只有一条边

➔边数为$n-1$

特性2

- 二叉树的**高度**（height）（**深度**，depth）：二叉树的层数
- **特性2**　若二叉树的高度为h，$h \geqslant 0$，则它最少有h个节点，最多有2^h-1个节点

特性2的证明

证明

每层最少要有1个节点➔节点数最少为h

每个节点最多有2个子节点➔则第i层节点最多为2^{i-1}个，$i \geqslant 1$

➔节点的总数不会超过

$$\sum_{i=1}^{h} 2^{i-1} = 2^0 + 2^1 + \cdots + 2^{h-1} = 2^h - 1$$

特性2说明

深度	该层最多节点数	二叉树的最多节点总数
1	$2^0=1$	1
2	$2^1=2$	3
3	$2^2=4$	7
4	$2^3=8$	15
5	$2^4=16$	31
6	$2^5=32$	63
……	……	……

特性3

- **特性3**　包含n个节点的二叉树的高度最大为n，最小为 $\lceil \log_2(n+1) \rceil$

证明

每层至少一个元素➔高度不会超过n

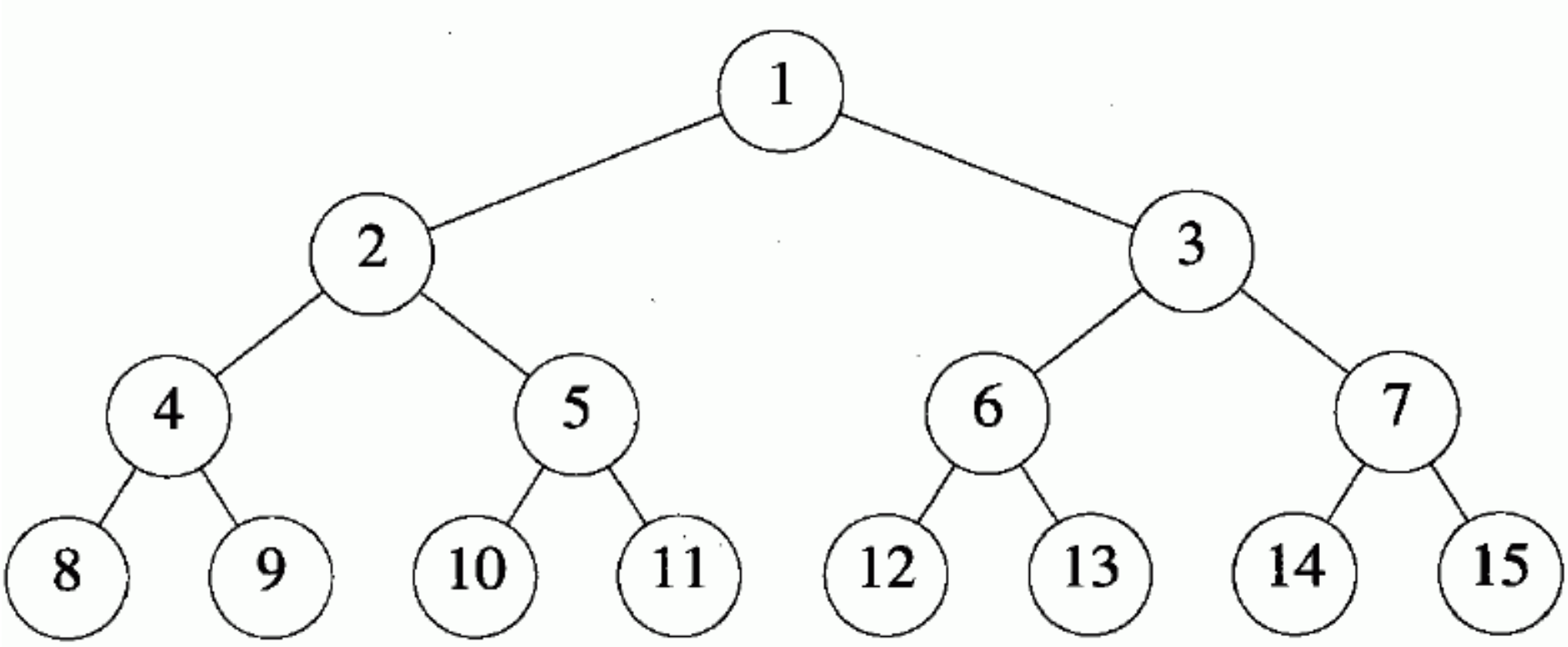
由特性2，可知高度为h，节点最多2^h-1

即$n \leqslant 2^h-1$➔$h \geqslant \log_2(n+1)$

且h是整数➔ $h \geq \lceil \log_2(n+1) \rceil$

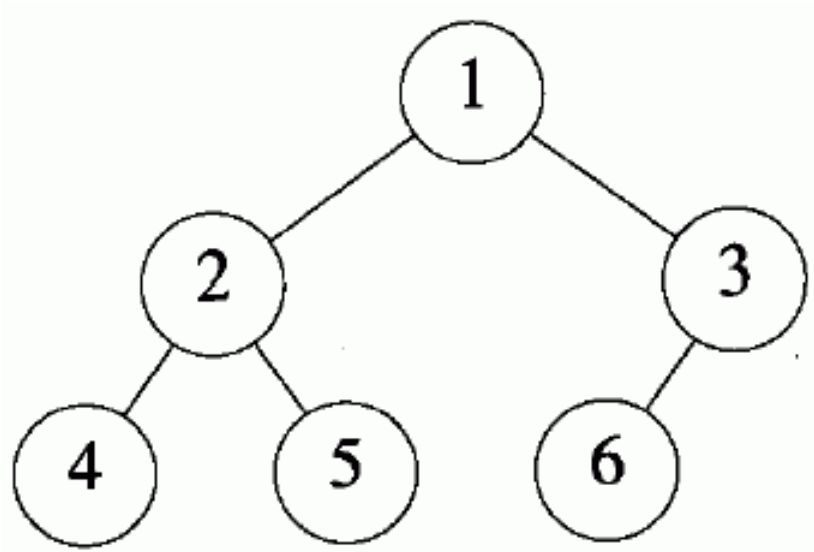
满二叉树（full binary tree ）

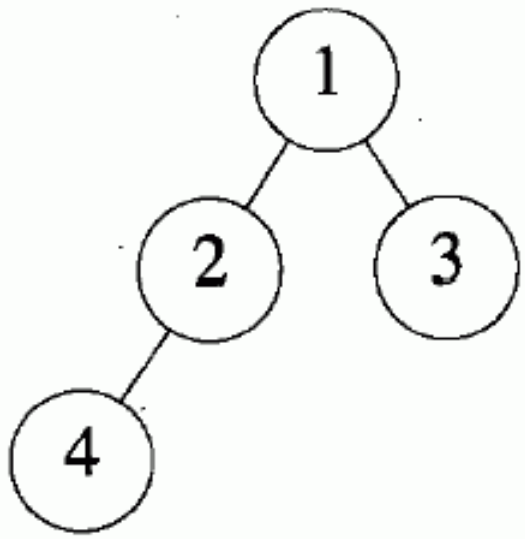
- 高度为h，节点数2^h-1

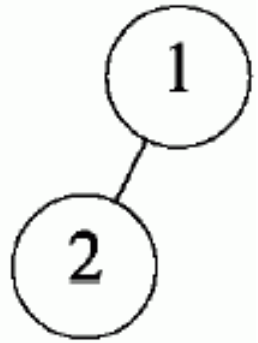


完全二叉树

- 高度为h 的满二叉树中节点按从上到下，从左到右的顺序从1到2^h-1进行编号
- 从中删除k个节点，编号为2^h-i，$1 \leqslant i \leqslant k$
- 即为完全二叉树，深度为$\lceil \log_2(n+1) \rceil$







