



离散数学

Discrete Mathematics

第二十八讲：根树与二叉树

吴楠

南京大学计算机学院



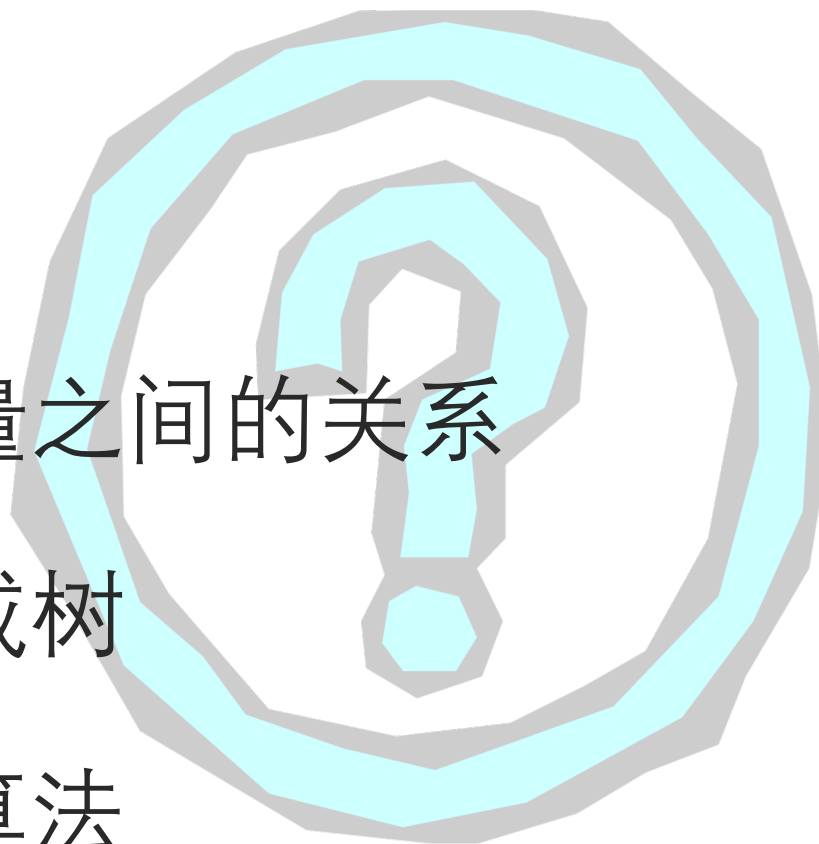
2025年6月3日



前情提要

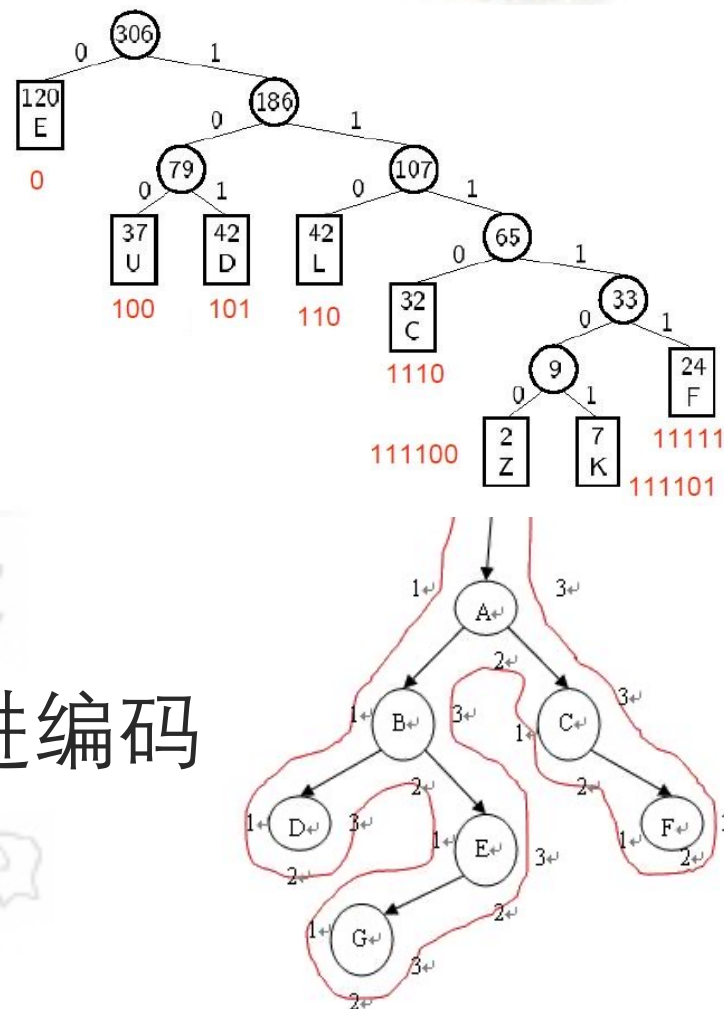


- 树的定义
- 树的连通性质
- 树中边和顶点数量之间的关系
- 生成树与最小生成树
- 求最小生成树的算法





- 根 树
- 二叉树
- 二叉树的应用
 - Huffman树与最优二进编码
- 二叉树的遍历*

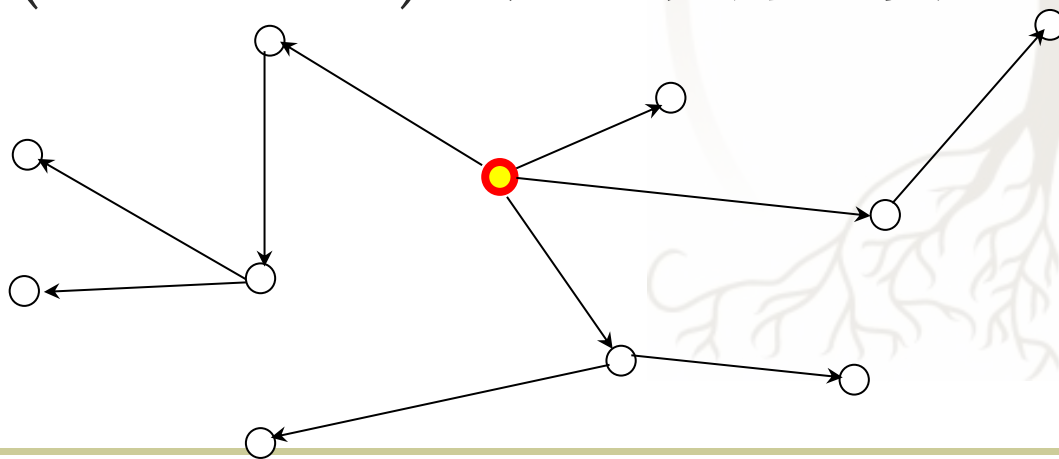




根树的定义



- **定义（有向树）**：底图（*i.e.*有向图去掉方向信息之后的无向图）为树的有向图称为**有向树**
- **定义（根树）**：若 $n(n \geq 2)$ 阶有向树恰含一个入度为0的顶点，其它顶点入度均为1，则该有向树称为**根树**(rooted tree)，其入度为0的顶点称为**根**

