复习

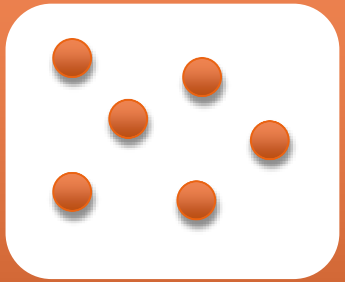
叶枫

考试题型：
一、填空（2%*10=20%）
二、选择（2%*10=20%）
三、简答（8%*6=48%）
四、算法填空（2%*3=6%）
五、算法设计（6%）

目录

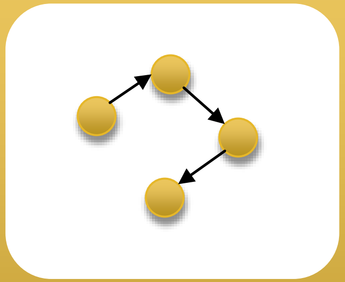
基本逻辑结构+算法时间复杂度分析

集合结构



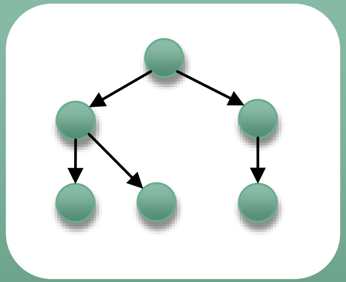
无关系
0前驱 and 0后继

线性结构



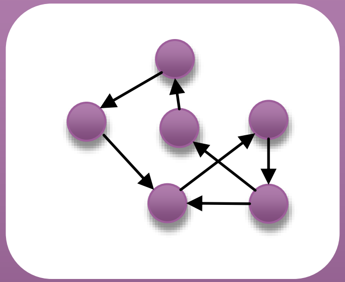
一对一关系
1前驱 or 1后继

树形结构



一对多关系
根：0前驱&n后继
非根：1前驱 and n后继
路径唯一

图形结构



多对多关系
n前驱 or n后继

基本逻辑结构+算法时间复杂度分析

1、基本概念和术语：数据、数据元素、数据项概念、三者关系

2、时间复杂度求解方法

目录

- 基础知识
- 线性表
- 堆栈和队列
- 数组和字符串
- 树
- 集合和搜索
- 搜索树
- 散列表
- 图
- 排序

顺序表

顺序表的插入、移动元素的个数

