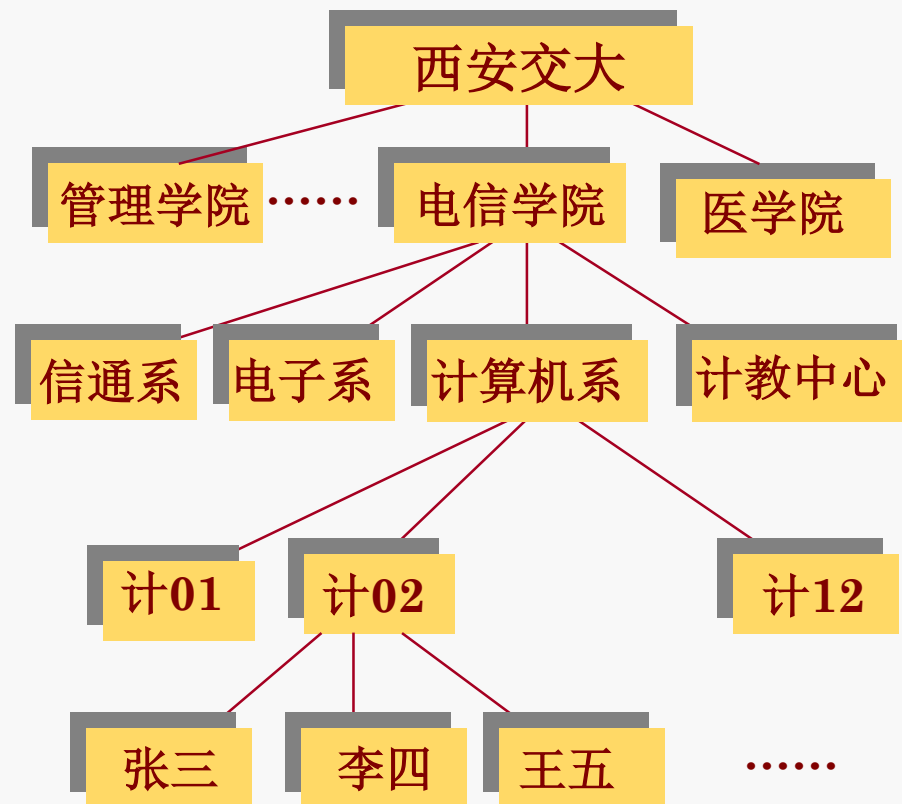


问题求解与实践 ——树和二叉树

主讲教师： 陈雨亭、沈艳艳

树的基本概念

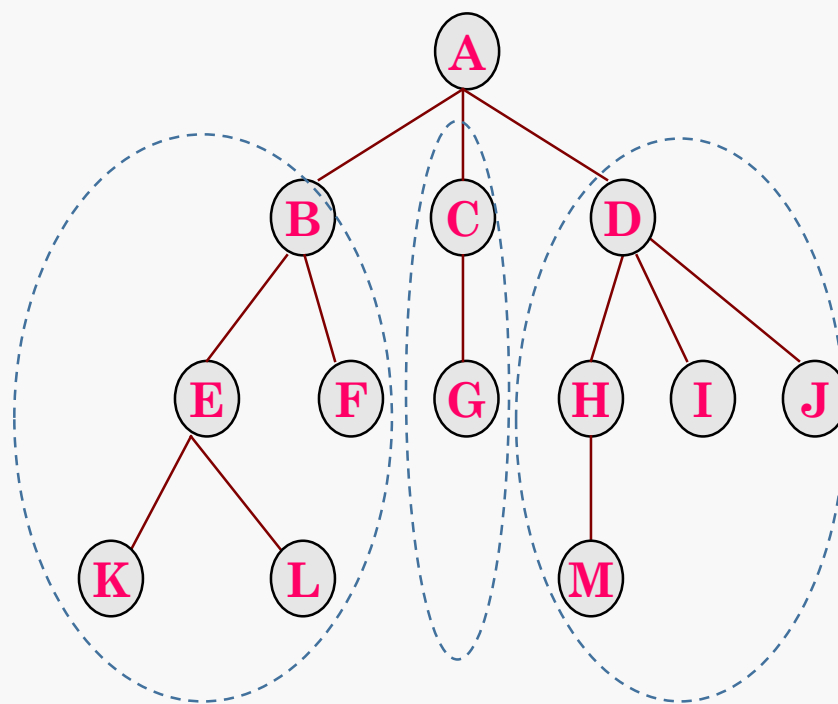
- 树形结构是逐层向下分支定义的层次结构
- 树形结构广泛存在客观世界中：家谱、行政区域划分、各种社会组织机构、操作系统中的目录等



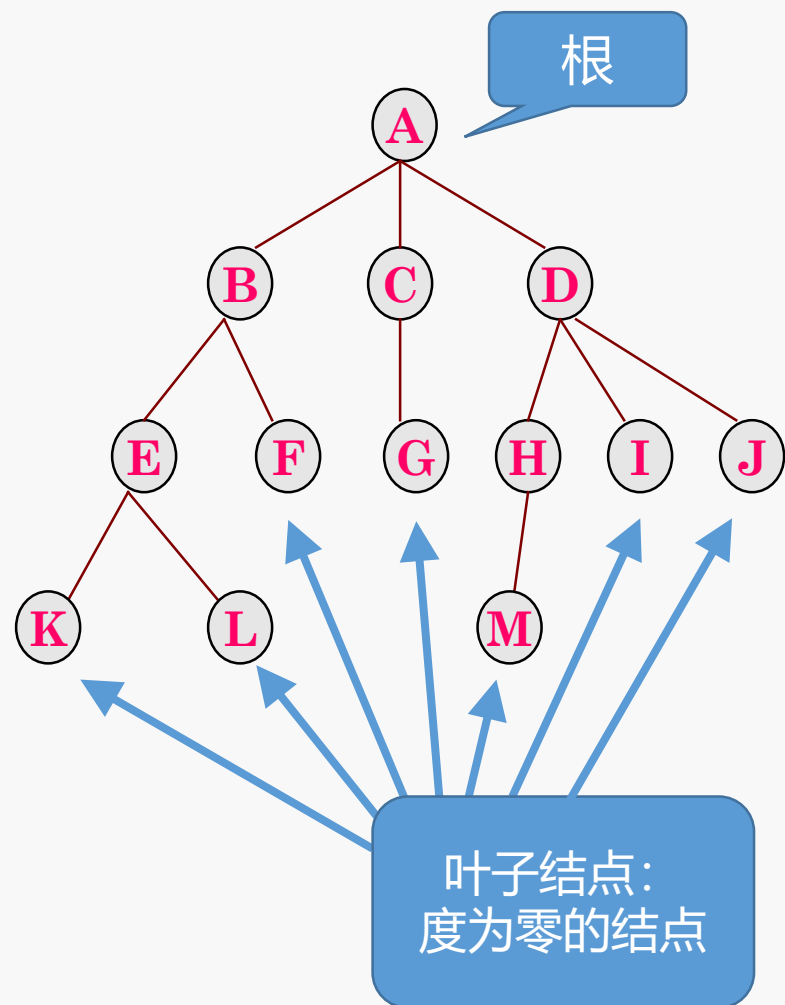
树的定义

树是一个或多个结点组成的有限集合 T ，有一个特定结点称为**根**，其余结点分为 m ($m \geq 0$) 个互不相交的集合 T_1, T_2, \dots, T_m 。每个集合又是一棵树，被称为这个根的**子树**。