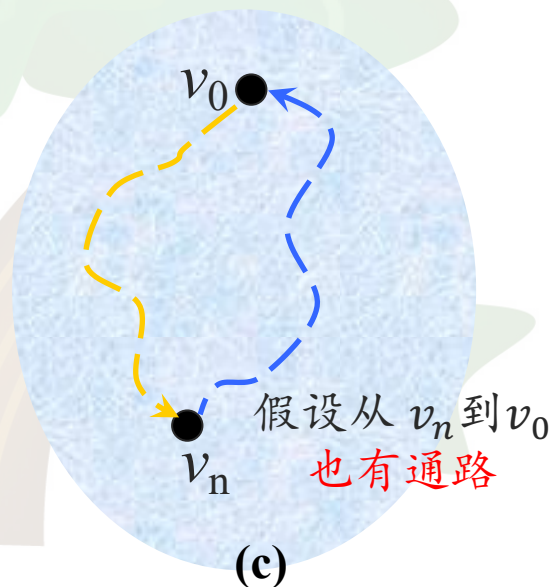
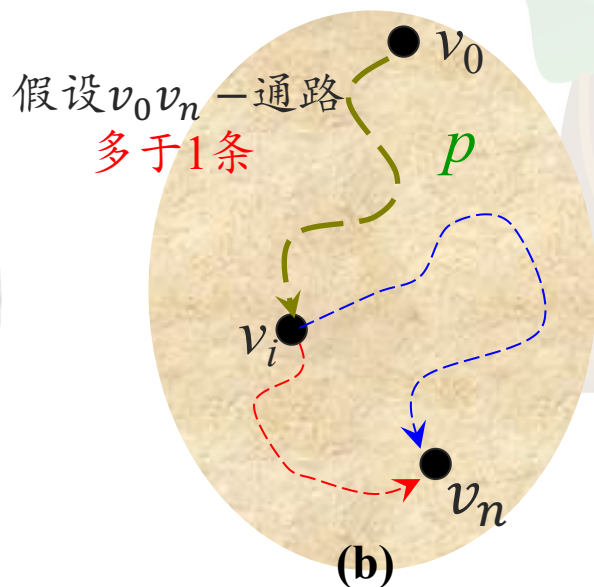
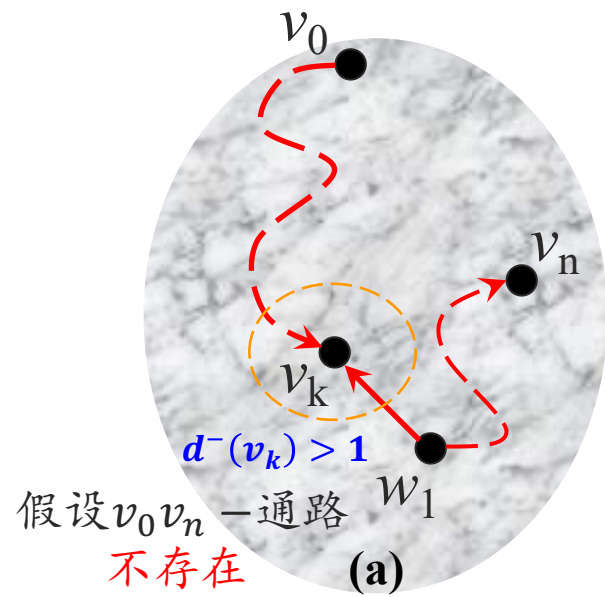




根树中的有向通路



- 若 v_0 是根树 T 中唯一的入度为0的顶点，则对 T 中任意其它顶点 v_n ，存在**唯一**的 v_0v_n -有向通路，但不存在 v_nv_0 -有向通路