另一种定义方法

- 设二叉树T有n个节点，令

$$k = \lceil \log_2(n+1) \rceil \qquad r = n - (2^{k-1} - 1)$$

则k代表最下一层，也就是二叉树的深度，r代表第k层的节点数，其中2^{k-1}个节点放满第1到第k-1层，则：

若$0<r<=2^{k-1}$，且这r个节点集中存放在第k层的左侧，则T是一棵**完全二叉树**

特别地，若$r=2^{k-1}$，则T是一棵**满二叉树**

结论：满二叉树是完全二叉树

特性4

- **特性4** 设完全二叉树中一节点的序号为 i，$1 \leqslant i \leqslant n$。则有以下关系成立：

1）当 $i = 1$时，该元素为二叉树的根。若 $i > 1$，则该元素父节点的编号为 $\lfloor i/2 \rfloor$

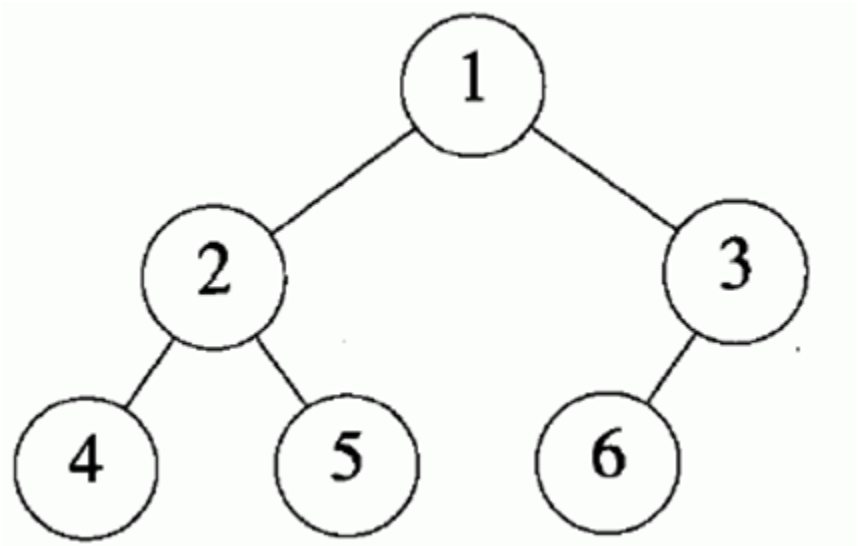
2）当$2i > n$时，该元素无左孩子。否则，其左孩子的编号为$2i$

3）若$2i + 1 > n$，该元素无右孩子。否则，其右孩子编号为$2i + 1$

特性4（续）

证明

通过对 i 进行归纳即可得证



特性5

- 设二叉树中度为2的节点有n_2个，度为1的节点有n_1个，度为0的节点有n_0个，则$n_0=n_2+1$

$$n_0 = \left(\sum_{i=1}^{m} (i-1) n_i \right) + 1$$

- 一棵二叉树有1024个节点，其中465个是叶节点，那么树中度为2和度为1的节点各有多少？

思考

- 一棵二叉树有19个节点，其中6个是叶节点，那么树中度为2和度为1的节点各有多少？
  - 能否构造一棵符合条件的二叉树？
  - 能否构造一棵符合条件的完全二叉树？