一般的树



树的有关术语



- ◆ 结点的度: 结点拥有的非空子树的个数。

结点 A 度=3 结点 C 度=1

- ◆ 树的度: 树中所有结点的度中的最大值。

这颗树 度=3

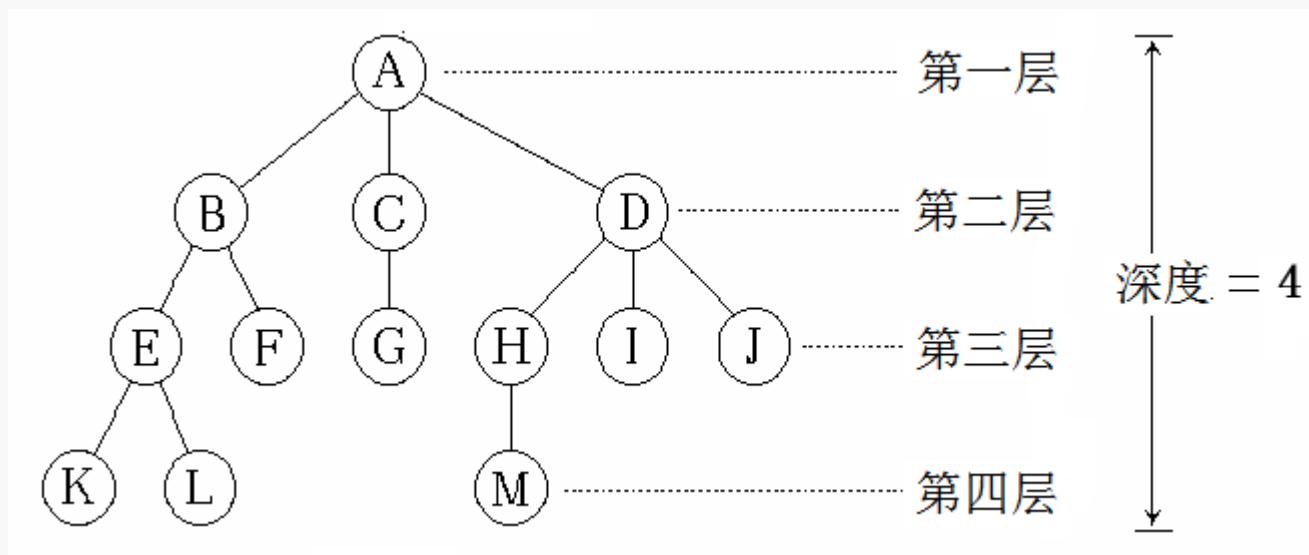
- ◆ 孩子结点和父结点: 某结点所有子树的根结点都称为该结点的孩子结点, 同时该结点也称为其孩子结点的父结点或双亲结点。

A 的子结点 有B、C、D

A是B、C、D父结点

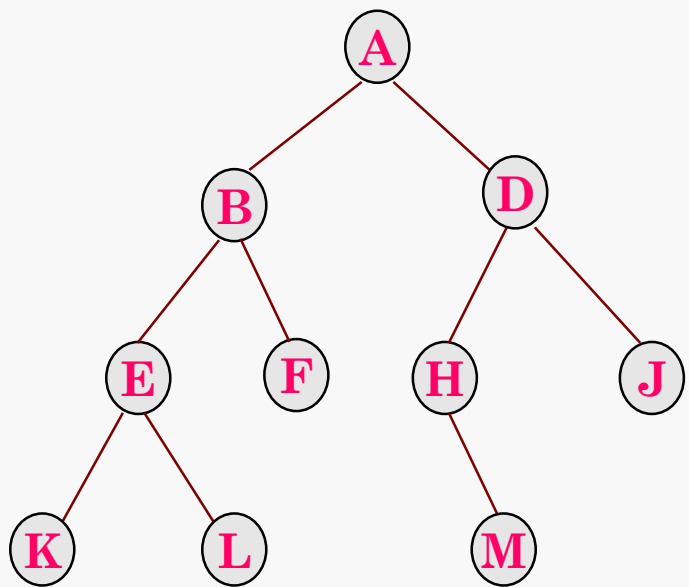
树的有关术语

- ◆ 结点的层次：根结点的层次为1,其子结点的层次为2。依次类推。
- ◆ 树的深度：树中结点所在的最大层次。
- ◆ 有序树和无序树：树中各结点的子树看成自左向右有序的，则称该树为有序树，否则称为无序树。



二叉树

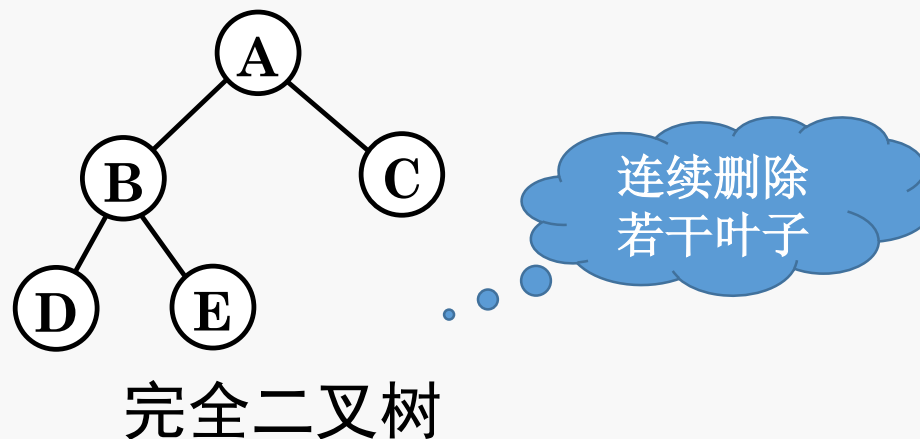
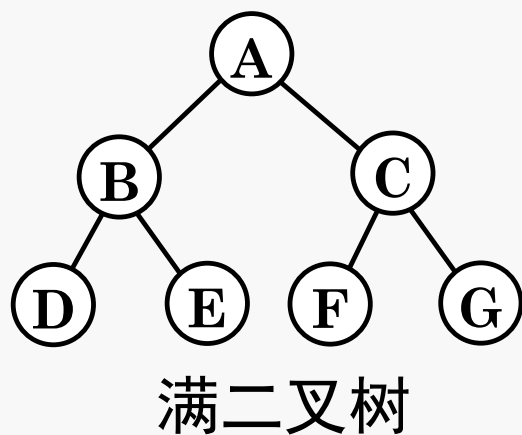
- ◆ 二叉树是每个节点最多有两个子树的树结构



- ◆ 二叉树是有序树，结点的子树分别称为根的左子树和右子树

特殊形式的二叉树

- ◆ **满二叉树**：当二叉树每个分支结点的度都是2，且所有叶子结点都在同一层上，则称其为满二叉树。
- ◆ **完全二叉树**：从满二叉树叶子所在的层次中，自右向左连续删除若干叶子所得到的二叉树被称为完全二叉树。满二叉树可看作是完全二叉树的一个特例。



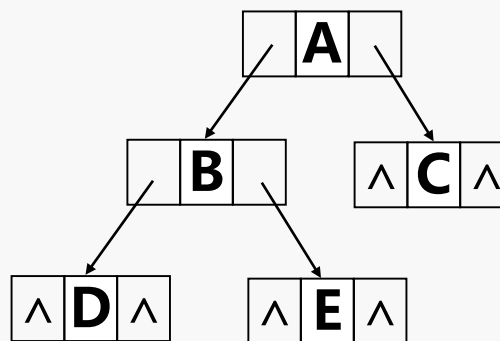
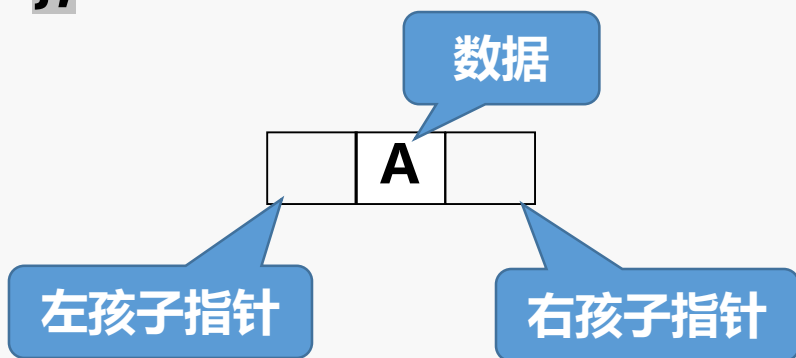
二叉树的实现——链式存储

- ◆ 二叉树是一种非线性数据结构，描述的是元素间一对多的关系，这种结构最常用、最适合的描述方法是用链表的形式

- ◆ 首先定义结点

每个结点都包含一个数据域和两个指针域。一般可采用下面的形式定义结点：

```
struct BinTreeNode {  
    char data;           // 这里以字符型的数据为例  
    struct BinTreeNode *leftChild, *rightChild;  
};
```



二叉树的实现——链式存储

- ◆ 定义一颗二叉树就是定义一个空树，也就是定义一个空指针，可描述如下：

```
BinTreeNode *root; //定义根结点指针  
root=NULL;         //定义空树
```

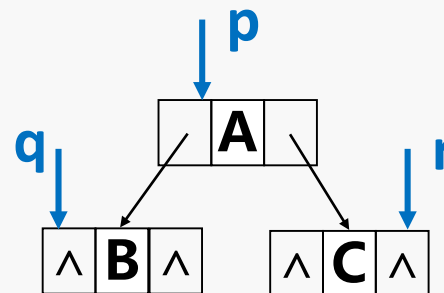
- ◆ 新建一个结点

```
BinTreeNode *p = new BinTreeNode;  
p->data = 'A'; //给数据域赋值  
p->leftChild=NULL; //左子树为空  
p->rightChild=NULL; //右子树为空
```