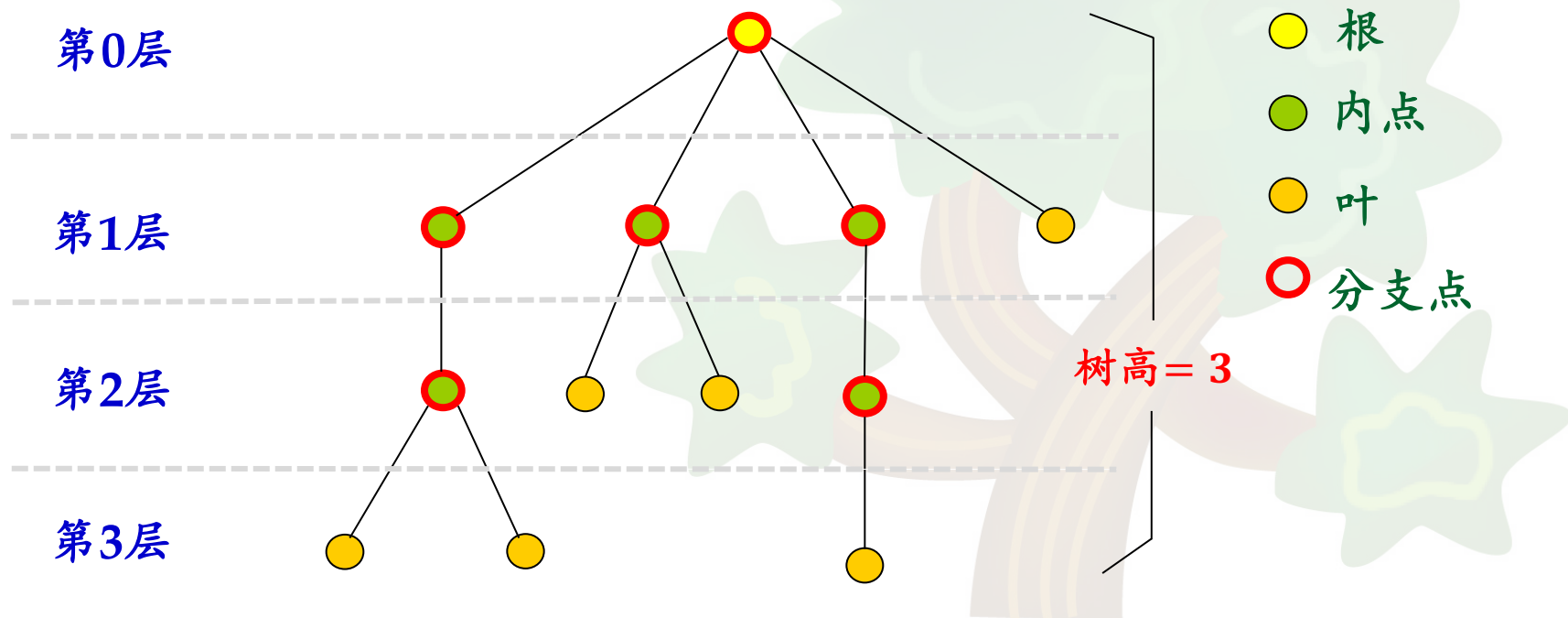




根树的表示



- 边上的方向用约定的位置(层次)关系表示

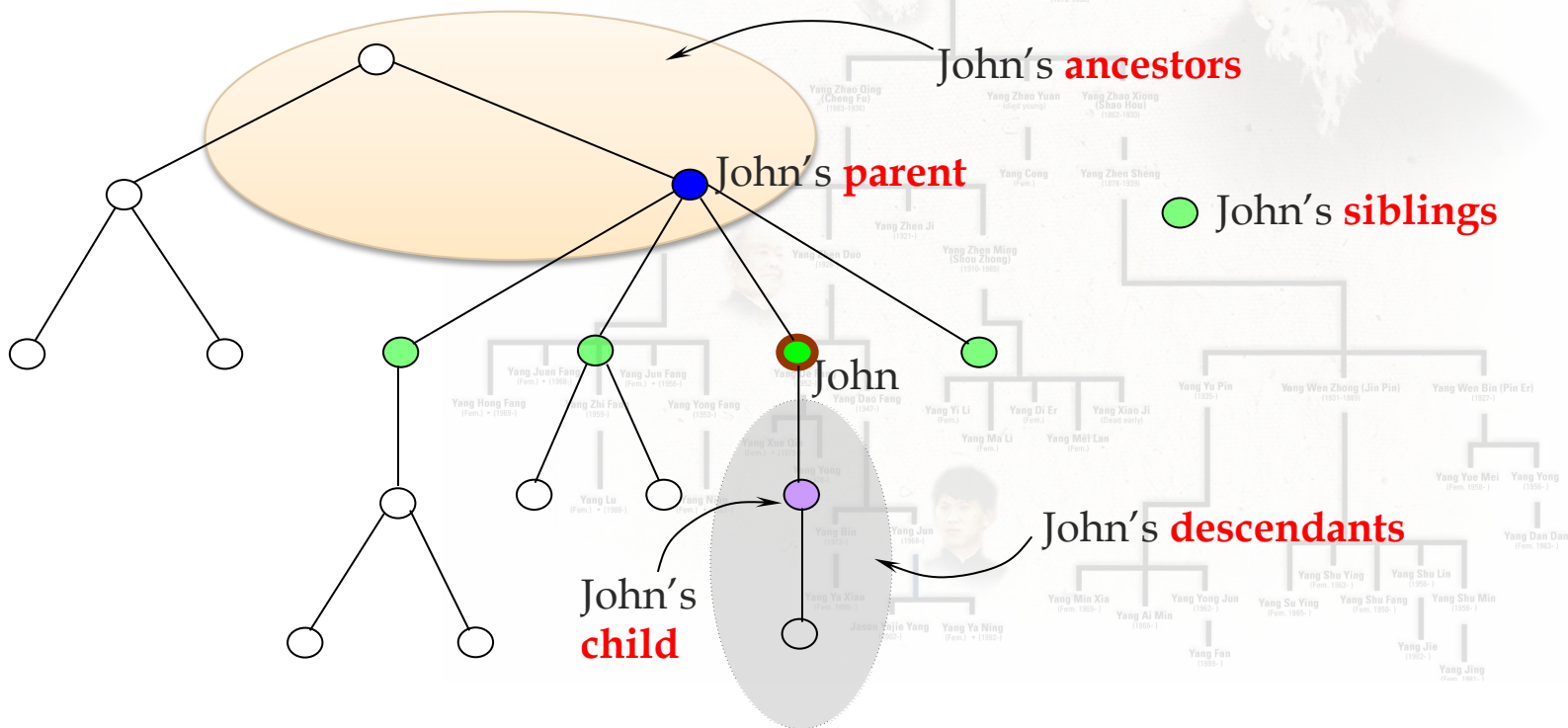




根树与家族关系(family tree)



- 用根树可方便地描述家族关系，同样地，家族关系术语常被用于描述根树中顶点间的关系

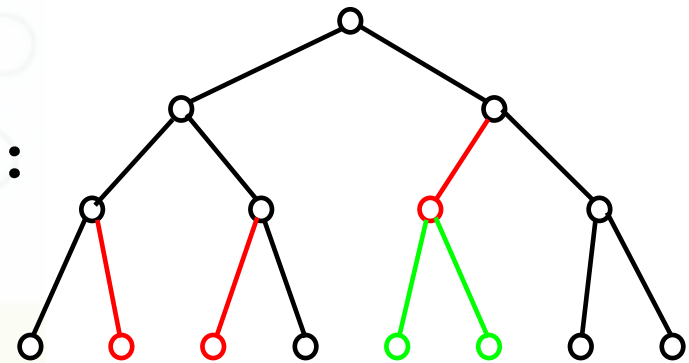




有关根树性质的术语



- **有序**：同层中每个顶点排定次序
- **r 叉**：每个分支点至多有 r 个孩子
- **r 叉正则**：每个分支点恰好有 r 个孩子
- **完全**：每个叶的层数恰等于树高
- 完全正则 r 叉树顶点数(k :树高):
$$(r^{k+1} - 1)/(r - 1)$$





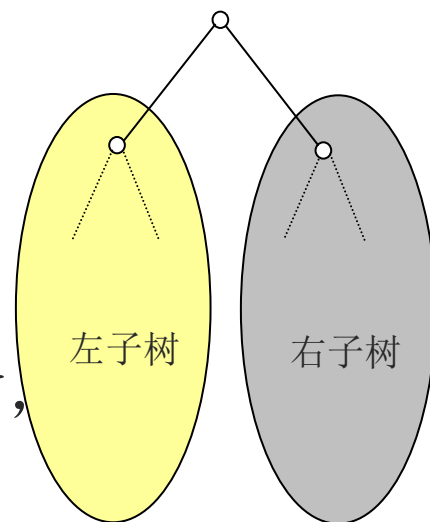
子树与二叉树(binary tree)



- **定义 (子树)**：设 T 是根树， T 中任一顶点 v 及其所有后代的**导出子图**也是根树(以 v 为根)，称为 T 的子树

- **有序二叉树**的子树分为**左子树**和**右子树**

- 即使不是正则二叉树，也可以分左、右子树，因此必须留意子树的根顶点的位置





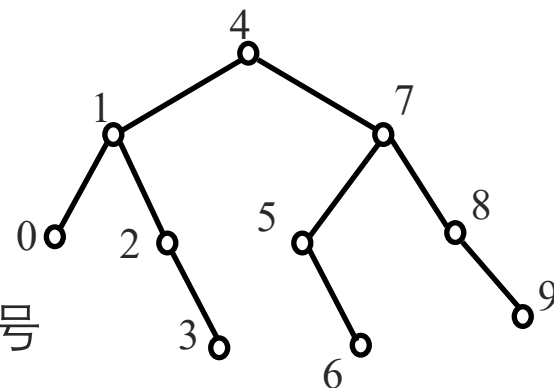
特殊的二叉树：二叉搜索树



■ 二叉搜索树 (binary search tree, BST)

是一种重要的**数据结构**，它满足下列条件：

- 每个顶点有一个唯一的标号，标号取自一个**全序集**
- 若 u 是树中任意的顶点，则：
 - u 的**左**子树中任意顶点的标号都**小于** u 的标号
 - u 的**右**子树中任意顶点的标号都**大于** u 的标号