链表

单链表的插入算法、时间复杂度

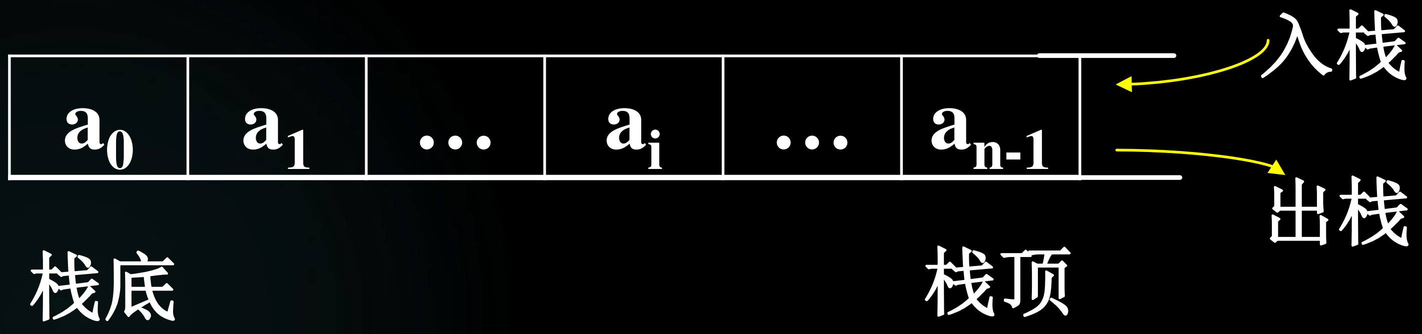
目录

- 基础知识
- 线性表
- 堆栈和队列
- 数组和字符串
- 树
- 集合和搜索
- 搜索树
- 散列表
- 图
- 排序

堆栈的基本概念

- 堆栈（Stack，栈）限定插入和删除操作都在同一端进行的线性表

$$S=(a_0,\ a_1,\ \cdots,\ a_{n-1})$$



后进先出(LIFO)的线性数据结构

- 后缀表达式求值算法：abc-/de*+

① 从左往右顺序扫描后缀表达式；

② 遇到操作数就进栈；

③ 遇到操作符就从栈中弹出两个操作数，并执行该操作符规定的运算；并将结果进栈；

④ 重复上述操作，直到表达式结束，弹出栈顶元素即为结果。

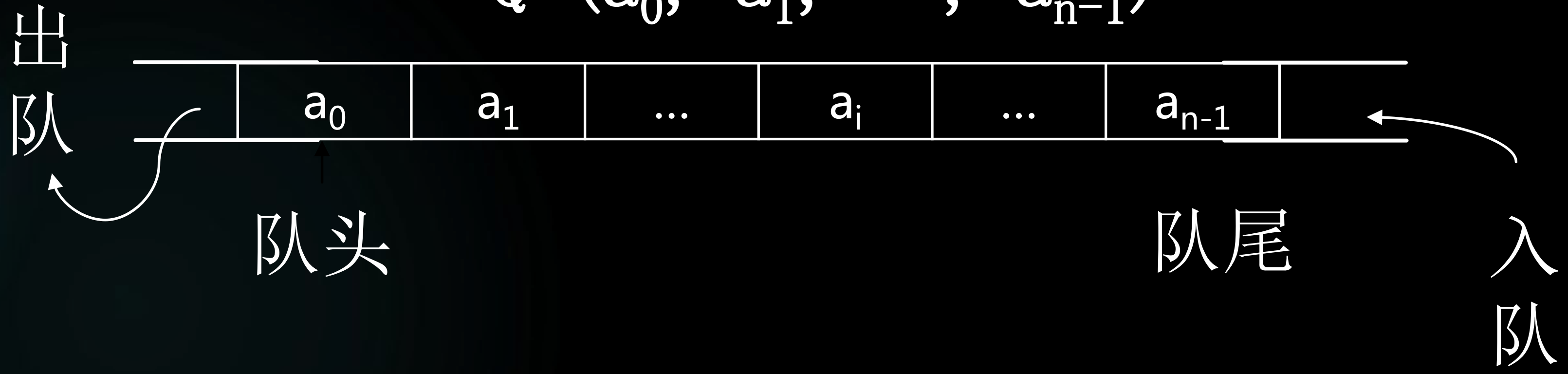
中缀表达式转换成后缀表达式

1. 从左到右扫描中缀表达式，遇到#转（2）；
   ① 遇到操作数直接输出；
   ② 遇到“）”，则连续出栈，直到“（”为止
   ③ 遇到其它操作符，与栈顶的操作符比较优先级；若优先级<=栈顶的优先级，则连续出栈，直到>栈顶，操作符进栈
2. 输出栈中剩余操作符