必须通过结点指针操作

- ◆ 建立二叉树

```
..... 建立结点A、B、C（方法如上面所示）  
..... 指针p、q、r 分别指向结点A、B、C  
p->leftChild=q; //左孩子为B  
p->rightChild=r; //右孩子为C
```



问题1：输入与输出

1. 用户如何（手工或者利用文件）向计算机程序输入一棵树？可能会有什么输入异常？
2. 如何输出一棵树到外部文件里？
3. 如何展示一棵树？

问题2：树的读写

- 令输入 $\langle a, b \rangle$ 代表 **b是a的父结点**
- 针对一组数据
 - 计算该组数据可以形成多少棵树？
 - 输入任意一个结点，提供两个成员函数，计算其父结点和子结点集？
 - 提供操作，可以修改树。操作与向输入中增加或者删除一个 $\langle x, y \rangle$ 效果一致
 - 检查是否每一棵树都是结构良好？

问题求解与实践 ——二叉树生成

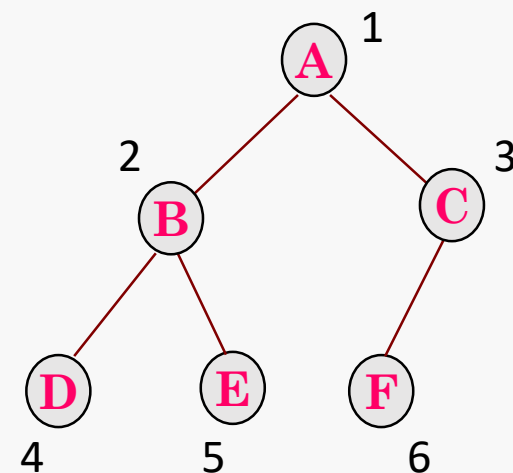
主讲教师： 陈雨亭、沈艳艳

完全二叉树的一个特性

将完全二叉树的每个结点从上到下、每一层从左至右进行1至n的编号

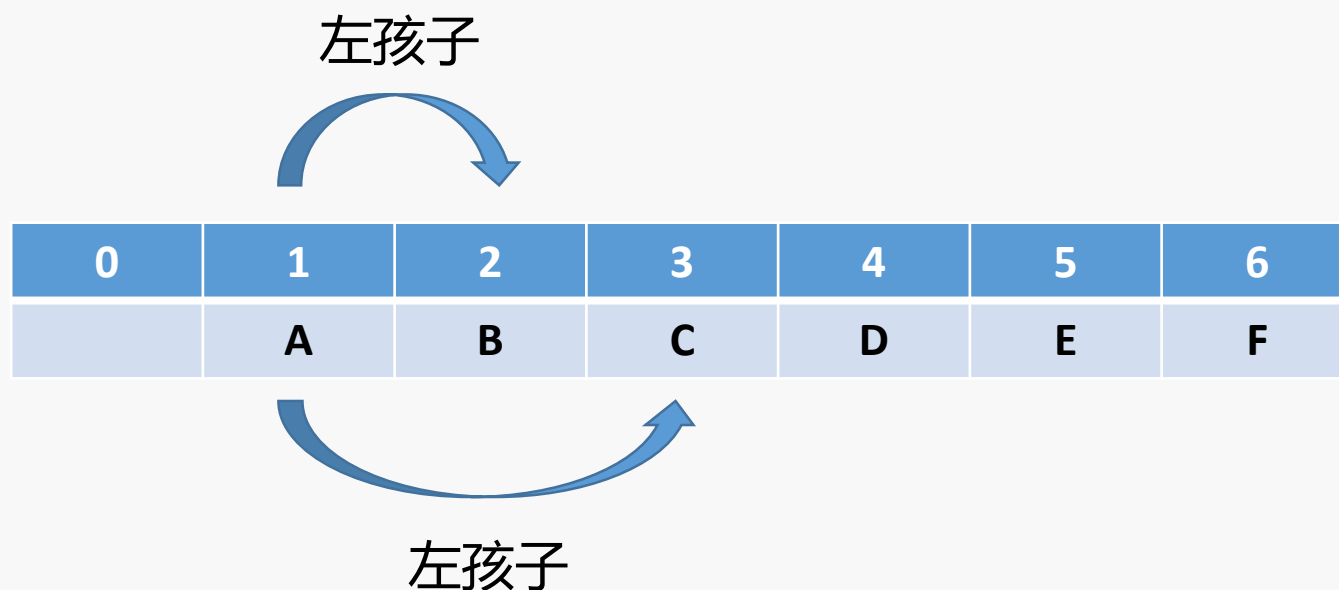
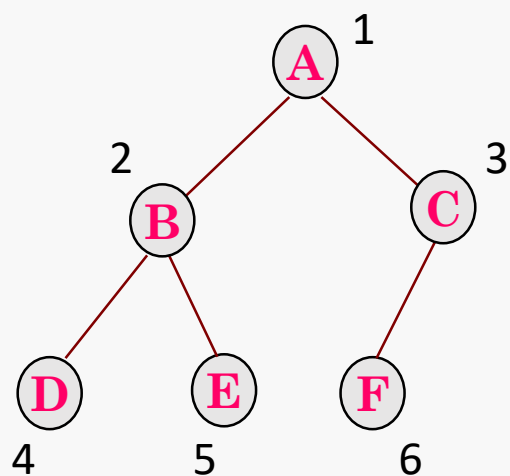
◆ 性质:

- ① 若 $i=1$ ，则该结点是二叉树的根，否则，编号为 $\lfloor i/2 \rfloor$ 的结点为结点 i 的父结点；
- ② 若 $2*i > n$ ，则该结点无左孩子。否则，编号为 $2*i$ 的结点为结点 i 的左孩子；
- ③ 若 $2*i+1 > n$ ，则该结点无右孩子。否则，编号为 $2*i+1$ 的结点为结点 i 的右孩子。

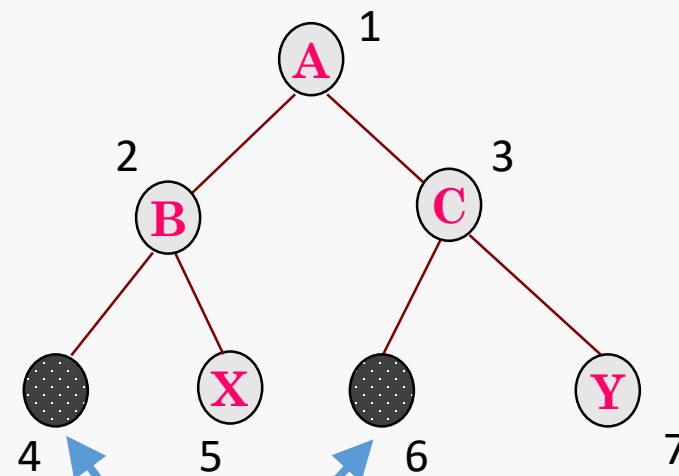
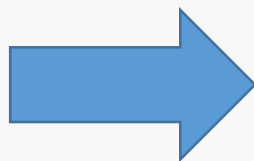
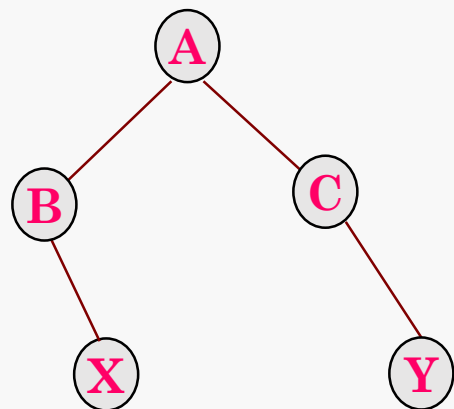


完全二叉树的顺序存储

可以用一维数组存储：空出数组下标为0的位置，将结点存储在下标为其编号的位置。



二叉树转换为完全二叉树后存储



添加虚结点

存储情况

0	1	2	3	4	5	6	7
	A	B	C	#	X	#	Y

生成一个二叉树

可以有很多方式生成一个二叉树，这里我们讨论的问题为：

根据数组中存储的完全二叉树，生成一个链式结构的二叉树

比如将下面数组转换为链式结构的二叉树

0	1	2	3	4	5	6	7
	A	B	C	#	X	#	Y

其中#代表虚
结点

生成一个二叉树

◆ 算法:

下标为1的元素作为根结点生成;

依次处理数组中的其他元素: 下标为 i 的元素, 若 i 为偶数就作为 $[i/2]$ 结点的左孩子, 若 i 为奇数就作为 $[i/2]$ 结点的右孩子生成;

遇到虚元素就跳过, 继续处理下一个元素。

◆ 问题:

1. 依次处理数组中元素, 循环结束条件是什么?