思考

- 有m个叶子的**二叉树**最多有多少个节点？

- 有m个叶子的**完全二叉树**最多有多少个节点？

课堂练习

- 设高度为h的二叉树中只有度为0和度为2的节点，则此类二叉树所包含的节点数至少为_____，至多为_____。

课堂练习

- 设完全二叉树的第6层有24个叶节点，则此树最多有_____个节点

A. 55

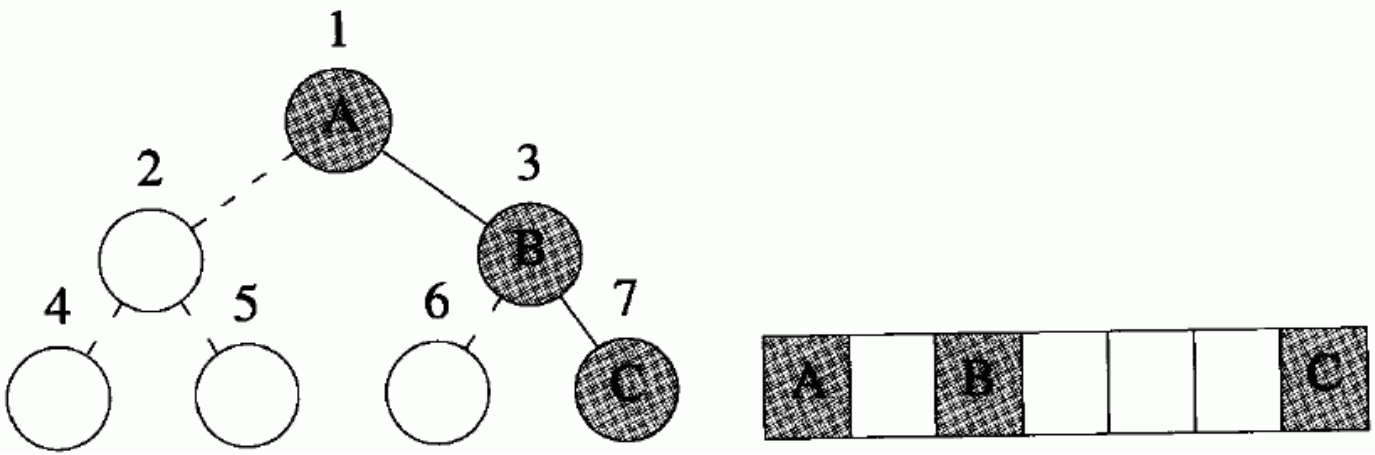
B. 79

C. 81

D. 127

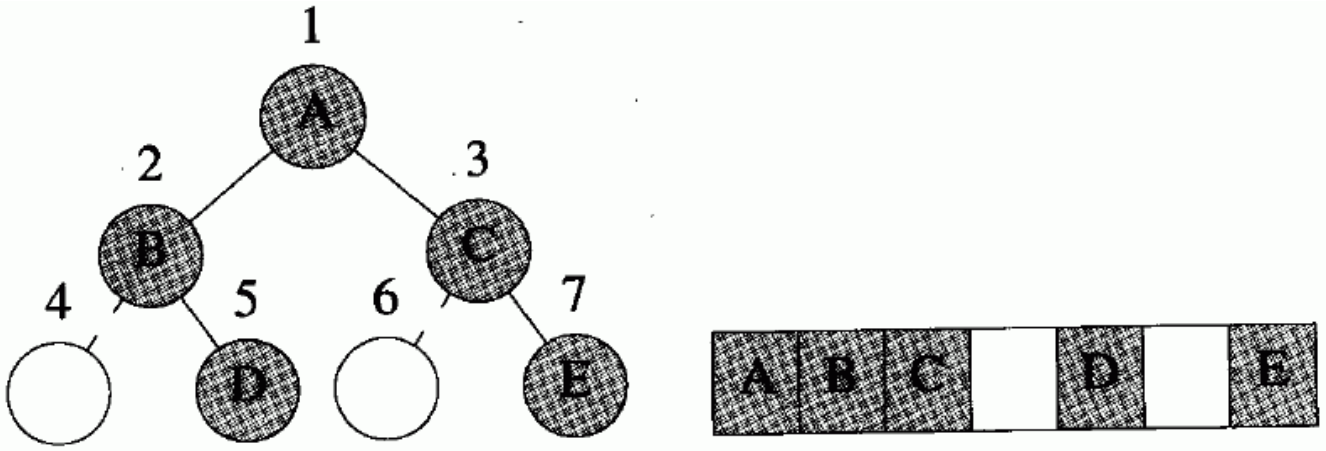
二叉树描述

- 公式化描述——利用特性4
  - 普通二叉树——缺少部分节点的完全二叉树



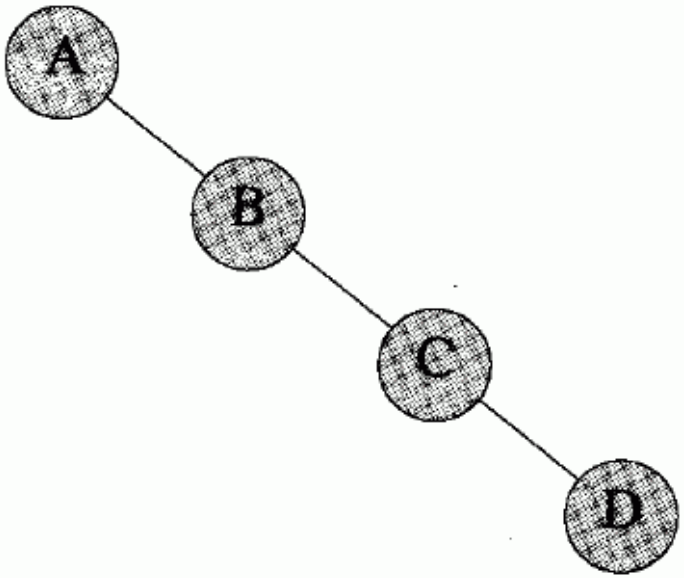
公式化描述

- 数组位置——节点编号
- 父子节点关系——特性4

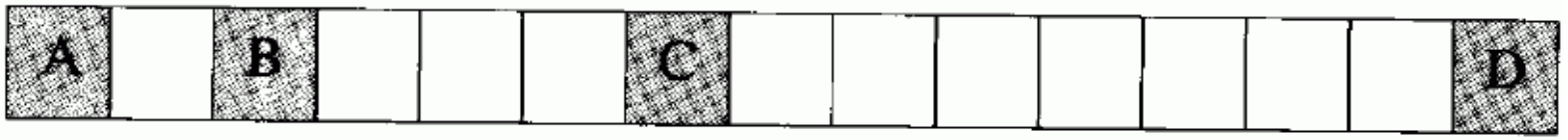


公式化描述

- n层二叉树——2^n-1数组保存，可能空间浪费！

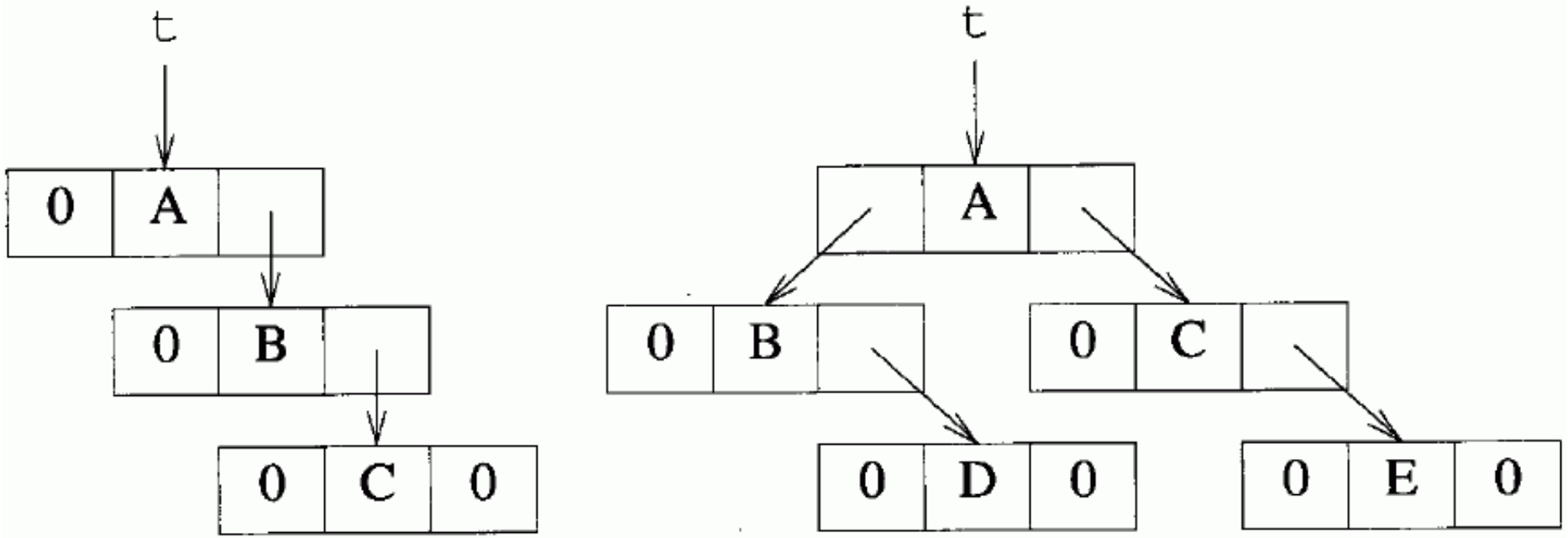


a)



链表描述

- 树节点——节点类对象
  - 数据域、LeftChild、RightChild
  - n-1条边➔2n-(n-1)=n+1个空指针



BinaryTreeNode类

```
template <class T>
class BinaryTreeNode {
  public:
    BinaryTreeNode() {LeftChild = RightChild = 0;}
    BinaryTreeNode(const T& e) {data = e;  LeftChild =
  RightChild = 0;}
    BinaryTreeNode(const T& e, BinaryTreeNode *l,
  BinaryTreeNode *r){data = e;  LeftChild = l;  RightChild = r;}

  private:
    T data;
    BinaryTreeNode<T> *LeftChild,   // left subtree
                      *RightChild;  // right subtree
};
```

抽象数据类型BinaryTree

抽象数据类型*BinaryTree*{

实例

元素集合；如果不空，则被划分为根节点、左子树和右子树；每个子树仍是一个二叉树

操作

Create()：创建一个空的二叉树；

IsEmpty：如果二叉树为空，则返回true，否则返回false

Root(*x*)：取x为根节点；如果操作失败，则返回false，否则返回true

抽象数据类型（续）

MakeTree(*root*, *left*, *right*)：创建一个二叉树，*root*作为根节点，*left*作为左子树，*right*作　为右子树

BreakTree(*root*, *left*, *right*)：拆分二叉树

PreOrder：先序遍历

InOrder：中序遍历

PostOrder：后序遍历

LevelOrder：逐层遍历

}

类BinaryTree

```
template<class T>
class BinaryTree {
public:
    BinaryTree() {root = 0;};
    ~BinaryTree(){};
    bool IsEmpty() const
      {return ((root) ? false : true);}
    bool Root(T& x) const;
    void MakeTree(const T& element,
        BinaryTree<T>& left, BinaryTree<T>& right);
    void BreakTree(T& element, BinaryTree<T>& left,
                BinaryTree<T>& right);
    void PreOrder(void(*Visit)(BinaryTreeNode<T> *u))
        {PreOrder(Visit, root);}
```

类BinaryTree（续）

```
    void InOrder(void(*Visit)(BinaryTreeNode<T> *u))
        {InOrder(Visit, root);}
    void PostOrder(void(*Visit)(BinaryTreeNode<T> *u))
        {PostOrder(Visit, root);}
    void LevelOrder(void(*Visit)(BinaryTreeNode<T> *u));
private:
    BinaryTreeNode<T> *root;  // pointer to root
    void PreOrder(void(*Visit)
     (BinaryTreeNode<T> *u), BinaryTreeNode<T> *t);
    void InOrder(void(*Visit)
     (BinaryTreeNode<T> *u), BinaryTreeNode<T> *t);
    void PostOrder(void(*Visit)
     (BinaryTreeNode<T> *u), BinaryTreeNode<T> *t);
};
```

获取根节点数据

```
template<class T>
bool BinaryTree<T>::Root(T& x) const
{// Return root data in x.
 // Return false if no root.
   if (root) {x = root->data;
              return true;}
   else return false;  // no root
}
```