特殊的二叉树：平衡二叉树



- **定义**（平衡二叉树，**AVL tree**）：（高度）平衡二叉树中**任意顶点**左右子树高度的差**不大于1** ⇨

- 假设（高度）平衡二叉树 T 中顶点个数为 n ， T 的高度为 h ，则 $h \in O(\log n)$

即： $\lim_{n \rightarrow \infty} \frac{h}{\log n} < \infty$

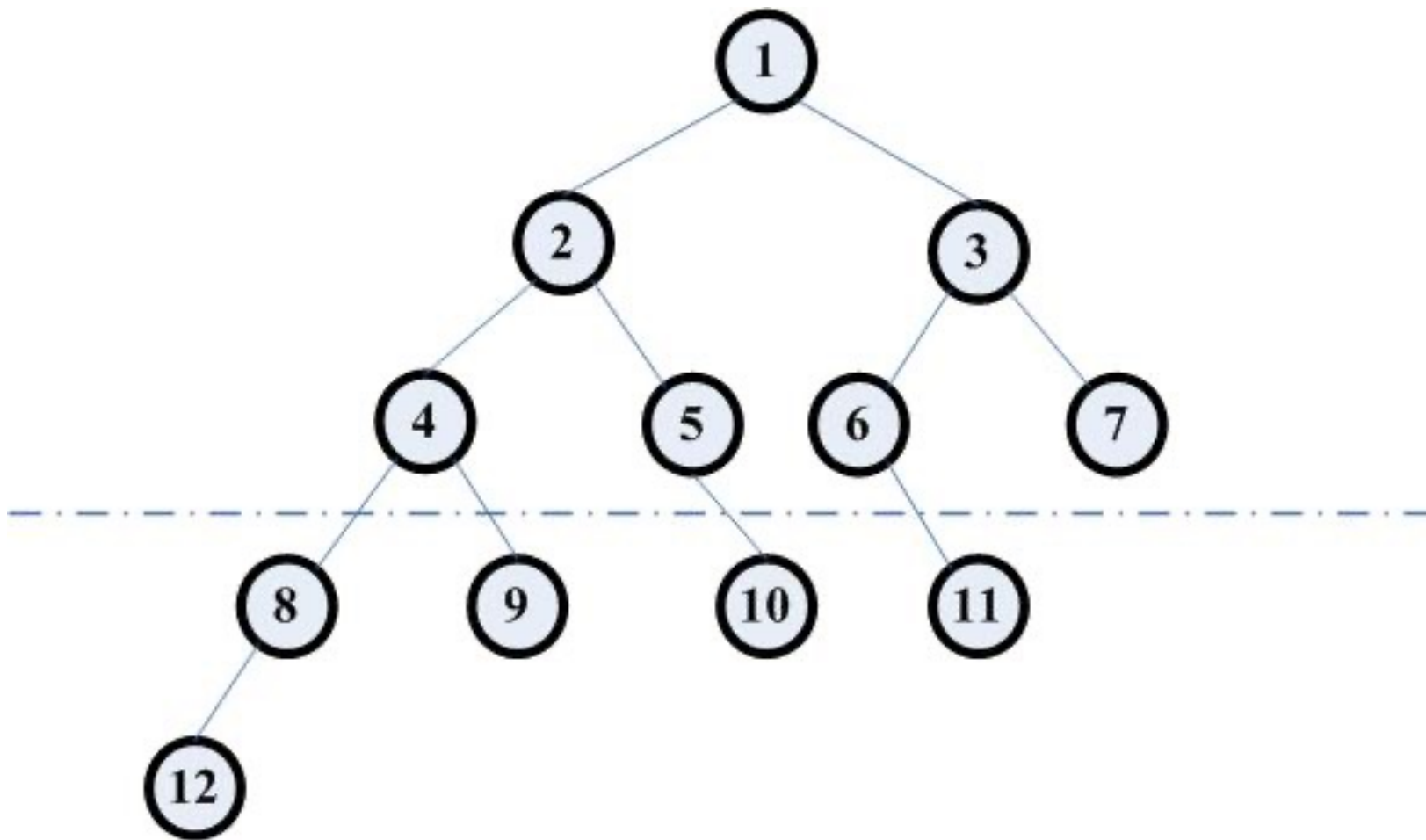
- 高度为 h 的（高度）平衡二叉树顶点数不小于：

$$\frac{1}{\sqrt{5}} \left(\frac{1 + \sqrt{5}}{2} \right)^{h+3} - 2$$

Note: $V(h) = 1 + V(h-1) + V(h-2)$



特殊的二叉树：平衡二叉树





平衡二叉树的旋转*

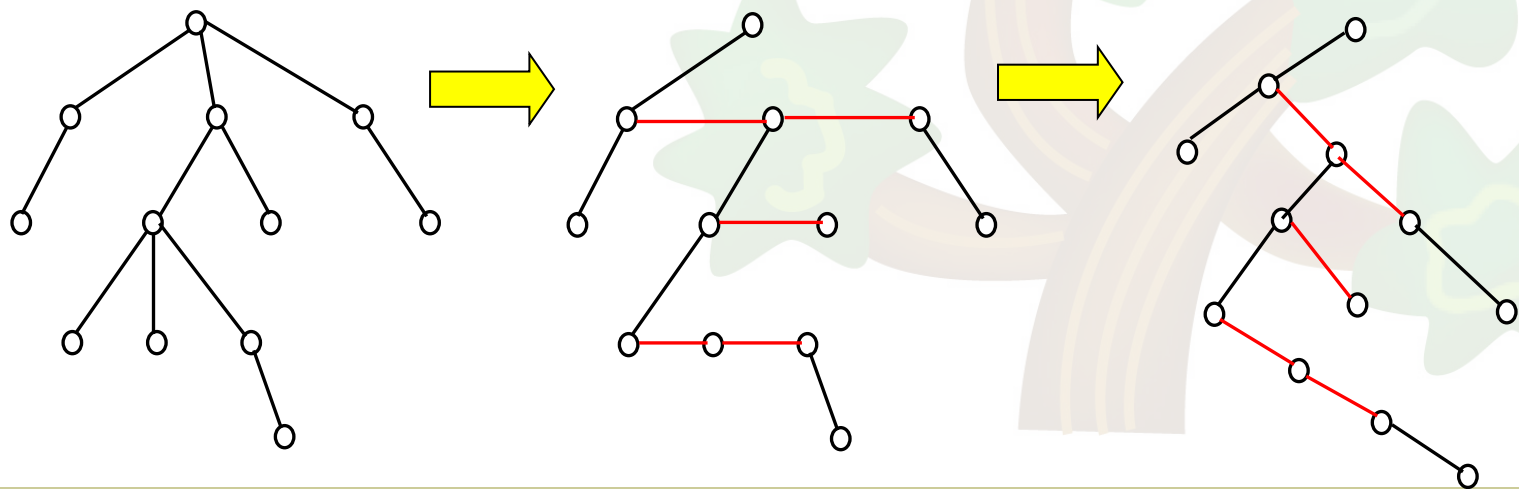




特殊的二叉树：有序二叉树



- **定义**（**有序树**）：树中同层结点从左至右有序排列，其序不可互换
- **事实**：任何有序树均可化为有序二叉树
- 易见，二叉搜索树就是一种有序二叉树

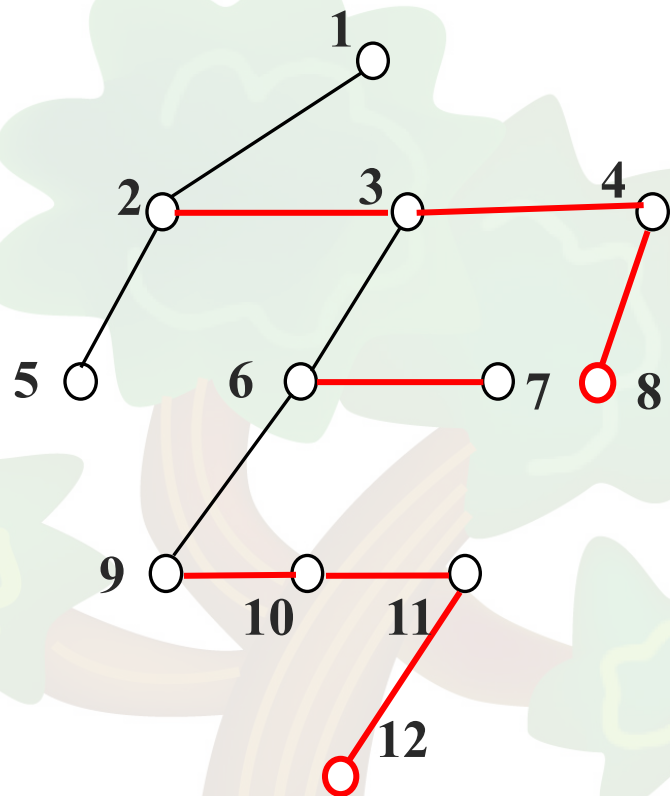
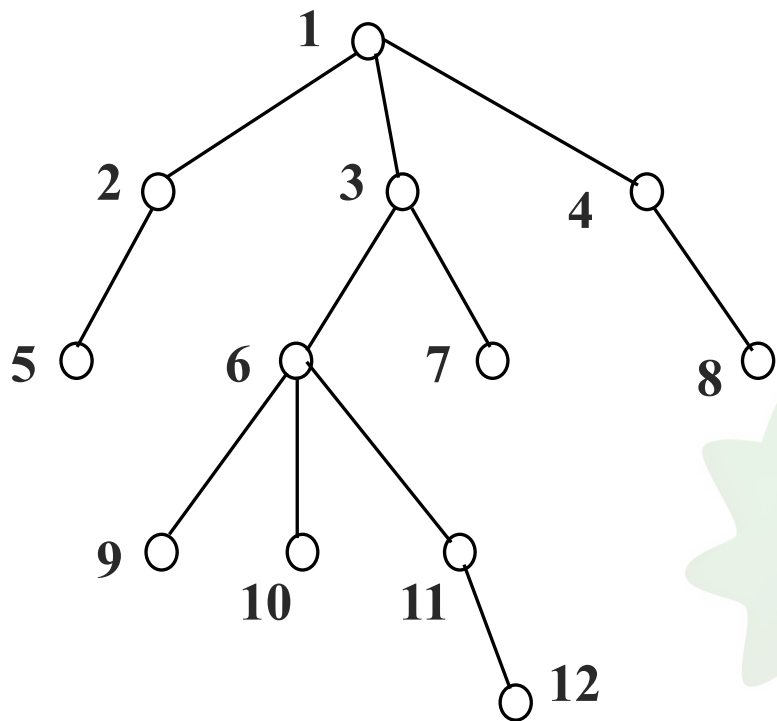