方法一: 预先得到结点总数 n (包括虚结点), 循环 n 次

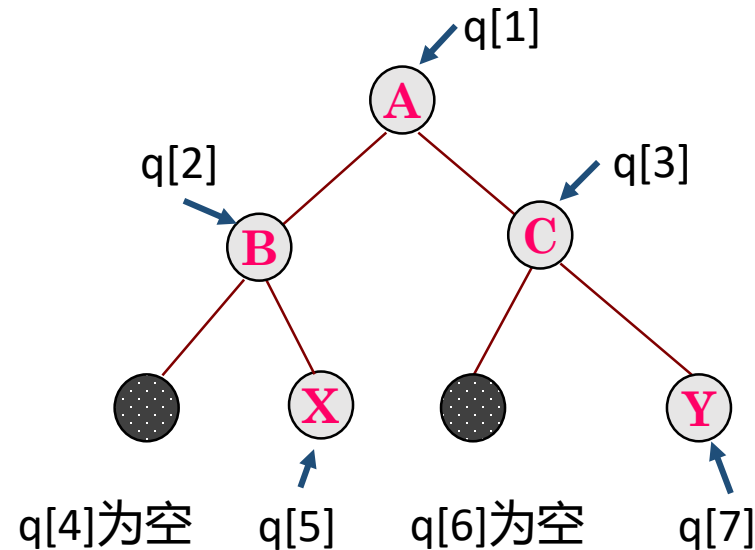
方法二: 在数组末尾加一个特殊符号 (比如 \$), 循环遇到该符号则停止

2. 如何让结点 i 和父结点 $[i/2]$ 连接起来?

在链式结构中, 由于对结点的操作要通过指针进行, 所以先要得到结点 $[i/2]$ 和结点 i 的指针 p 和 q , 而后可利用 $p \rightarrow \text{leftChild} = q$ 或 $p \rightarrow \text{rightChild} = q$ 连接即可

生成一个二叉树伪代码

```
BinTreeNode* create( char ch[] ) {  
    BinTreeNode* q[100];    //定义结点指针数组, 存放完全二叉树结点指针  
    BinTreeNode *s, *root;  //定义结点指针s、根结点指针root;  
    设下标 i 为 1;  
    while( ch[i] != '$' ) {    // 字符 '$' 在数组中最后一个元素后面, 结束循环  
        if (ch != '#') {      // 不是虚结点时  
            生成新结点( 指针为s ), 设初值, 并令 s->data = ch[i];  
            if(i==1) root = s; // s就是根指针  
            else {  
                if(i为偶数) q[i/2]->leftChild = s;  
                else q[i/2]->rightChild = s;  
            }  
            q[i] = s; // 将指针s 保存在q[i]中  
        } else q[i] = NULL;  
        i++; //计数器i加1, 准备处理下一个元素  
    }  
    返回root值;  
}
```



问题

1. 采用前面 $\langle a, b \rangle$ 这种方式，如何检查输入是一棵二叉树？
 1. 虚结点如何表示？
2. 你还能想到哪些二叉树的便捷的输入/输出形式？

问题求解与实践 ——二叉树遍历

主讲教师： 陈雨亭、沈艳艳

二叉树的遍历

- 二叉树遍历是按照某种顺序访问二叉树的每个结点，并且每个结点只被访问一次
- 这里“访问”的含义是指取出结点数据计算、输出等，或对结点数据进行修改等操作

二叉树的三种遍历方式

1. 先序遍历

首先访问根，然后按先序遍历方式访问左子树，再按先序遍历方式访问右子树

2. 中序遍历

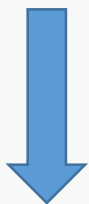
首先按中序遍历访问左子树，再访问根，最后按中序遍历方式访问右子树

3. 后序遍历

首先按后序遍历访问左子树，再按后序遍历方式访问右子树，最后访问根

二叉树先序遍历的实现

1. 首先访问根
2. 然后按先序遍历方式访问左子树
3. 再按先序遍历方式访问右子树

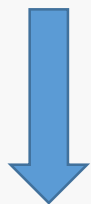


将以上步骤写成一个函数

```
先序遍历（根指针p）           //只能通过根的指针进入二叉树
{
    visit（p结点）；           // p结点——指针p指向的结点
    先序遍历（p->leftChild）；   // 先序遍历左子树
    先序遍历（p->rightChild）； // 先序遍历右子树
}
```

二叉树先序遍历的实现

1. 首先访问根
2. 然后按先序遍历方式访问左子树
3. 再按先序遍历方式访问右子树



将以上步骤写成一个函数

这个递归函数能终止吗？

```
PreOrder (指针p)           //访问以p为根的二叉树
{
    visit (p结点) ;          // 指针p指向的结点
    PreOrder (p->leftChild) ; // 先序遍历左子树
    PreOrder (p->rightChild) ; // 先序遍历右子树
}
```


二叉树先序遍历的实现

先序遍历算法如下：

```
void PreOrder(BinTreeNode *p) {
```

```
    if (p) {  若遇到空指针则返回上一层函数
```

```
        Visit(p );           //访问根结点
```

```
        PreOrder(p->leftChild );  //先序遍历左子树
```

```
        PreOrder(p->rightChild ); //先序遍历右子树
```

```
    }
```

```
}
```

问题求解与实践 ——哈夫曼树

主讲教师： 陈雨亭、沈艳艳

哈夫曼树

◆ 什么是路径、路径长度？

在一棵树中，从一个结点往下到另一个结点之间的通路，称为路径。通路中分支的数目称为路径长度。

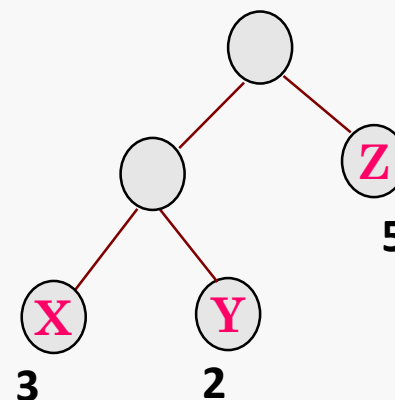
◆ 二叉树带权路径长度

设二叉树有 n 个带有权值的叶子结点，每个叶子到根的路径长度乘以其权值之和称为二叉树带权路径长度。记作：

$$WPL = \sum_{i=1}^n w_i * l_i$$

w_i — 第 i 个叶子的权重

l_i — 第 i 个叶子到根的路径长度



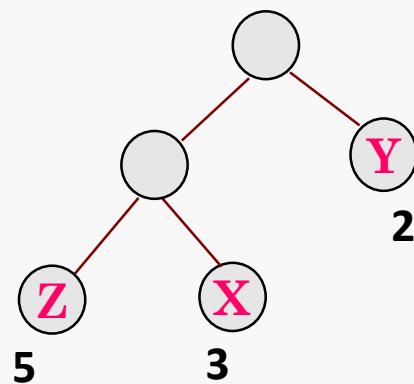
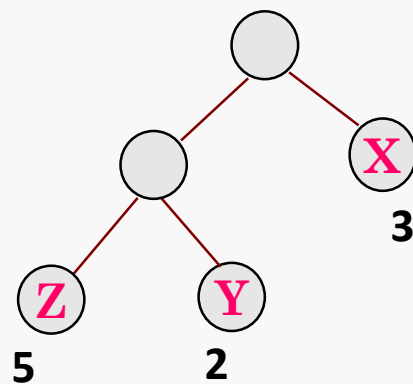
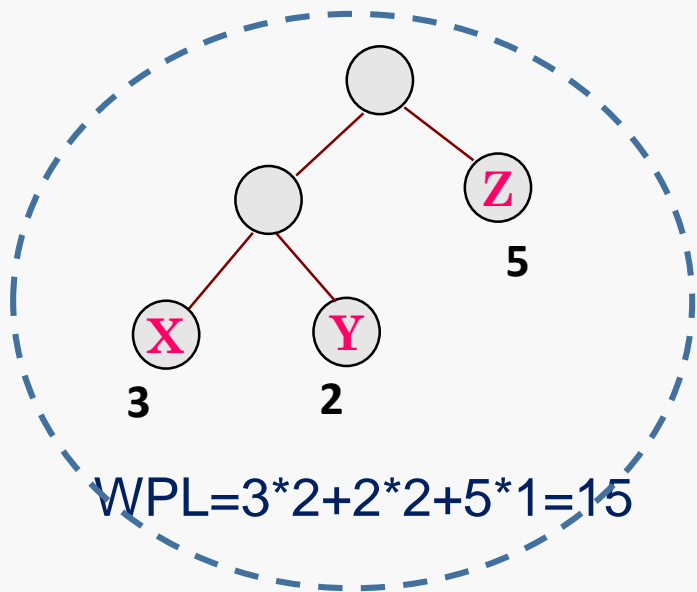
$$WPL = 3 * 2 + 2 * 2 + 5 * 1 = 15$$

哈夫曼树

◆ 什么哈夫曼树?