创建树

```
template<class T>
void BinaryTree<T>::MakeTree(const T& element,
        BinaryTree<T>& left, BinaryTree<T>& right)
{// Combine left, right, and element to make new tree.
 // left, right, and this must be different trees.
   // create combined tree
   root = new BinaryTreeNode<T>
            (element, left.root, right.root);
   // deny access from trees left and right
   left.root = right.root = 0;
}
```

缺陷：允许MakeTree(e, X, X)且X不为空

分裂树

```
template<class T>
void BinaryTree<T>::BreakTree(T& element,
      BinaryTree<T>& left, BinaryTree<T>& right)
{// left, right, and this must be different trees.
   // check if empty
   if (!root) throw BadInput(); // tree empty

   // break the tree
   element = root->data;
   left.root = root->LeftChild;
   right.root = root->RightChild;

   delete root;
   root = 0;
}
```

思考

- 在上述存储方式下，寻找某一节点孩子的复杂度是O(1)，寻找其父亲的复杂度是O(n)，为什么？

- 如果希望将寻找父节点的效率提高到O(1)，如何做？

主要内容

- 树的一般定义
- 二叉树的定义和操作
- **二叉树的遍历**

二叉树遍历

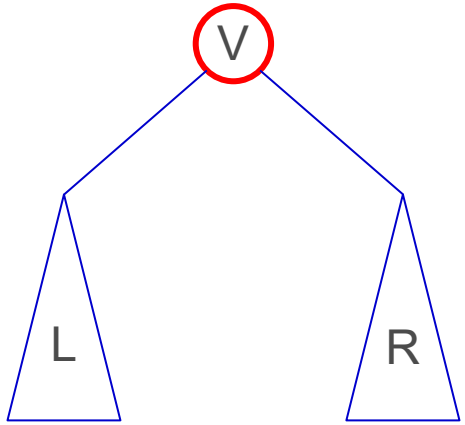
- 按照某种顺序访问树中的每个节点，
- 要求每个节点被访问一次且仅被访问一次
- 这就是**二叉树遍历问题**

二叉树遍历

- **遍历顺序【关键】**
  - 访问根节点、左子树、右子树的顺序
  - 左右子树的访问（遍历）？——递归！
- 可能的遍历顺序
  - V—根，L—左子树，R—右子树
  - VLR　LVR　LRV　VRL　RVL　RLV

标准遍历顺序

- 都是左子树先于右子树，关键——根的访问次序
- 先序遍历（preorder）——VLR
- 中序遍历（inorder）——LVR
- 后序遍历（postorder）——LRV



先序遍历

```
template <class T>
void PreOrder(BinaryTreeNode<T> *t)
{// Preorder traversal of *t.
  if (t) {
    Visit(t);                  // visit tree root
    PreOrder(t->LeftChild);   // do left subtree
    PreOrder(t->RightChild);  // do right
  subtree
    }
}
```

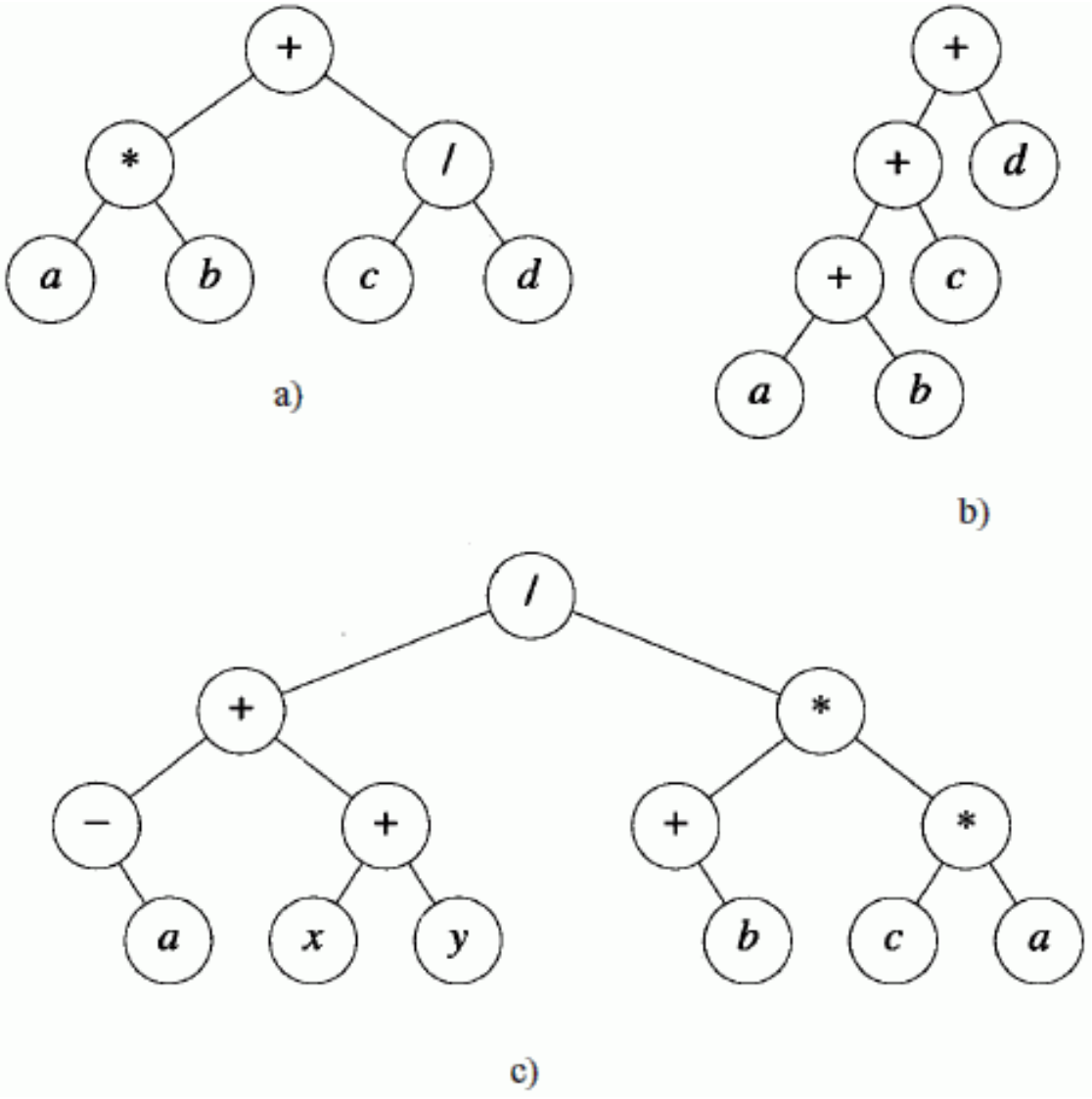
中序遍历

```
template <class T>
void InOrder(BinaryTreeNode<T> *t)
{// Inorder traversal of *t.
  if (t) {
    InOrder(t->LeftChild);   // do left subtree
    Visit(t);                // visit tree root
    InOrder(t->RightChild);  // do right
  subtree
    }
}
```

后序遍历

```
template <class T>
void PostOrder(BinaryTreeNode<T> *t)
{// Postorder traversal of *t.
  if (t) {
    PostOrder(t->LeftChild);   // do left
  subtree
    PostOrder(t->RightChild);  // do right
  subtree
    Visit(t);                  // visit tree root
    }
}
```