特殊的二叉树：有序二叉树(续)



来源: <http://nptel.com/series/261>

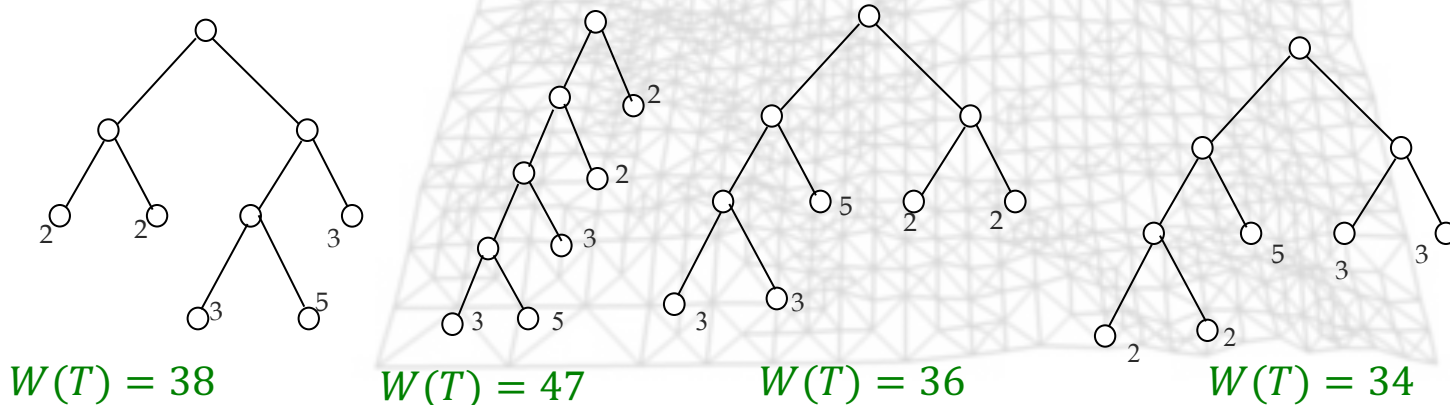




特殊的二叉树：最优二叉树



- 若 T 是二叉树，且每个叶 v_1, v_2, \dots, v_t 带有非负数值权 w_1, w_2, \dots, w_t ，则二叉树 T 的权 $W(T)$ 定义为： $W(T) = \sum_{i=1}^t w_i l(v_i)$ ，其中 $l(v_i)$ 表示 v_i 的层数（根为第0层）
- 具有相同权序列的二叉树中权最小的一个称为最优二叉树



易见：最优二叉树一定是二叉正则树（不能为链）



二叉树的应用：不等长编码



- 从信号流中识别（解码）字符的方式
 - 等长度编码 vs. 不等长度编码
- 例子：对包含 $\{a_{(45)}, b_{(13)}, c_{(12)}, d_{(16)}, e_{(9)}, f_{(5)}\}$ 6个字符的10万个字符的数据文件编码每个字符后面的数字表示该字符出现的频度(%)
 - 编码方案一： $a(000), b(001), c(010), d(011), e(100), f(101)$ ；则文件总长度30万字位
 - 编码方案二： $a(0), b(101), c(100), d(111), e(1101), f(1100)$ ；则文件总长度22.4万字位，空间节省约四分之一

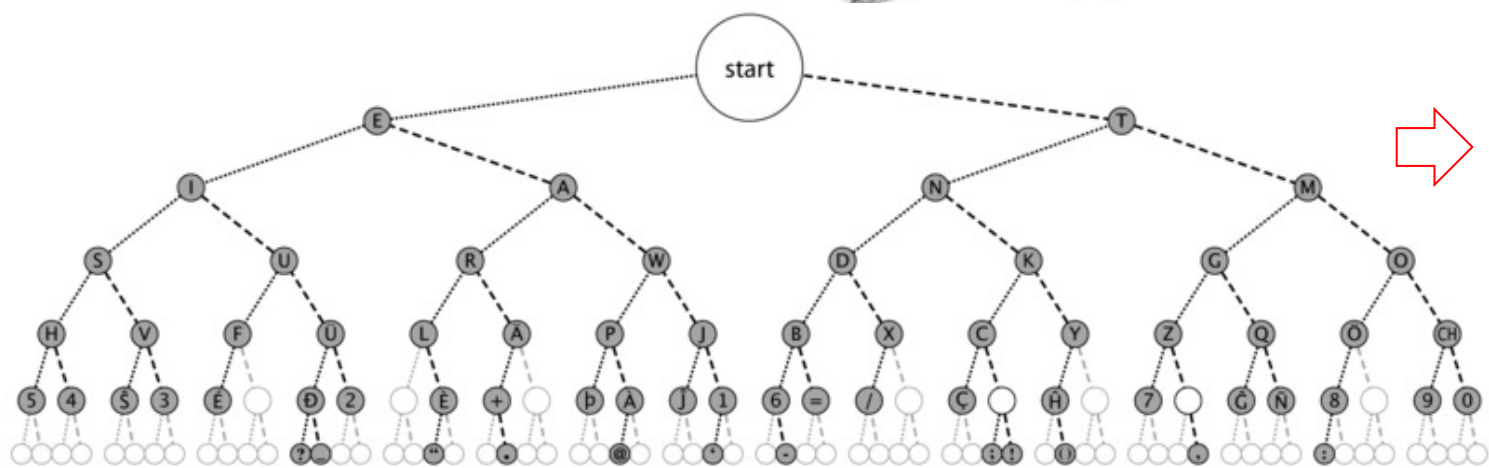


二叉树的应用：不等长编码（续）



A ● ■■■
B ■■■ ● ●
C ■■■ ● ■■ ●
D ■■■ ● ●
E ●
F ● ● ■■■
G ■■■ ■■■
H ● ● ● ●
I ● ●
J ● ■■■ ■■■ ■■■
K ■■■ ● ■■■
L ● ■■■ ● ●
M ■■■ ■■■
N ■■■ ●
O ■■■ ■■■ ■■■
P ● ■■■ ■■■ ●
Q ■■■ ■■■ ■■■
R ● ■■■ ●
S ● ● ●
T ■■■

U ● ● ■■■
V ● ● ■■■
W ● ■■■ ■■■
X ■■■ ● ● ■■■
Y ■■■ ● ■■■ ■■■
Z ■■■ ■■■ ● ●





不等长编码的分隔



- 从信号流中识别（解码）不等长编码表示的字符
 - 显式地表示长度：专用位或特定结束信号
 - 匹配的唯一性
- 如果符号 α 可以表示成符号串 β_1 和 β_2 的并置，则 β_1 称为 α 的一个前缀（注意： β_1 和 β_2 可以是空串）
- 设 $A = \{\beta_1, \beta_2, \dots, \beta_m\}$ 是符号串的集合，且对任意 $\beta_i, \beta_j \in A$ ，若 $i \neq j$ ， β_i, β_j 互不为前缀，则称 A 为前缀码
- 若 A 中的任意串 β_i 只含符号0, 1，则称 A 是二元前缀码