以一些带有固定权值的结点作为叶子所构造的，具有最小带权路径长度的二叉树。

设X、Y、Z权值为3、2、5，可以构造多种叶子含权的二叉树，例如

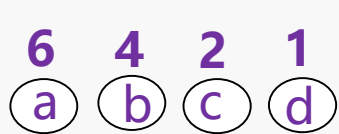


哈夫曼树的构造过程

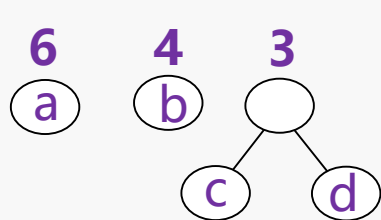
假定有 n 个具有权值的结点，则哈夫曼树的构造算法如下：

- ① 根据 n 个权值，构造 n 棵二叉树，其中每棵二叉树中只含一个权值为 w_i 的根结点；
- ② 在所有二叉树中选取根结点权值最小的两棵树，分别作为左、右子树构造一棵新的二叉树，这棵新的二叉树根结点权值为其左、右子树根结点的权值之和；删去原来的两棵树，留下刚生成的新树；
- ③ 重复执行②，直至最终合并为一棵树为止。

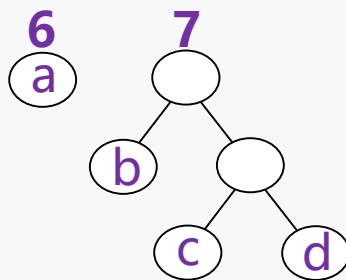
假定有a、b、c、d四个字符，它们的使用权重比为6 : 4 : 2 : 1



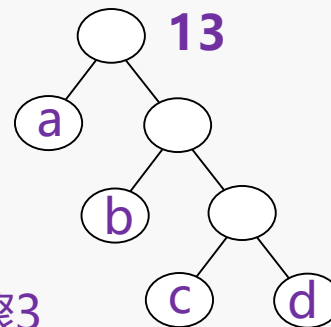
初始状态



步骤1



步骤2



步骤3

哈夫曼树与哈夫曼编码

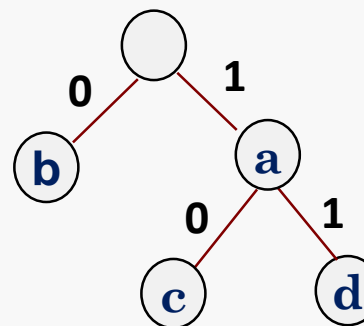
问题

假定有一段报文由a、b、c、d四个字符构成，它们的使用频率比为6 : 4 : 2 : 1，请构造一套二进制编码系统，使得报文翻译成二进制编码后**无二义性**且**长度最短**

任一编码方案

a -1 b-0 c-10 d-11

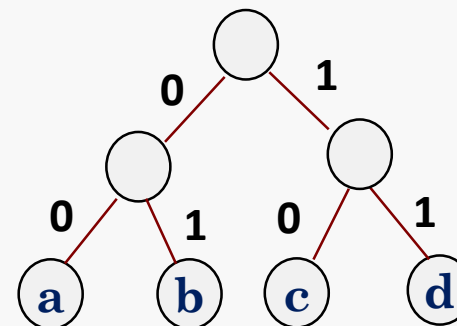
一一对应



无二义性要求任一编码不能是另一个编码的前缀；

无二义性编码对应

若a为1, d为11, 则a为d的前缀。这时111可以理解为 ad, da, aaa



- 字符为叶子
- 其他结点不包含字符

哈夫曼树与哈夫曼编码

问题

假定有一段报文由a、b、c、d四个字符构成，它们的使用频率比为6 : 4 : 2 : 1，请构造一套二进制编码系统，使得报文翻译成二进制编码后**无二义性**且**长度最短**

最优编码方案

对应

哈夫曼树

哈夫曼编码

a-0 b-10

c-110 d-111

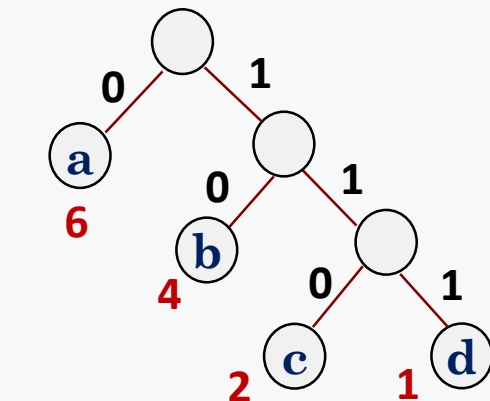
长度最短 要求
下面式子的值最小

$$\sum_{i=1}^4 w_i * l_i$$

加权平均
长度的4倍

w_i : 第 i 个字符权重

l_i : 第 i 个字符编码长度



$$\sum_{i=1}^4 w_i * l_i \text{ 最小}$$

w_i : 第 i 个叶子权重

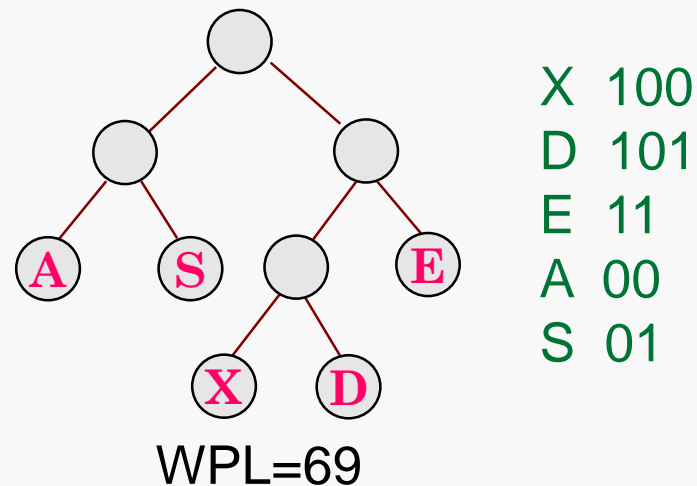
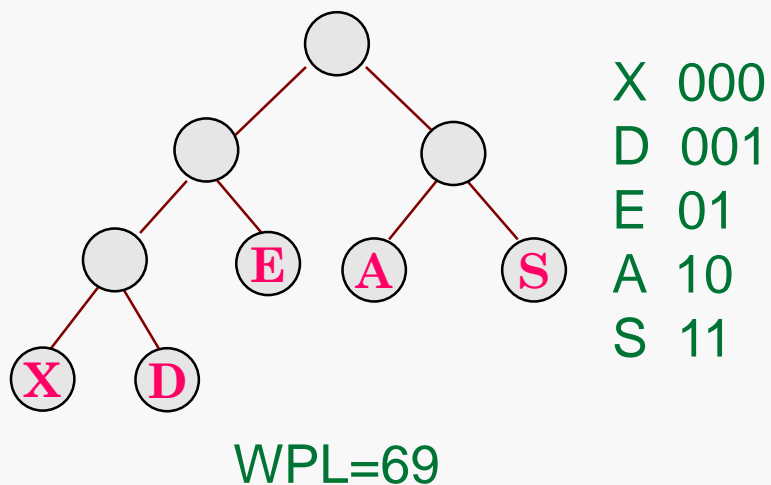
l_i : 第 i 个叶子到根的路径长度

哈夫曼树编程分析

◆ 以下面的问题为例进行分析

设一段文本由字符 X, S, D, E, A 构成，它们的使用权重为 2: 9: 5: 7: 8，请以这些字符构造哈夫曼树，并求出它们的哈夫曼编码

◆ 注意：哈夫曼树及哈夫曼编码不是唯一的



哈夫曼树结点定义

◆ 结点要存储字符、权重

```
struct HNode
{
    char    data;           //字符数据
    int     weight;         //权重
    struct  HNode *lchild;  //左孩子指针
    struct  HNode *rchild;  //右孩子指针
};
```