先序遍历表达式树



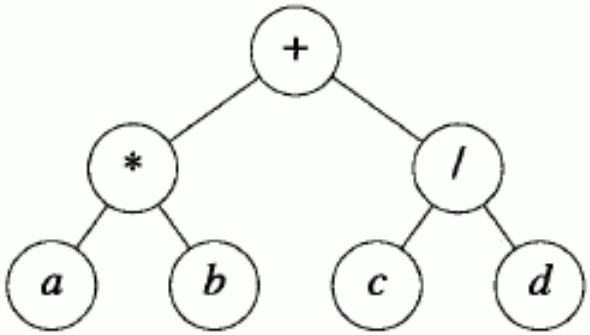
a) +*ab/cd

b) +++abcd

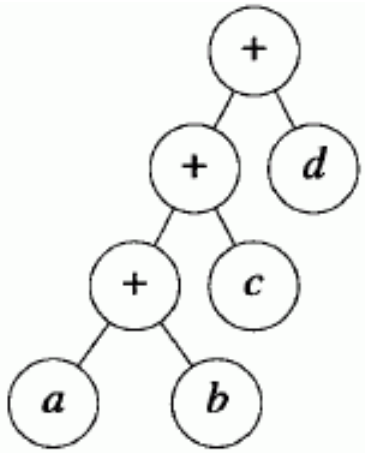
c) /+-a+xy*+b*ca

- 前缀表达式

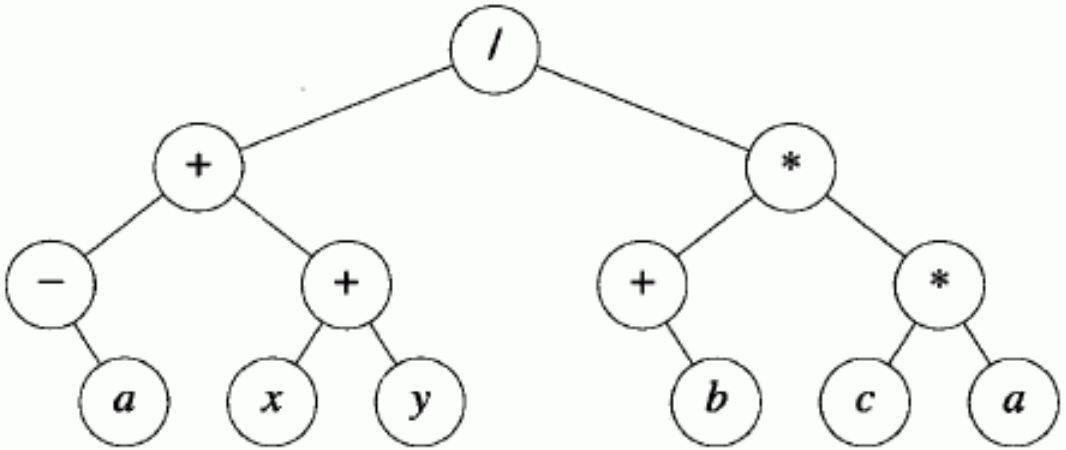
中序遍历表达式树



a)



b)



c)

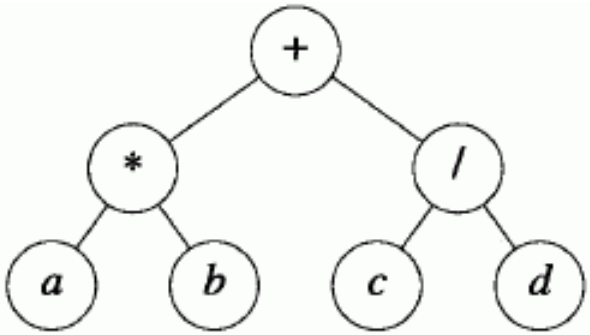
a)a*b + c/d

b)a+b+c+d

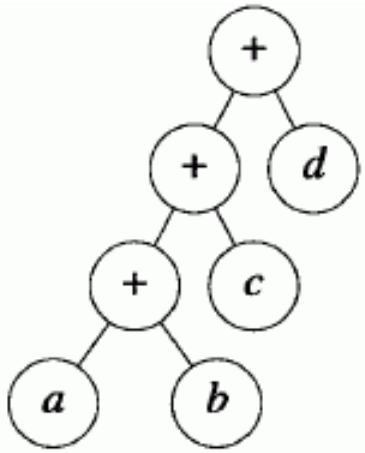
c)-a+x+y/+b*c*a

- 中缀表达式——人类习惯
- 需加括号

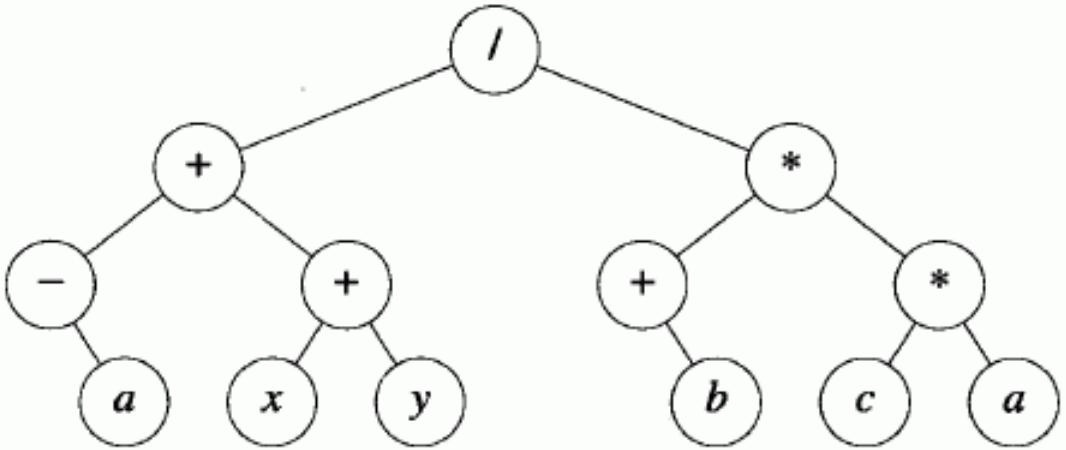
后序遍历表达式树



a)



b)



c)

a)ab*cd/+

b)ab+c+d+

c)a-xy++b+ca**/

- 后缀表达式——计算机计算非常方便

输出完全括号化的中缀表达式

```
template <class T>
void Infix(BinaryTreeNode<T> *t)
{// Output infix form of expression.
  if (t) {cout << '(';
          Infix(t->LeftChild);  // left operand
          cout << t->data;      // operator
          Infix(t->RightChild); // right operand
          cout << ')';}
}
```

逐层遍历（宽度优先）

- 根→叶逐层，同层由左至右

```
template <class T>
void LevelOrder(BinaryTreeNode<T> *t)
{// Level-order traversal of *t.
   LinkedQueue<BinaryTreeNode<T>*> Q;
   while (t) {
      Visit(t);  // visit t
```

逐层遍历（宽度优先）

```
        // put t's children on queue
        if (t->LeftChild) Q.Add(t->LeftChild);
        if (t->RightChild) Q.Add(t->RightChild);

        // get next node to visit
        try {Q.Delete(t);}
        catch (OutOfBounds) {return;}
        }
}
```