队列的基本概念

- 队列（Queue）是一种限定在表的一端插入，在表的另一端删除的线性表

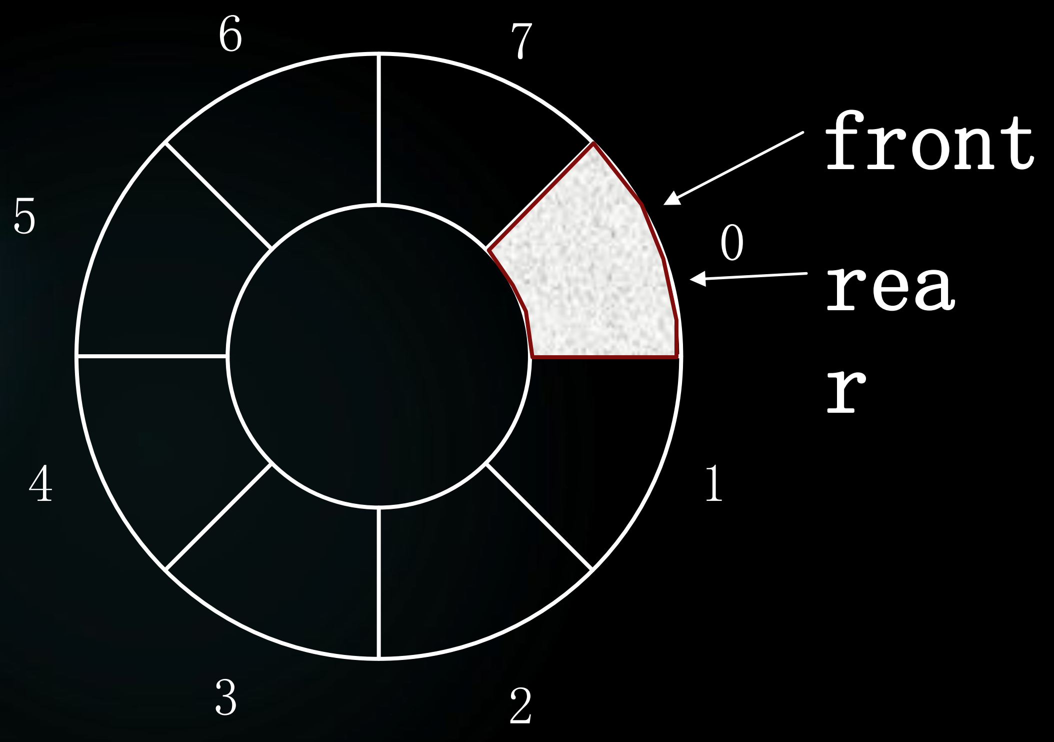
$$Q=(a_0, a_1, \cdots, a_{n-1})$$



先进先出(FIFO)的线性数据结构

队列的顺序存储表示

front指向的空间不可用



初始时：

front = rear = 0

指针前进

front = (front+1)%maxSize

rear = (rear+1)%maxSize

队列满的判断条件

(rear+1)%maxSize == front

队列空的判断条件

rear == front

循环队列插入元素算法

目录

- 基础知识
- 线性表
- 堆栈和队列
- 数组和字符串
- 树
- 集合和搜索
- 搜索树
- 散列表
- 图
- 排序

数组的顺序表示

- 二维数组的顺序表示

二维数组a[m][n]需按照行优先映射到一维的存储空间

$$\begin{bmatrix} a_{[0][0]} & a_{[0][1]} & \cdots & & \cdots & \cdots & a_{[0][n-1]} \\ a_{[1][0]} & a_{[1][1]} & \cdots & & \cdots & \cdots & a_{[1][n-1]} \\ \vdots & \vdots & \ddots & & & & \vdots \\ \vdots & \vdots & & a_{[i][j]} & & & \vdots \\ \vdots & \vdots & & & & \ddots & \vdots \\ a_{[n-1][0]} & a_{[n-1][1]} & \cdots & & \cdots & \cdots & a_{[n-1][n-1]} \end{bmatrix}$$

$$\underbrace{\boxed{\mathbf{a}_{[0][0]}\ \mathbf{a}_{[0][0]} \cdots\cdots \mathbf{a}_{[0][n-1]}}}_{\text{下标为0的行}}\underbrace{\boxed{\mathbf{a}_{[1][0]}\ \mathbf{a}_{[1][0]} \cdots\cdots \mathbf{a}_{[1][n-1]}}}_{\text{下标为1的行}}\underbrace{\boxed{\mathbf{a}_{[2][0]}\ \mathbf{a}_{[2][0]} \cdots\cdots \mathbf{a}_{[2][n-1]}}}_{\text{下标为2的行}}$$

数组的顺序表示

行优先顺序的地址计算

若对于二维数组a[m][n],已知每个数组元素占k个存储单元，第一个数组元素a[0][0]的存储地址是loc(a[0][0]),则数组元素a[i][j]的存储地址loc(a[i][j])为

$loc(a[i][j])=loc(a[0][0])+(i*n+j)*k \quad (0\leq i<m;0\leq j<n)$

$$\begin{bmatrix} a_{[0][0]} & a_{[0][1]} & \cdots & \cdots & \cdots & a_{[0][n-1]} \\ a_{[1][0]} & a_{[1][1]} & \cdots & \cdots & \cdots & a_{[1][n-1]} \\ \vdots & \vdots & \ddots & & & \vdots \\ \vdots & \vdots & & a_{[i][j]} & & \vdots \\ \vdots & \vdots & & & \ddots & \vdots \\ a_{[n-1][0]} & a_{[n-1][1]} & \cdots & \cdots & \cdots & a_{[n-1][n-1]} \end{bmatrix}$$