初始 n 棵单根树构造

//假定结点数不超30 (只用前 5 棵树)

HNode *h[30], *root;

//让h[i]指向第i棵单根树

for (int i = 0; i<30; i++) h[i] = new HNode;

//为每个单根树赋权值、字符

char ch[] = { 'X','S','D','E','A' };

int weight[] = { 2,9,5,7,8 };

for (int i = 0; i<5; i++) {

h[i]->data = ch[i];

h[i]->weight = weight[i];

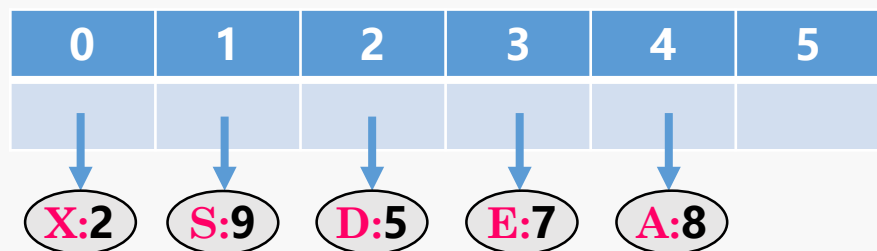
h[i]->lchild = NULL;

h[i]->rchild = NULL;

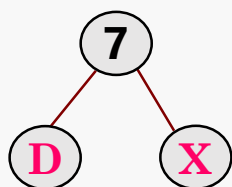
}

哈夫曼树合并生成过程

第一周期

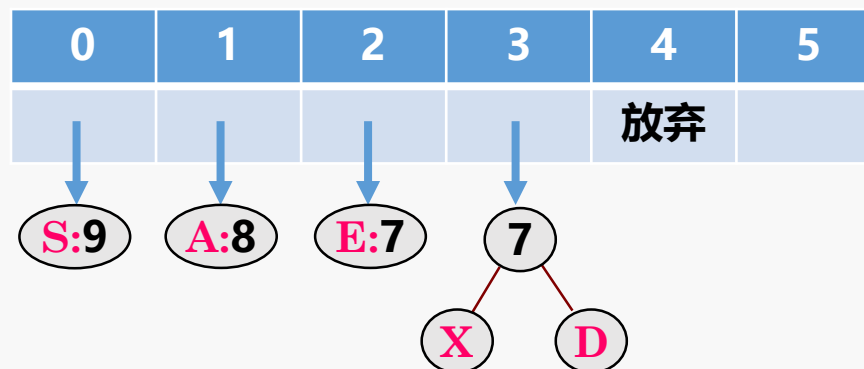
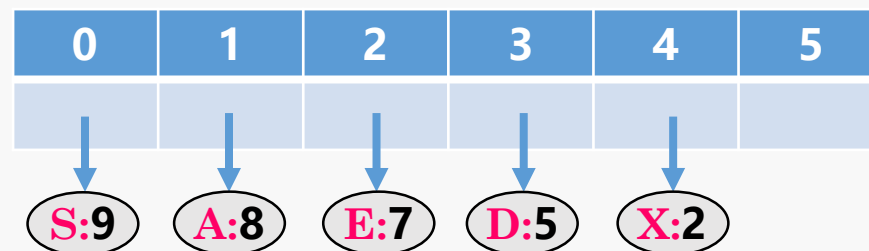


生成子树



放回数组

排序



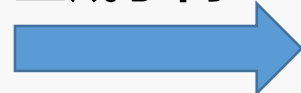
第二周期

排序



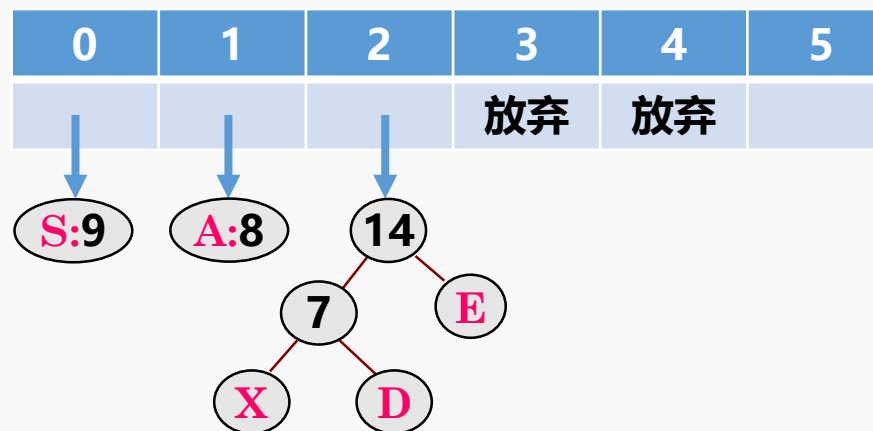
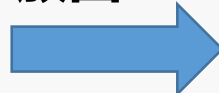
.....

生成子树



.....

放回



按权值排序函数Sort

将数组 h[] 前 n 项按权值排序（冒泡）

```
void Sort(HNode* h[], int n)
{
    for(int i=1; i<n; i++)
        for (int j = 0; j < n - i; j++)
        {
            if (h[j]->weight < h[j + 1]->weight)
            {
                HNode *t = h[j];
                h[j] = h[j + 1];
                h[j + 1] = t;
            }
        }
}
```

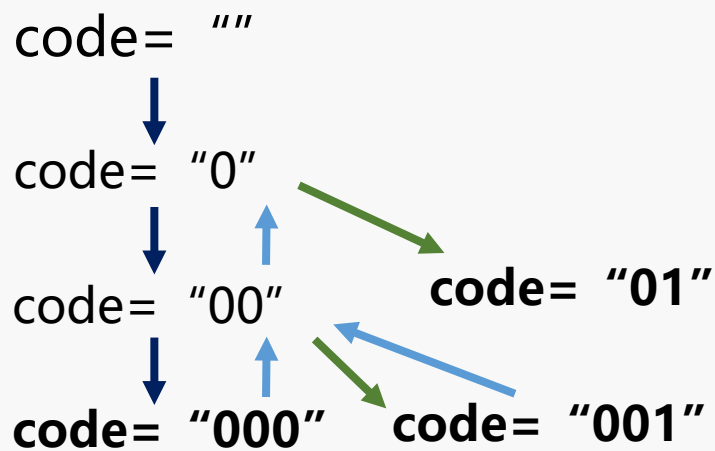
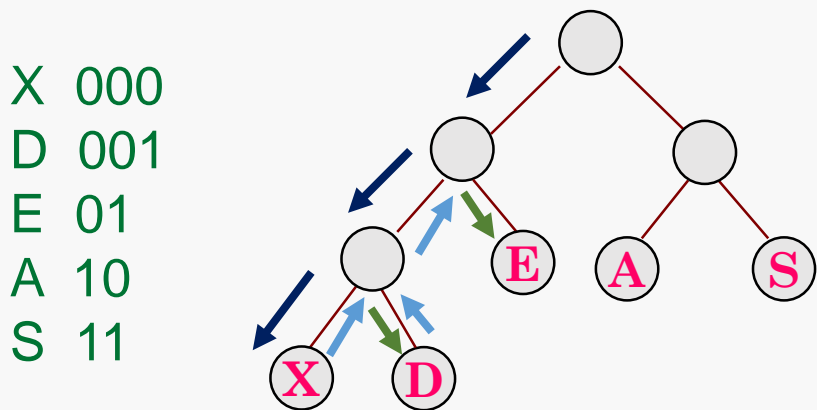
合并生成哈夫曼树

◆ 关键代码分析

```
.....  
while(n>1)      // 合并 n-1 次  
{  
    Sort(h, n);  //将数组 h[] 前 n 项按权值排序  
  
    HNode* s = new HNode;      //生成子树的根  
    s->data = ' ';  
    s->lchild = h[n-1];         //插入左子树  
    s->rchild = h[n-2];         //插入右子树  
    s->weight = h[n-1]->weight + h[n-2]->weight; //子树权值  
  
    h[n - 2] = s;              //放回数组 h[] 中  
    n = n - 1;  
}  
  
//哈夫曼树的根指针就是 h[0]
```