出队次序即是遍历次序；

同时控制t移向下一节点。

由遍历顺序推导二叉树结构

- 一棵二叉树
  先序遍历结果1，2，4，7，3，5，6，8，9
  中序遍历结果4，7，2，1，5，3，8，6，9
  能推导出其结构吗？
- 可以！

第一步

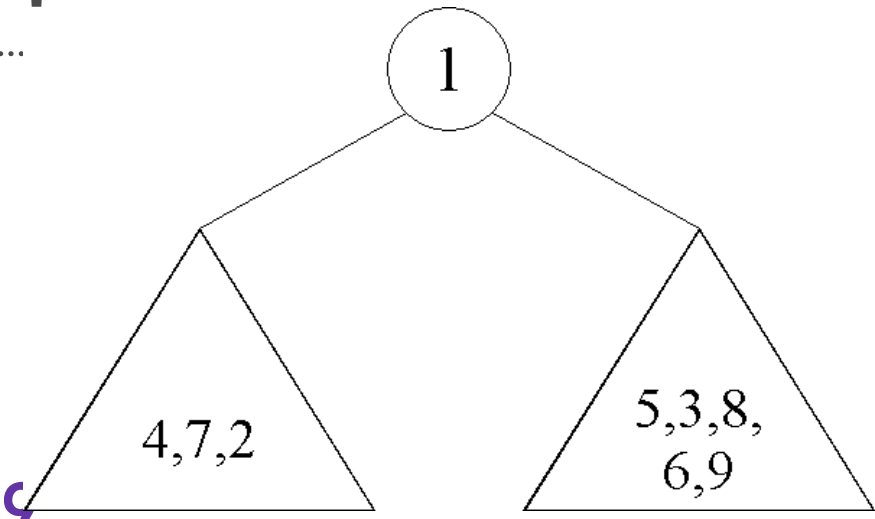
1，2，4，7，3，5，6，8，9

4，7，2，1，5，3，8，6，9

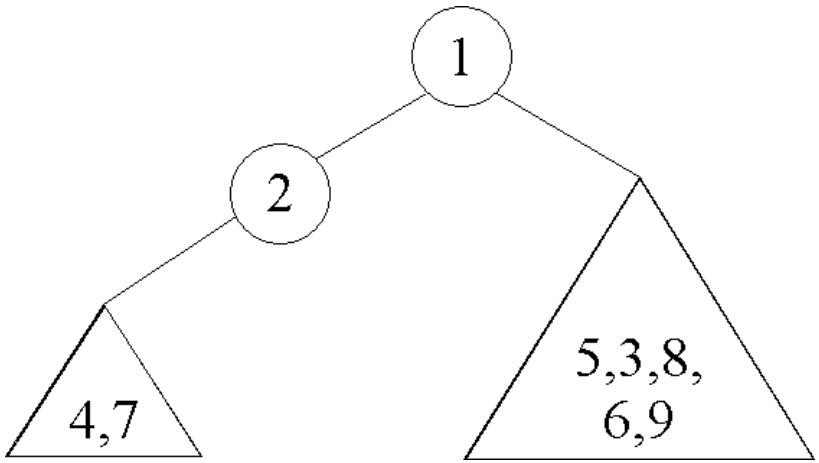
- 先序遍历
  根－左子树－右子树➔“1”必为根节点
- 中序遍历
  左子树－根－右子树，且1为根节点
  ➔4，7，2为左子树，5，3，8，6，9为右子树

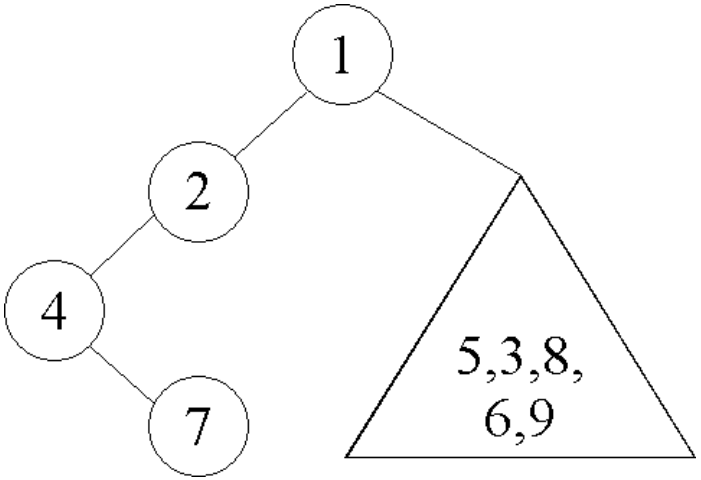
下面怎么办？递归！

- 左子树
  先序遍历2，4，7
  中序遍历4，7，2
- 右子树
  先序遍历3，5，6，8，9
  中序为5，3，8，6，9
- 利用相同方法构造左、右子树，直至列表长度为0——空子树，无需构造



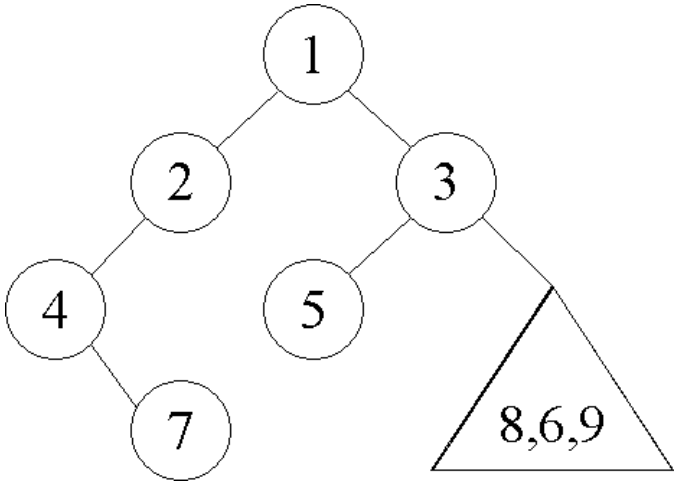
遍历顺序→二叉树结构（续）





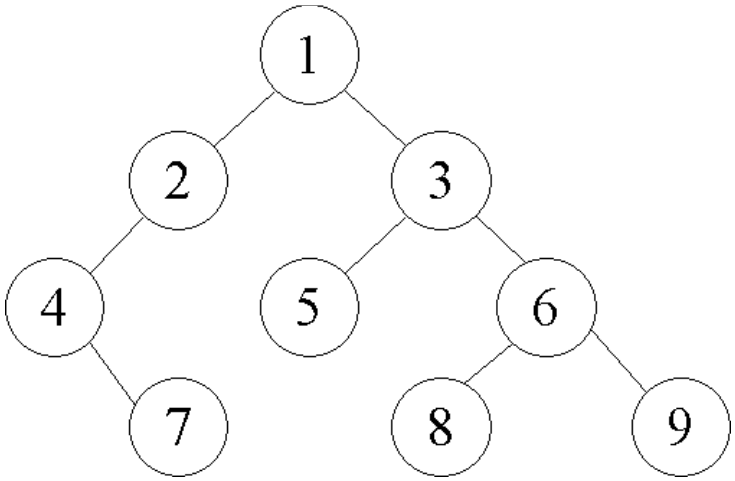
遍历顺序→二叉树结构（续）

先序3, 5, 6, 8, 9，中序5, 3, 8, 6, 9



遍历顺序→二叉树结构（续）

先序6, 8, 9
中序8, 6, 9



抽象数据类型及类的扩充

- 增加如下二叉树操作：

PreOutput()：按前序方式输出数据域

InOutput()：按中序方式输出数据域

PostOutput()：按后序方式输出数据域

LevelOutput()：逐层输出数据域

Delete()：删除一棵二叉树，释放其节点

Height()：返回树的高度

Size()：返回树中节点数

销毁二叉树

- 删除所有节点
- **后序遍历**：删除左子树、删除右子树、删除根
- 辅助函数——删除单个节点

```
static void Free(BinaryTreeNode<T> *t)
  {delete t;}
```

- 删除二叉树

```
void Delete() {PostOrder(Free, root); root = 0;}
```

计算高度

- **后序遍历**
  - 左子树高度、右子树高度较大者+1（根节点）→树的高度
  - 递归公式：h=max{hl, hr} + 1→递归函数
- 公共接口

```
int Height() const {return Height(root);}
```

计算高度的递归函数Height

```
template <class T>
int BinaryTree<T>::Height(BinaryTreeNode<T> *t)
  const
{// Return height of tree *t.
  if (!t) return 0;              // empty tree
  int hl = Height(t->LeftChild);  // height of left
  int hr = Height(t->RightChild); // height of right
  if (hl > hr) return ++hl;
  else return ++hr;
}
```

统计节点数目

- **任意一种遍历方法**，每个节点将统计计数加1
- 私有辅助函数Add1

```
static void Add1(BinaryTreeNode<T> *t)
  {_count++;}
```