数组的顺序表示

列优先顺序的地址计算

若对于二维数组a[m][n],已知每个数组元素占k个存储单元，第一个数组元素a[0][0]的存储地址是loc(a[0][0]),则数组元素a[i][j]的存储地址loc(a[i][j])为

loc(a[i][j])=loc(a[0][0])+(j*m+i)*k (0≤i<m;0≤j<n)

已知两个元素的地址，判断是行优先还是列优先

$$\begin{bmatrix} a_{[0][0]} & a_{[0][1]} & \cdots & \cdots & \cdots & a_{[0][n-1]} \\ a_{[1][0]} & a_{[1][1]} & \cdots & \cdots & \cdots & a_{[1][n-1]} \\ \vdots & \vdots & \ddots & & & \vdots \\ \vdots & \vdots & & a_{[i][j]} & & \vdots \\ \vdots & \vdots & & & \ddots & \vdots \\ a_{[n-1][0]} & a_{[n-1][1]} & \cdots & \cdots & \cdots & a_{[n-1][n-1]} \end{bmatrix}$$

特殊矩阵之对称矩阵

- 以行优先顺序在数组B中存储下三角元素，则矩阵元素$a_{[i][j]}$在数组b中的存储位置k为：

$$
k=\begin{cases}\dfrac{i(i+1)}{2}+j & i\geq j\\[2mm] \dfrac{j(j+1)}{2}+i & j\geq i\end{cases}
$$

$$
\begin{pmatrix}
a_{[0][0]} & & & & \\
a_{[1][0]} & a_{[1][1]} & & & \\
a_{[2][0]} & a_{[2][1]} & a_{[2][2]} & & \\
a_{[3][0]} & a_{[3][1]} & a_{[3][2]} & a_{[3][3]} & \\
a_{[4][0]} & a_{[4][1]} & a_{[4][2]} & a_{[4][3]} & a_{[4][4]}
\end{pmatrix}
$$

$a_{[0][0]}$	$a_{[1][0]}$ $a_{[1][1]}$	$a_{[2][0]}$ $a_{[2][1]}$ $a_{[2][2]}$	$a_{[3][0]}$ $a_{[3][1]}$ $a_{[3][2]}$ $a_{[3][3]}$	$a_{[4][0]}$ $a_{[4][1]}$ $a_{[4][2]}$ $a_{[4][3]}$ $a_{[4][4]}$
1	2	3	4	5

特殊矩阵之上下三角矩阵

- 主对角以下元素全为0的方阵称为上三角矩阵。
- 主对角以上元素全为0的方阵称为下三角矩阵。
- 上、下三角矩阵采用对称矩阵的存储方式，只存储主对角线及其以上（或以下）的元素。

$$k=\begin{cases}\dfrac{i(i+1)}{2}+j & \text{下三角矩阵}\\[2ex] \dfrac{j(j+1)}{2}+i & \text{上三角矩阵}\end{cases}$$

$$\begin{pmatrix} a_{[0][0]} & 0 & 0 & 0 & 0\\ a_{[1][0]} & a_{[1][1]} & 0 & 0 & 0\\ a_{[2][0]} & a_{[2][1]} & a_{[2][2]} & 0 & 0\\ a_{[3][0]} & a_{[3][1]} & a_{[3][2]} & a_{[3][3]} & 0\\ a_{[4][0]} & a_{[4][1]} & a_{[4][2]} & a_{[4][3]} & a_{[4][4]} \end{pmatrix}$$

稀疏矩阵的顺序表示

$$
\begin{array}{c|cccccc}
 & 0 & 1 & 2 & 3 & 4 & 5 \\
\hline
0 & 16 & 0 & 0 & 22 & 0 & -16 \\
1 & 0 & 12 & 3 & 0 & 0 & 0 \\
2 & 0 & 0 & 0 & -8 & 0 & 0 \\
3 & 0 & 0 & 0 & 0 & 0 & 0 \\
4 & 91 & 0 & 0 & 0 & 0 & 0 \\
5 & 0 & 0 & 0 & 0 & 0 & 0 \\
6 & 0 & 0 & 15 & 0 & 0 & 0
\end{array}
$$