生成哈夫曼编码

◆ 利用二叉树先序遍历构造编码



每次进入**左孩子** code 尾部**添加 0**

每次从左孩子退回上一级 code 尾部**截掉1位数**

每次进入**右孩子** code 尾部**添加 1**

每次从右孩子退回上一级 code 尾部**截掉1位数**

生成哈夫曼编码

◆ 关键代码分析

```
char code[100] = "";  
void PreOrder( HNode *t )           // 对二叉树t进行先序遍历  
{  
    if (t) {  
        PreOrder(t->lchild);        //先序遍历左子树  
        PreOrder(t->rchild);        //先序遍历右子树  
    }  
}
```

生成哈夫曼编码

◆ 关键代码分析

```
char code[100] = "";  
void PreOrder( HNode *t )           // 对二叉树t进行先序遍历  
{  
    if (t) {  
        进入左孩子 code 尾部添加 0  
        PreOrder(t->lchild);         //先序遍历左子树  
        退回上一级 code 尾部截掉1位数  
        进入右孩子 code 尾部添加 1  
        PreOrder(t->rchild);         //先序遍历右子树  
        退回上一级 code 尾部截掉1位数  
        如果t指向叶子，则  
        输出 字符 和 code  
    }  
}
```


生成哈夫曼编码

◆ 关键代码分析

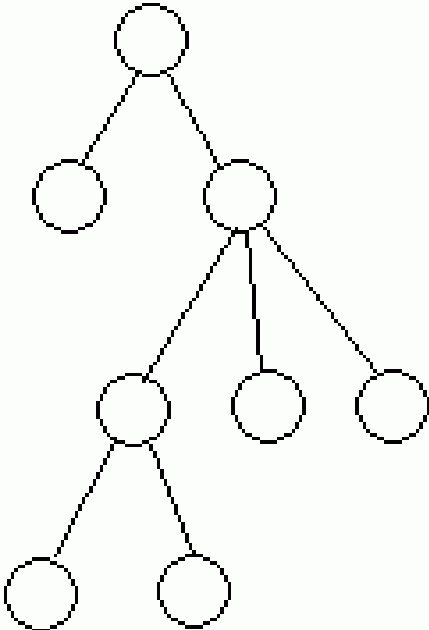
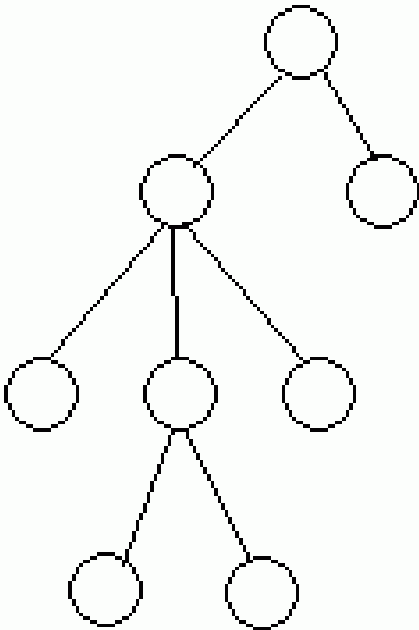
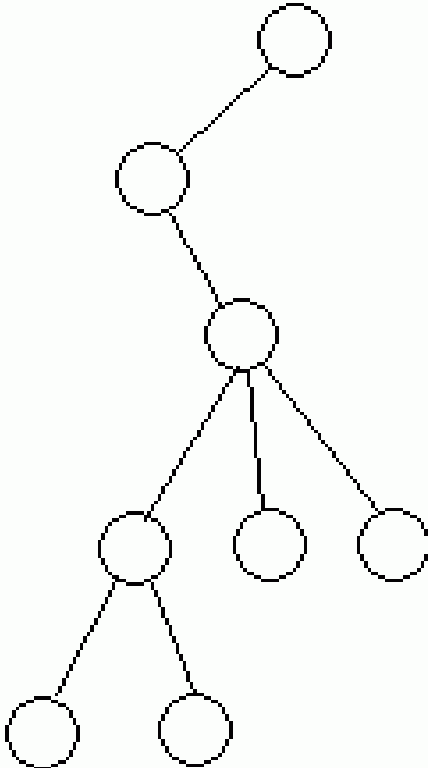
```
char code[100] = "";  
void PreOrder( HNode *t )           // 对二叉树t进行先序遍历  
{  
    if (t) {  
        进入左孩子 code 尾部添加 0  
        PreOrder(t->lchild);         //先序遍历左子树  
        退回上一级 code 尾部截掉1位数  
  
        进入右孩子 code 尾部添加 1  
        PreOrder(t->rchild);         //先序遍历右子树  
        退回上一级 code 尾部截掉1位数  
  
        如果t指向叶子, 则  
        输出 字符 和 code  
    }  
}
```

问题求解与实践 ——树的同构

主讲教师： 陈雨亭、沈艳艳

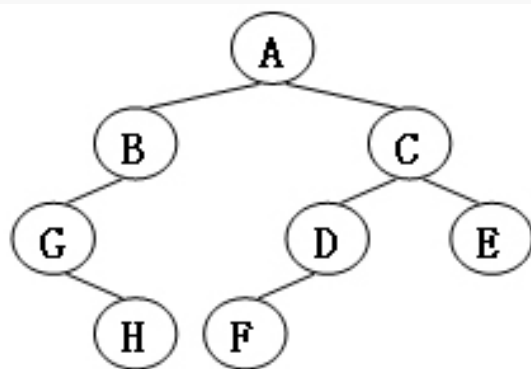
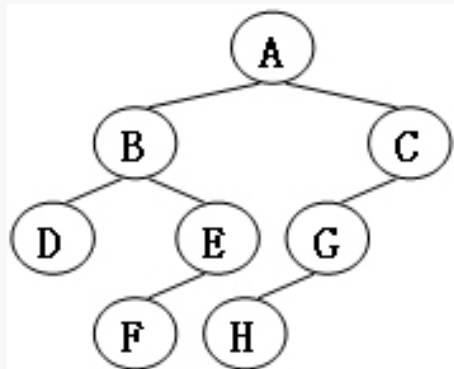
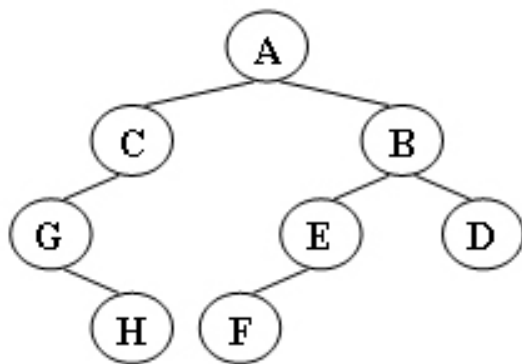
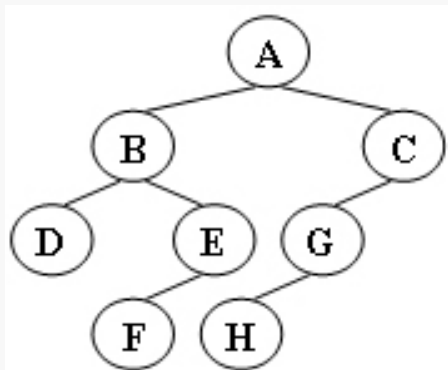
问题描述

- 给出两棵有根树，判断是否同构？

		
<i>Tree T</i>	✓ <i>isomorphic to T</i>	✗ <i>NOT isomorphic to T</i>

树的同构（简化版）

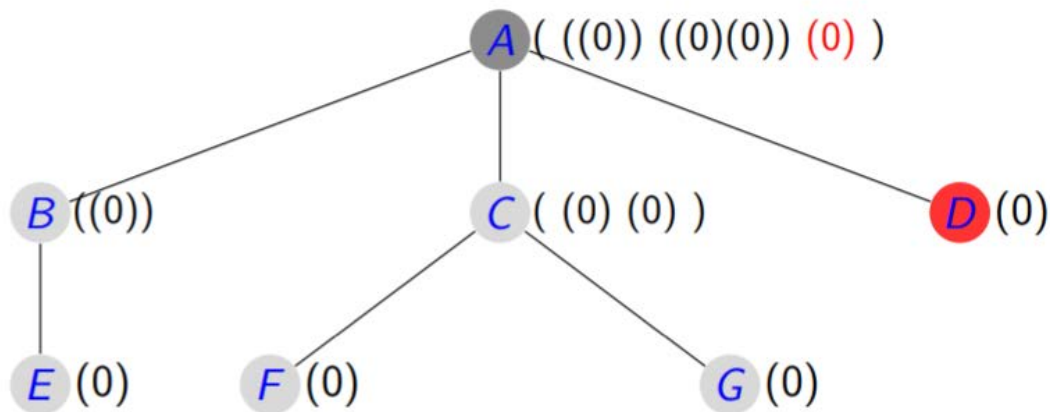
- 给定两棵树T1和T2。如果T1可以通过若干次左右孩子互换就变成T2，则我们称两棵树是“**同构**”的



思路：设计一个树到序列的映射，保证同构的树映射到相同的序列，不同构的树映射到不同的序列。如果要判断两棵树同构，只需检查它们对应的序列是否相同

树同构的AHU algorithm

- Let's assign parenthetical tuples to all tree vertices.