- 节点数统计函数

```
int Size()
      {_count = 0; PreOrder(Add1, root); return _count;}
```

统计节点数目——方法二

- 利用递归公式：s=sl+sr+1

```
template <class T>
int BinaryTree<T>::Size(BinaryTreeNode<T> *t)
  const
{
   if (!t) return 0;
   else return Size(t->LeftChild)+Size(t-
   >RightChild)+1;
}
```

二叉树遍历小结

- 前序、中序、后序遍历
  - 深度优先
  - 表明访问根节点的次序
- 按层遍历
  - 宽度优先
- 遍历思想简述
  - 软件工程、自主学习、房地产建设……

课堂练习

- 已知二叉树的前序序列是abdcef，中序序列是dbaecf，其后序序列是什么？

思考

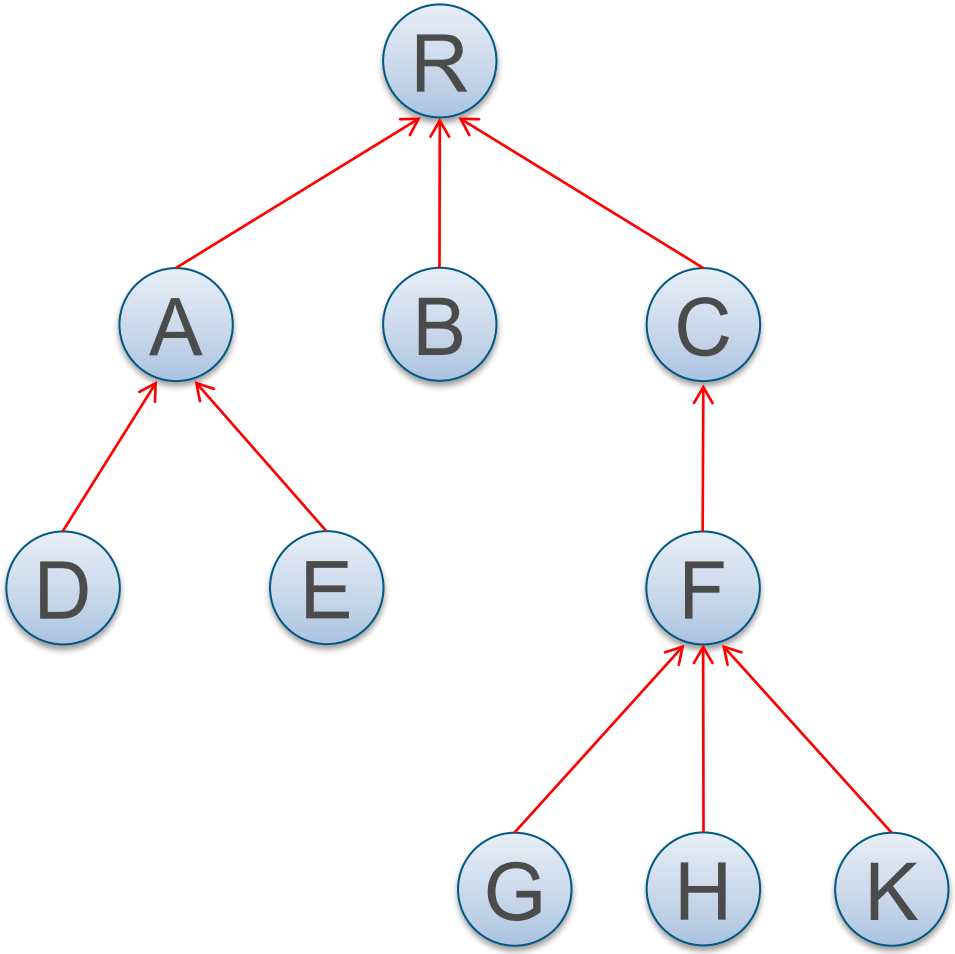
- 前序序列与中序序列相同的是什么二叉树？

- 一棵二叉树的前序序列的最后一个节点是否是它层次序列的最后一个节点？