用二叉树生成二元前缀码

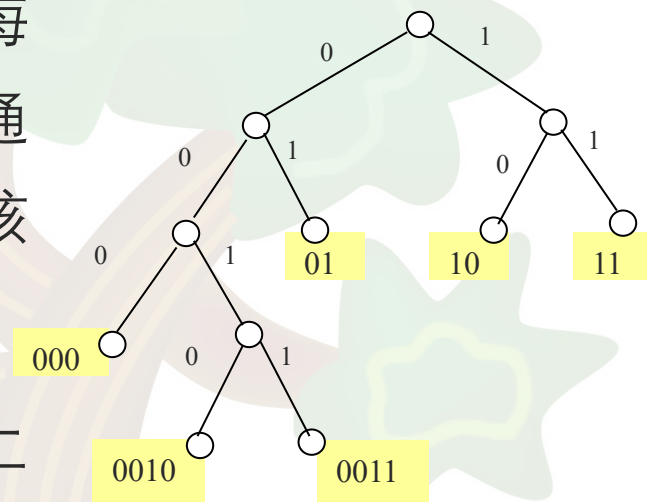


来源: <http://npu.edu.cn/> 课程: 第10讲

■ 生成方法:

- **给边标号**: 对任意非叶顶点, 对其出边标上号, 约定左为0, 右为1
- **给叶编号**: 根据根树性质, 从根到每个叶存在**唯一的有向通路**, 构成该通路的边的标号依次并置, 所得作为该叶的编号

- 给定一棵**二叉正则树**, 其叶节点上可以产生**唯一**的二元前缀码





最优二叉树与最佳前缀码



- **问题：** 二进前缀码 $A = \{\beta_1, \beta_2, \dots, \beta_m\}$ 表示 m 个不同的字母，如果各字母使用频率不同，如何设计编码方案可以使总传输量**最少**
- **基本思想：** 使用频率高的字母用尽量短的符号串表示
- **问题的解：** 若用频率(相对值)作为叶顶点的权，最佳二进前缀码对应的二叉树应该是**最优二叉树**



求最优二叉树的算法



■ 算法 (Huffman) :

- **输入:** 非负实数序列 w_1, w_2, \dots, w_t , 设 w_1, w_2 是其中最小的两实数
- **输出:** 具有 t 个树叶, 其权序列为 w_1, w_2, \dots, w_t 的最优二叉树
- **过程:**
 - 作 t 个叶顶点 v_1, v_2, \dots, v_t , 其相应的权分别是 w_1, w_2, \dots, w_t 。加入分支点 v_{t+1} , 使 v_1, v_2 是 v_{t+1} 的左、右子树。在上述权序列中用 $w'_{t+1} = w_1 + w_2$ 代替 w_1, w_2 两项
 - 考虑“**当前的**” w 序列, 选择其中最小的两项, 设其对应的顶点是 v_i, v_j , 加分支点 v_{t+k} , 使以 v_i, v_j 为根的子树为其左、右子树
 - 重复第2步, 直至加入 $t - 1$ 个顶点
- 本算法得到的结果**可能不唯一** (若权值最小顶点对的选择不唯一)



求最优二叉树的算法（续）



■ Huffman算法的递归描述*：

Huffman(C)

```
1   $n \leftarrow |C|$ 
2   $Q \leftarrow C$ 
3  for  $i \leftarrow 1$  to  $n - 1$ 
4      do allocate a new node  $z$ 
5           $left[z] \leftarrow x \leftarrow \text{EXTRACT-MIN}(Q)$ 
6           $right[z] \leftarrow y \leftarrow \text{EXTRACT-MIN}(Q)$ 
7           $f[z] \leftarrow f[x] + f[y]$ 
8           $\text{INSERT}(Q, z)$ 
9  return  $\text{EXTRACT-MIN}(Q)$  ▷ Return the root of the tree.
```