ASSIGN-KNUTH-TUPLES(v)

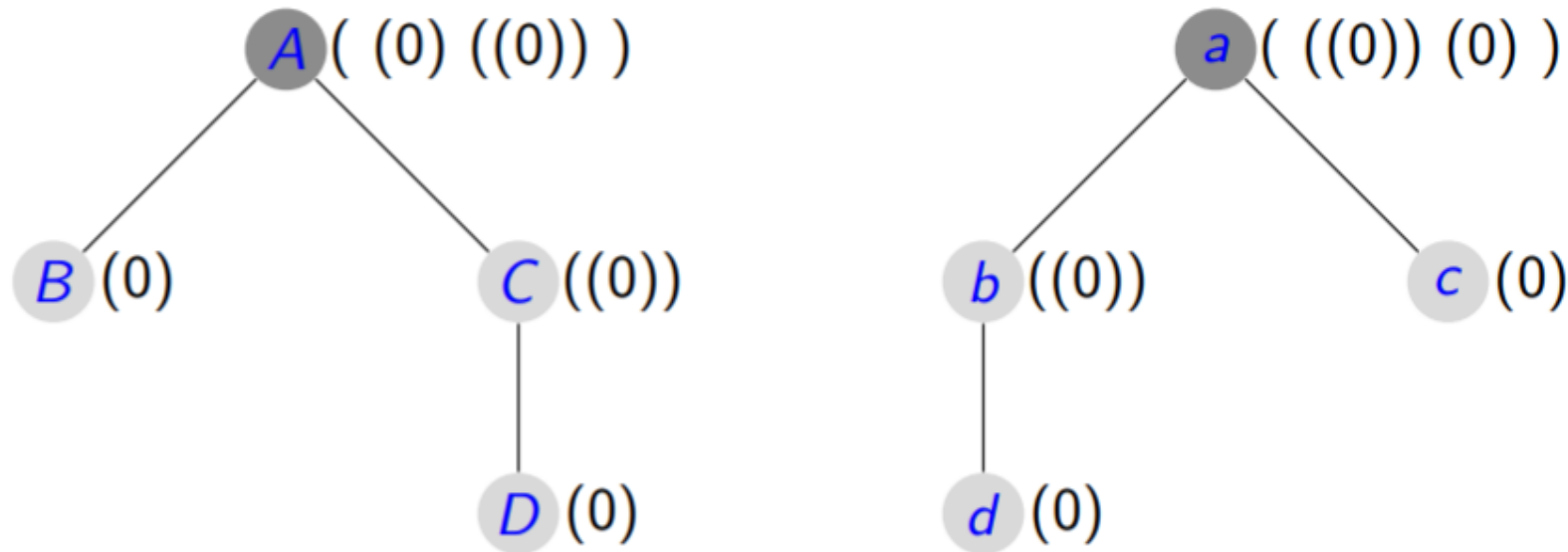
- 1: **if** v is a leaf **then**
- 2: Give v the tuple name (0)
- 3: **else**
- 4: **for all** child w of v **do**
- 5: ASSIGN-KNUTH-TUPLES(w)
- 6: **end for**
- 7: **end if**
- 8: Concatenate the names of all children of v to $temp$
- 9: Give v the tuple name $temp$

树同构的AHU algorithm

Observation

There is no order on parenthetical tuples.

Example



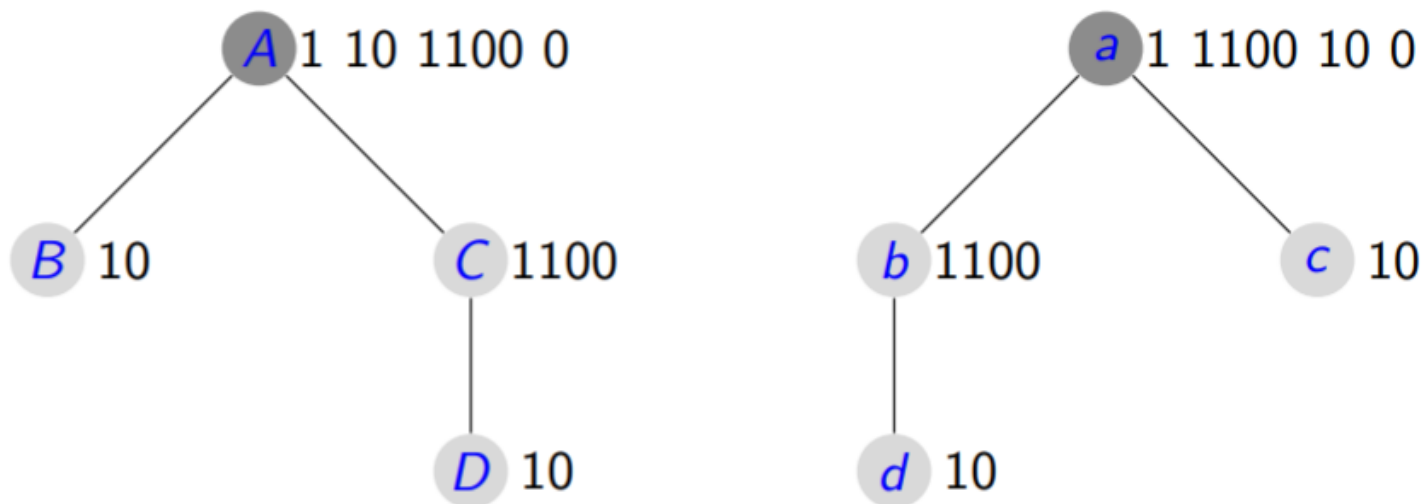
Let's convert parenthetical tuples to *canonical names*. We should drop all "0"-s and replace "(" and ")" with "1" and "0" respectively.

树同构的AHU algorithm

Observation

There is no order on parenthetical tuples.

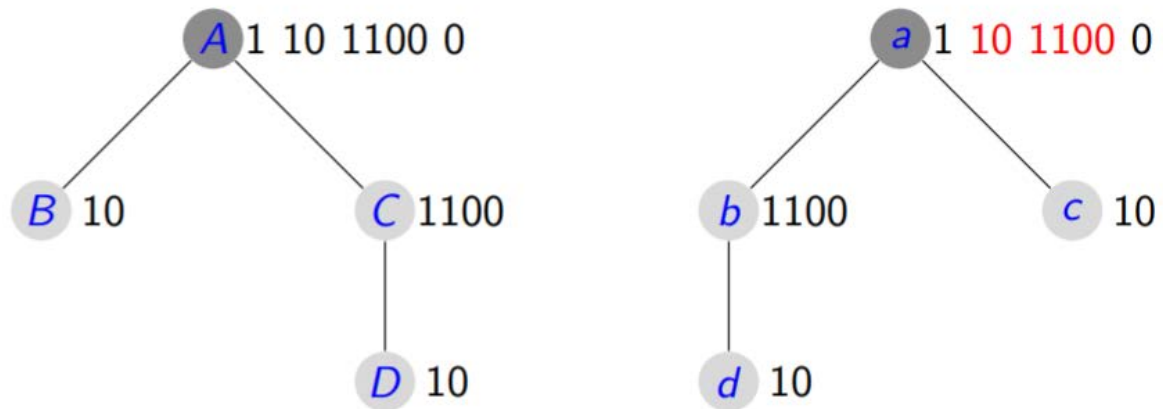
Example



Let's convert parenthetical tuples to *canonical names*. We should drop all "0"-s and replace "(" and ")" with "1" and "0" respectively.

树同构的AHU algorithm

Example



ASSIGN-CANONICAL-NAMES(v)

- 1: **if** v is a leaf **then**
- 2: Give v the tuple name "10"
- 3: **else**
- 4: **for all** child w of v **do**
- 5: ASSIGN-CANONICAL-NAMES(w)
- 6: **end for**
- 7: **end if**
- 8: Sort the names of the children of v
- 9: Concatenate the names of all children of v to $temp$
- 10: Give v the name $1temp0$

更详细的讨论，请参考：https://logic.pdmi.ras.ru/~smal/files/smal_jass08_slides.pdf