稀疏矩阵*M*

	row	*col*	*value*
0	0	0	16
1	0	3	22
2	0	5	–16
3	1	1	12
4	1	2	3
5	2	3	–8
6	4	0	91
7	6	2	15

行三元组

稀疏矩阵转置过程
会求稀疏矩阵转置过程中的k和Num数组

目录

- 基础知识
- 线性表
- 堆栈和队列
- 数组和字符串
- 树
- 集合和搜索
- 搜索树
- 散列表
- 图
- 排序

树的术语

结点的度(degree)：结点拥有的子树数

结点E的度为3，结点F的度为2，
结点A的度为1，结点G的度为0。

叶子(leaf)：度为零的结点

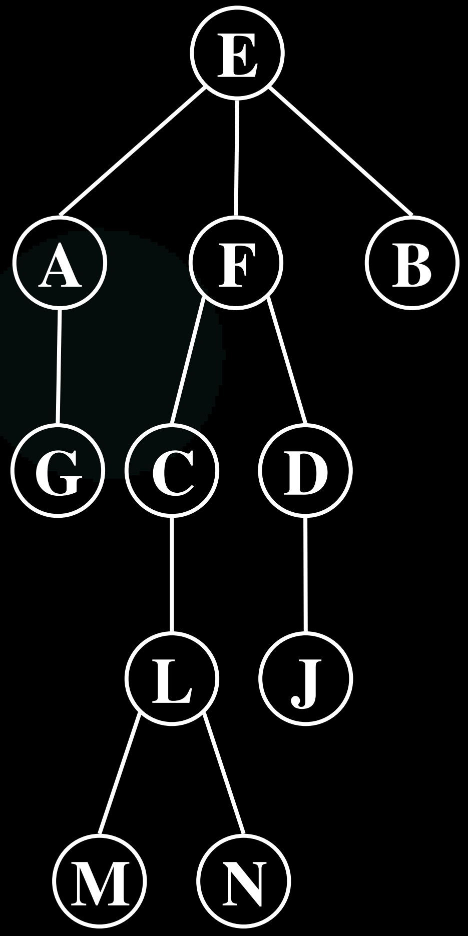
B、G、J、M、N均为叶子结点

树的度：树中结点的最大的度

该树的度为3

分支结点(branch)：度不为零的结点。

E、A、F、C等为分支结点



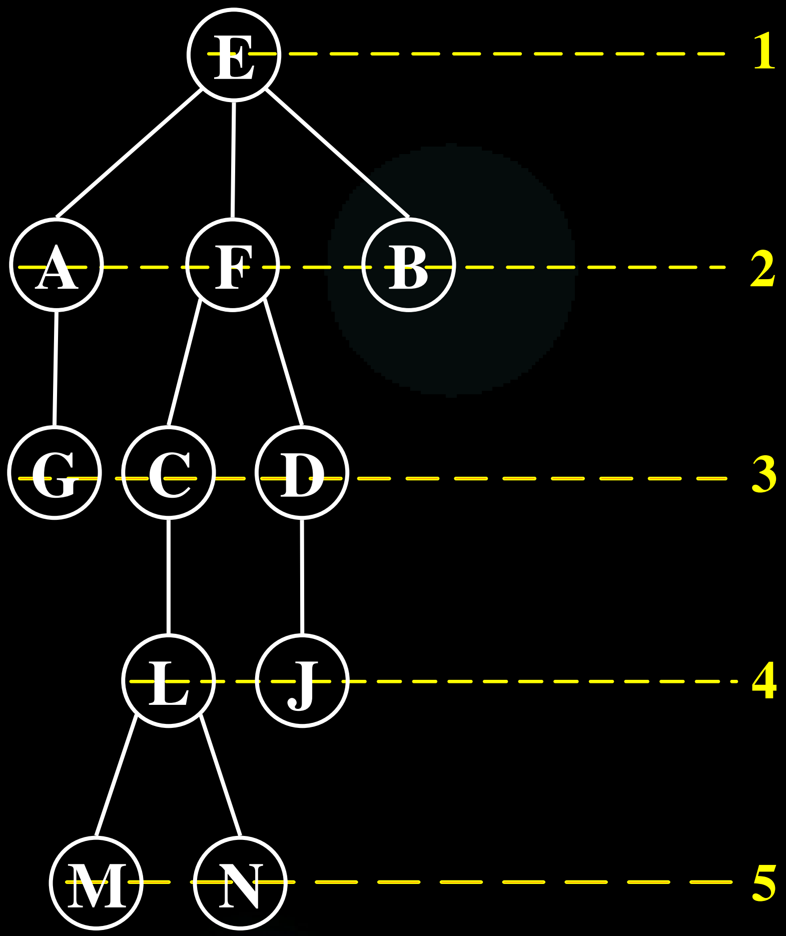
树的定义