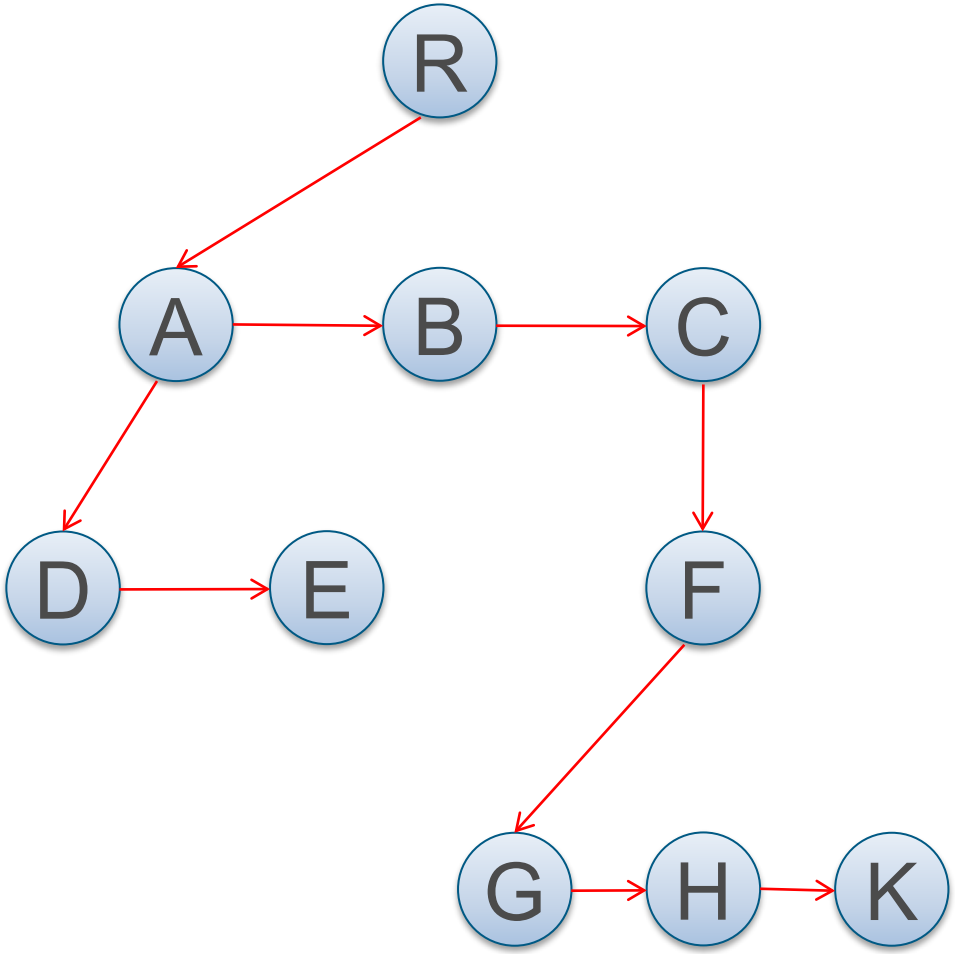
补充：二叉树与森林互转

- 已知：二叉树使用链表表示法
- 一般树如何表示呢？
  - 孩子指针表示法
  - 父指针表示法
  - 左孩子右兄弟表示法

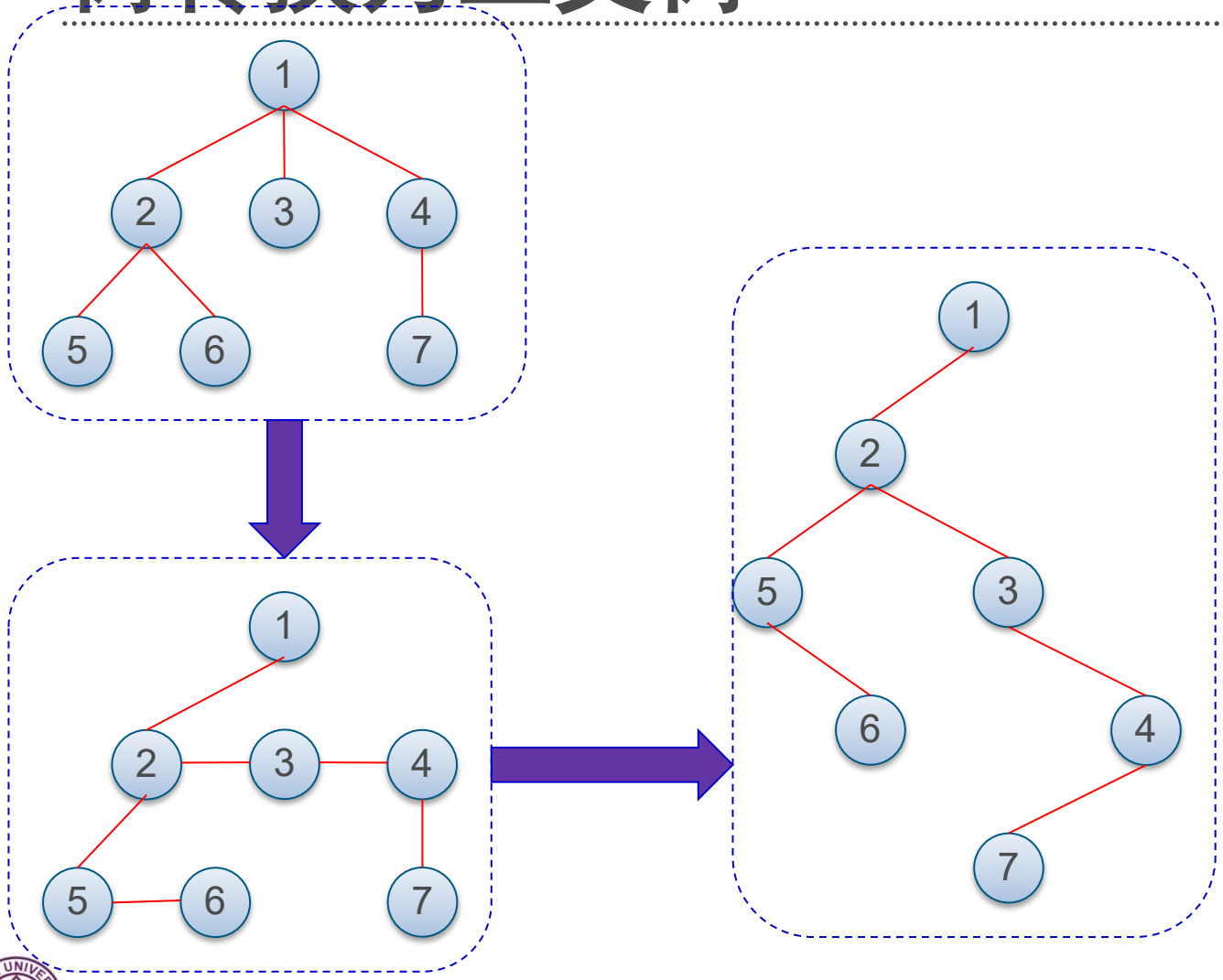
树的父指针表示法



树的左孩子右兄弟表示法



树转换为二叉树



规则：

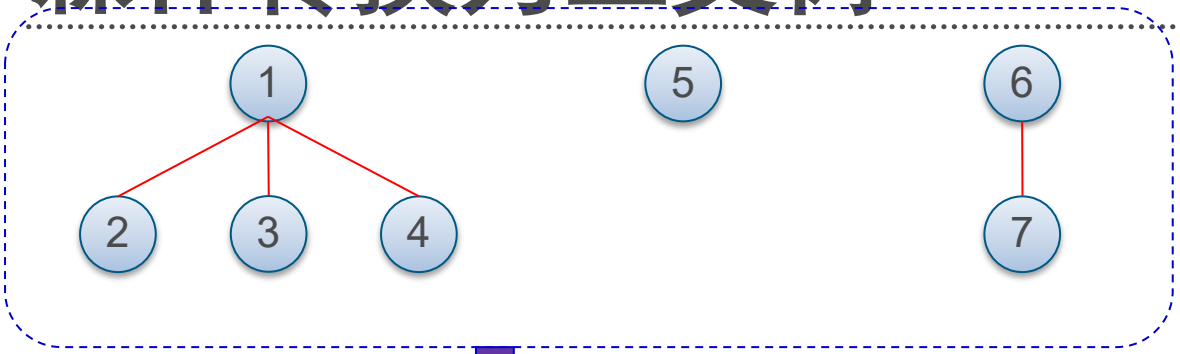
1.树中某节点的第一个孩子是二叉树中该节点的左孩子；

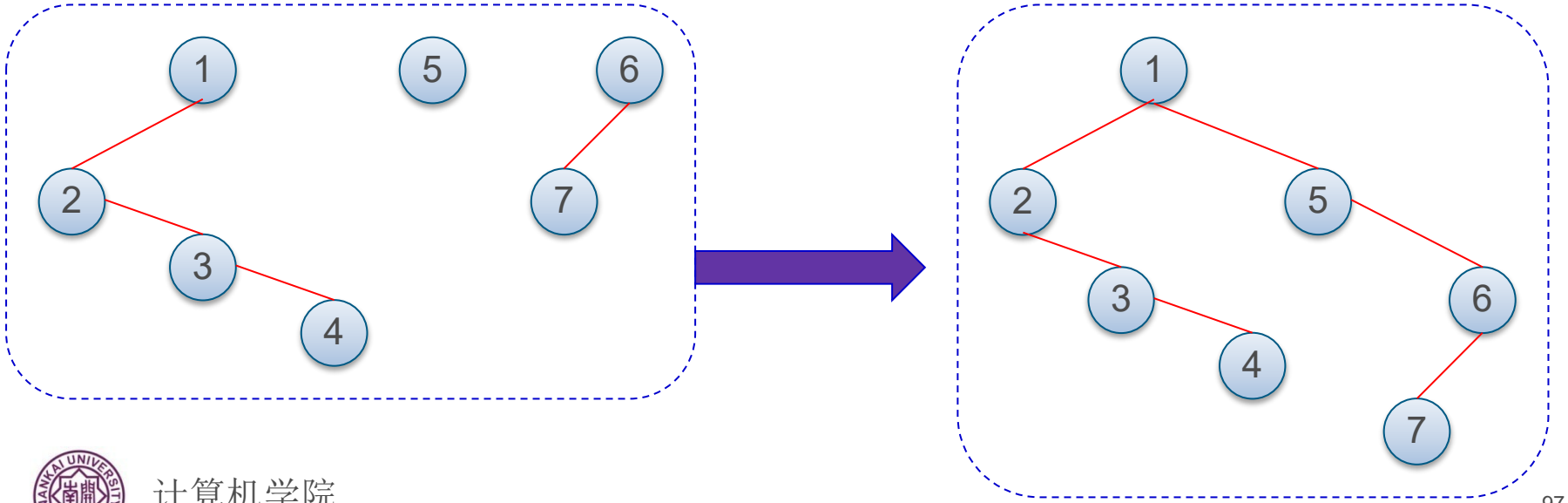
2.树中某节点的右兄弟是二叉树中该节点的右孩子。

森林转换为二叉树

- 思路
  - 由于根没有兄弟，所以树转换为二叉树后，二叉树的根一定没有右子树。
  - 先将森林中的每一棵树转换为二叉树；
  - 再将第一棵二叉树的根作为转换后二叉树的根；
  - 第一棵二叉树的左子树作为转换后二叉树的左子树；
  - 第二棵二叉树作为转换后二叉树的右子树；
  - 第三棵二叉树作为转换后二叉树根的右孩子的右子树
  - 依此类推

森林转换为二叉树





本章结束