Huffman算法的例子

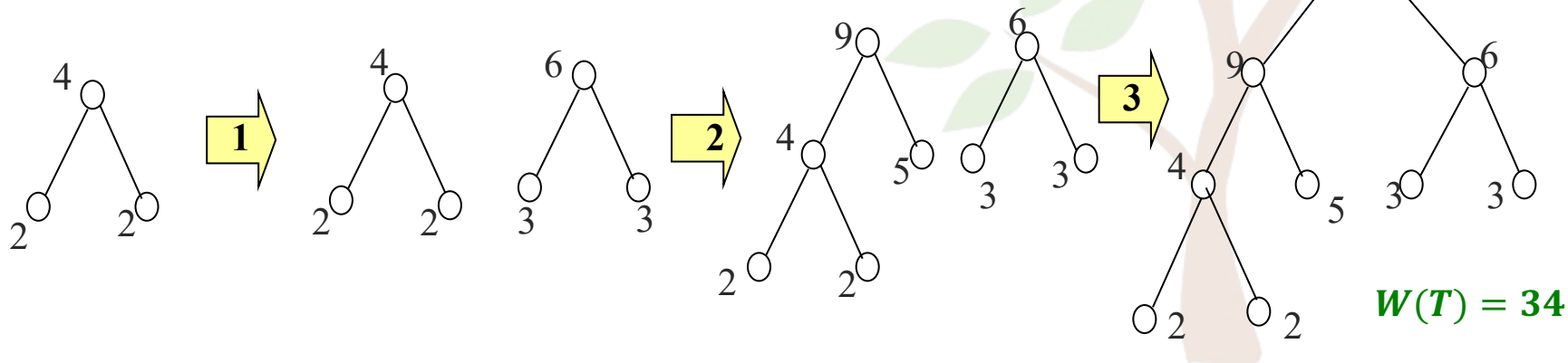


■ 例子：开始序列 **2, 2, 3, 3, 5**

○ 1步之后：4, **3, 3**, 5

○ 2步之后：**4, 6**, 5

○ 3步之后：**9, 6**





Huffman编码的一个例子



■ 八数字字符传输问题

○ 假设统计使用频率如下：

■ 0: 25% 1: 20%

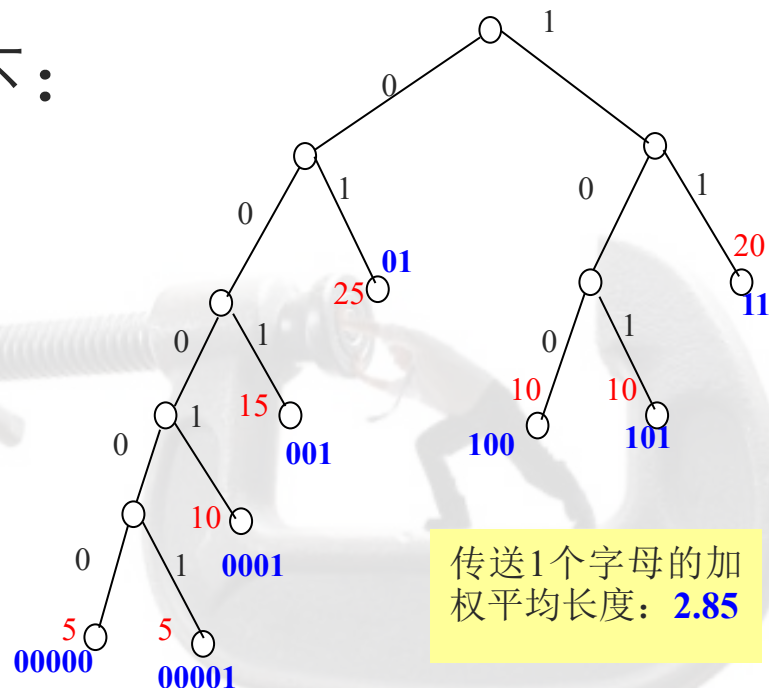
■ 2: 15% 3: 10%

■ 4: 10% 5: 10%

■ 6: 5% 7: 5%

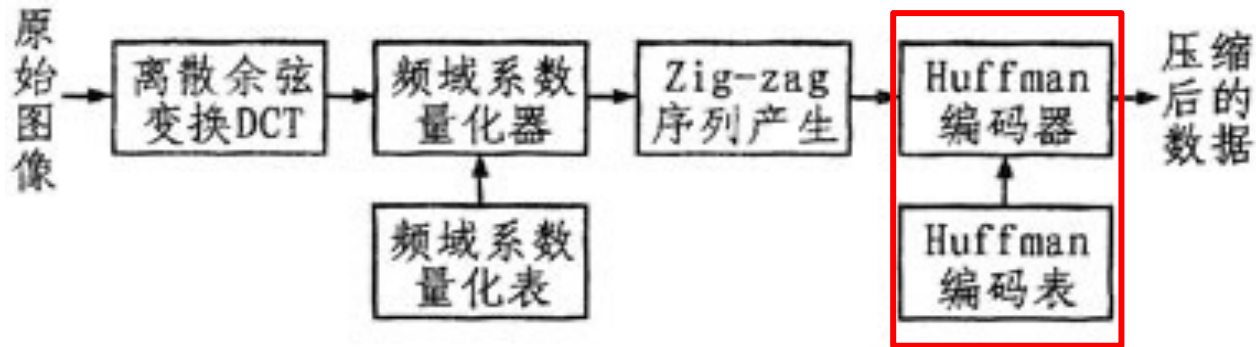
○ 则对应的权为：

■ 5,5,10,10,10,15,20,25

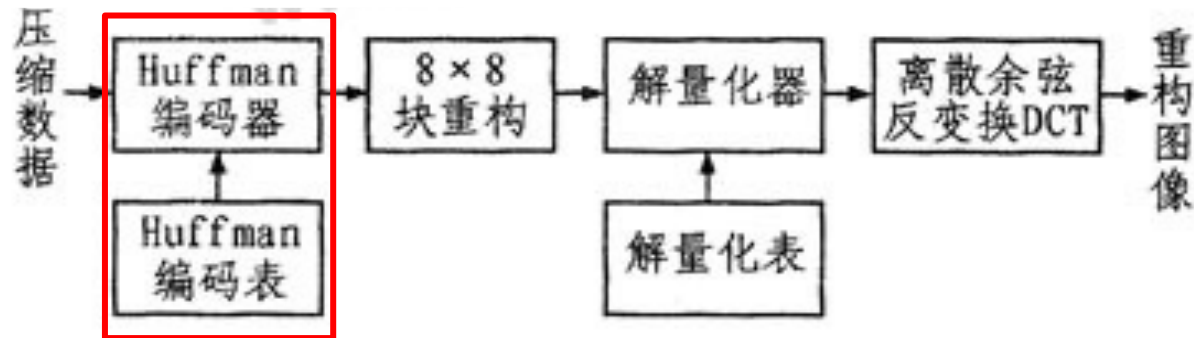




Huffman编码的一种应用*



JPEG（联合图像专家组）图像压缩与解压缩算法结构图





$(3-(2 \times x)) + ((x-2)-(3+x))$
as a doubly linked list

LEFT DATA RIGHT





根树的计算机表示* (续)



- // C++语言中二叉树结点类的定义

```
template<class T>
struct BTreeNode
{
    T data;
    BTreeNode <T> * Lchild, * Rchild;
    BTreeNode ( T nodeValue = T(), BTreeNode<T>* leftNode = NULL,
                BTreeNode <T>* rightNode = NULL )
        : data(nodeValue), Lchild(leftNode), Rchild(rightNode){ }
};
```



根树的遍历*



■ 三种遍历根树中所有顶点的递归算法

○ 中序遍历：依次为：

左子树，根，右子树

■ $h, d, i, b, e, a, f, c, g$

○ 前序遍历：依次为：

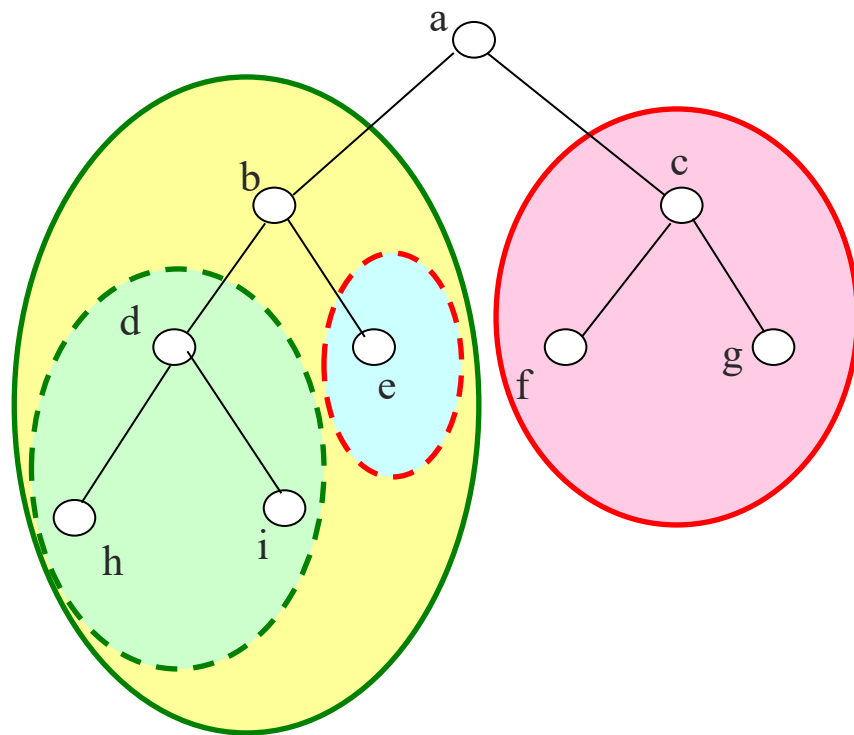
根，左子树，右子树

■ $a, b, d, h, i, e, c, f, g$

○ 后序遍历：依次为：

左子树，右子树，根

■ $h, i, d, e, b, f, g, c, a$





逆波兰表示法*



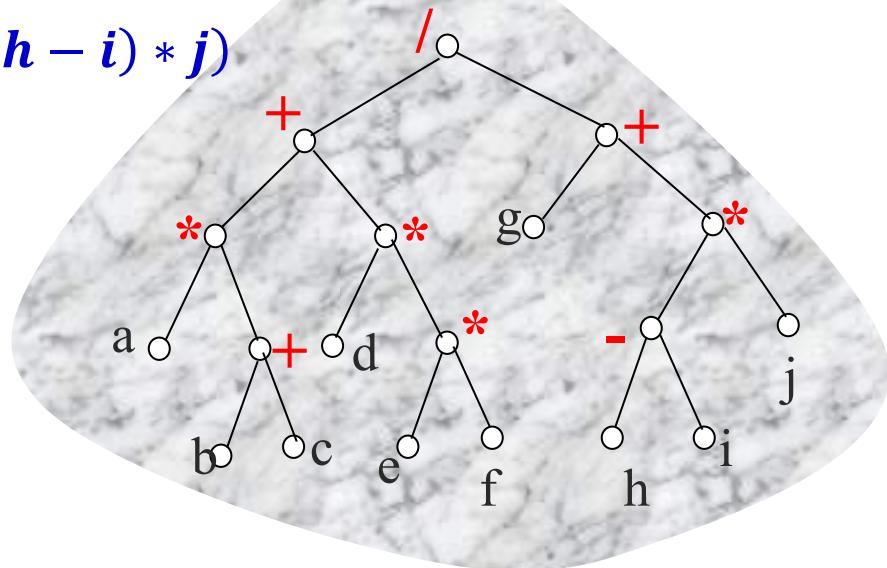
- 用根树表示表达式：分支点对应于运算符，叶顶点对应于运算变量（运算数），即“中序建树”，例如：

$$\left((a * (b + c) + d * (e * f)) \right) / (g + (h - i) * j)$$

- 采用后序遍历法列出该树的所有顶点，得：

abc +* def ** +ghi - j *+ /

- 这称为逆波兰表示法：无需括号便可唯一确定计算顺序

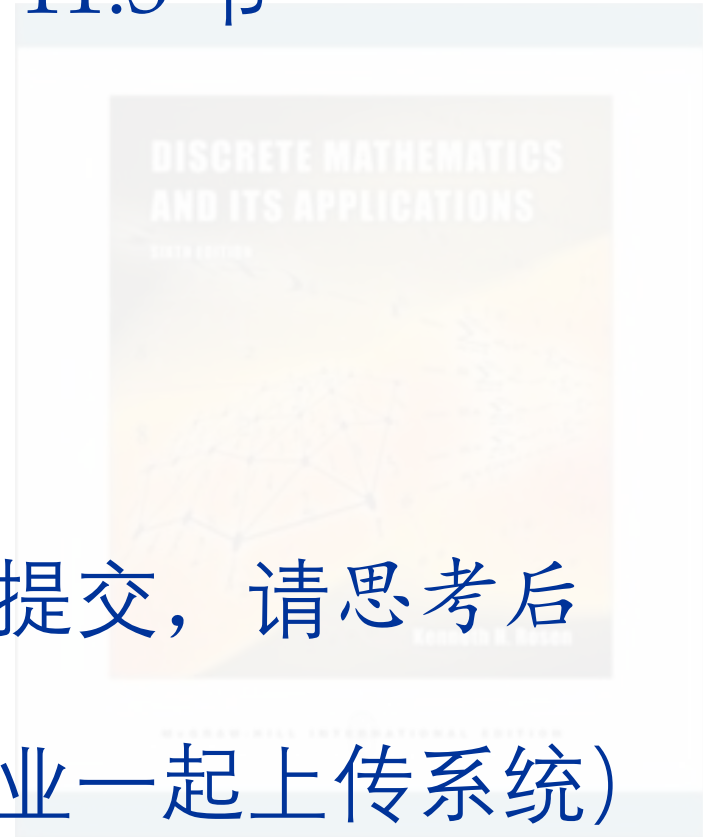




本次课后作业



- 教材内容：[Rosen] 11.1 – 11.3 节
- 课后习题：
 - Problem Set 28
- 提交时间：本次作业不再提交，请思考后
阅读参考解答（解答与作业一起上传系统）

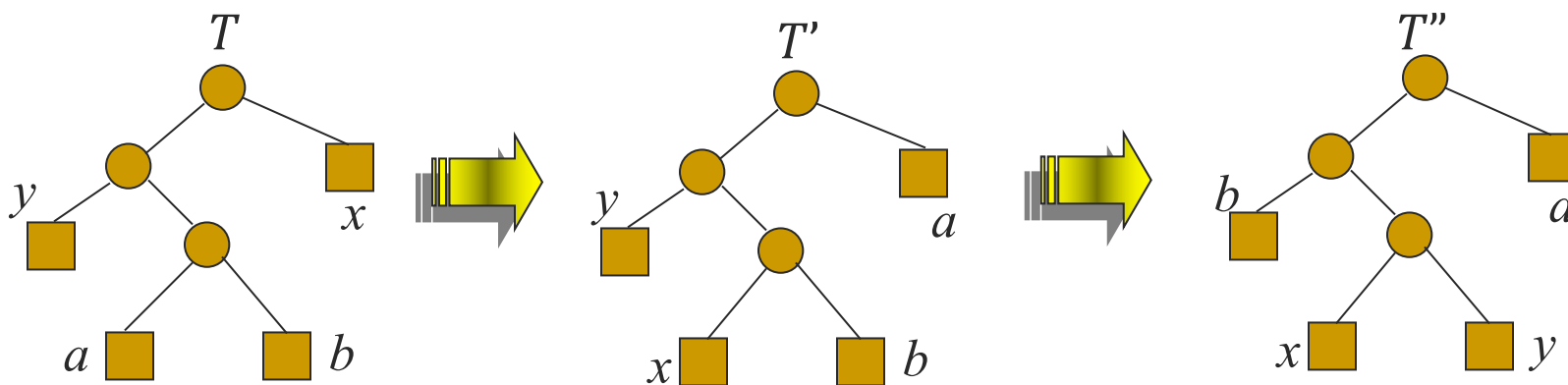




Huffman算法的正确性*



引理： 设 C 是字母表，其中每个字符 c 的出现频度为 $f[c]$ 。若 x, y 是两个频度最小的字符，则必存在 C 的一种最优前缀码，使得 x, y 的编码等长且仅有最后一位不同



T 为任意最优前缀码

在上图的变换中，二叉树的权保持不变
即： $B(T) = B(T') = B(T'')$



Huffman算法的正确性* (续)



不妨假设 $f[a] \leq f[b], f[x] \leq f[y]$; 于是 $f[x] \leq f[a], f[y] \leq f[b]$

$$\begin{aligned} B(T) - B(T') &= \sum_{c \in C} f(c)d_T(c) - \sum_{c \in C} f(c)d_{T'}(c) \\ &= f[x]d_T(x) + f[a]d_T(a) - f[x]d_{T'}(x) - f[a]d_{T'}(a) \\ &= f[x]d_T(x) + f[a]d_T(a) - f[x]d_T(a) - f[a]d_T(x) \\ &= (f[a] - f[x])(d_T(a) - d_T(x)) \geq 0 \\ \therefore B(T) &\geq B(T'); \text{同理, } B(T') \geq B(T''); \text{但 } B(T) \text{ 最小} \\ \therefore B(T) &= B(T') = B(T'') \end{aligned}$$



Huffman算法的正确性* (续)



定理 (最优子结构) : C 是字母表, 其中每个字符 c 的频率为 $f[c]$, x, y 是两个频率最小的字符。构造 $C' = C - \{x, y\} \cup \{z\}$, 并令 $f[z] = f[x] + f[y]$, 如果 T' 是表示 C' 的最优码的二叉树, 则将顶点 z 替换为分支点, 并以 x, y 为其子女, 所得二叉树 T 为 C 的一棵最优二叉树

对任意 $c \in C - \{x, y\}$, $d_T(c) = d_{T'}(c)$, so, $f[c]d_T(c) = f[c]d_{T'}(c)$

另一方面, $d_T(x) = d_T(y) = d_{T'}(z) + 1$,

因此, $f[x]d_T(x) + f[y]d_T(y) = (f[x] + f[y])(d_{T'}(z) + 1) =$
 $f[z]d_{T'}(z) + (f[x] + f[y])$

于是, $B(T) = B(T') + (f[x] + f[y])$

如果存在 T'' 满足 $B(T'') < B(T)$, 不失一般性, x 与 y 在 T'' 中为 siblings. 将 x, y 连同它们的父结点替换为一叶结点 z , 并令 $f[z] = f[x] + f[y]$, 设得到的新树为 T''' , 则:

$B(T''') = B(T'') - f[x] - f[y] < B(T) - f[x] - f[y] = B(T')$, 矛盾。