结点的层次：根结点的层次为1，其余结点的层次等于其双亲结点的层次加1

结点E的层次为1
结点M的层次为5

树的高度：树中结点的最大层次

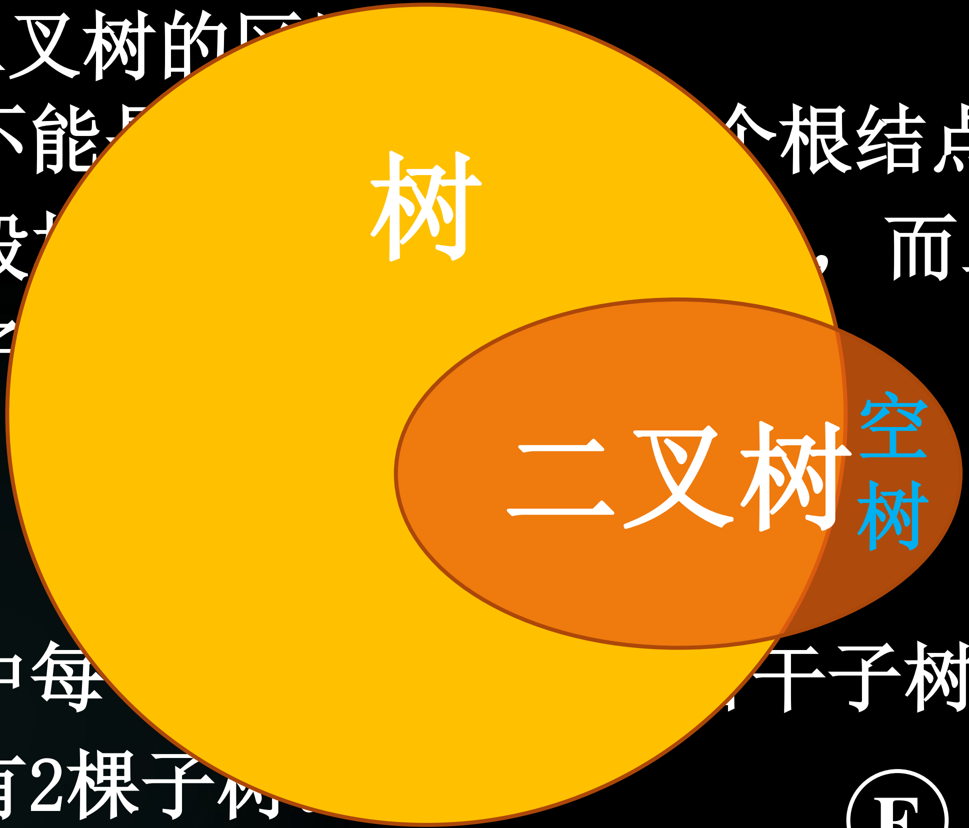
树的高度为5



二叉树的定义

树与二叉树的区

- 树不能是 个根结点。而二叉树可以是空树
- 一般 ，而二叉树中结点的子树要区分左、右子



- 树中每 干子树。而二叉树的每个节点最多只能有2棵子树

二叉树的定义

性质1：二叉树的第i(i≥1)层上至多有2^{i-1} 个结点（归纳法证明）

- 当i=1时，二叉树至多只有一个结点，结论成立。
- 设当i=k时结论成立，即二叉树上至多有2^k-1 个结点

- 当i=k+1时

∵每个结点最多只有两个孩子，
∴第k+1层上至多有$2*2^{k-1}=2^k$个结点，性质成立

二叉树的定义

性质2 高度为h的二叉树上至多有2^h-1个结点。

当h=0时，二叉树为空二叉树。

当h>0时，利用<u>性质1</u>，高度为h的二叉树中结点的总数最多为：

性质1 二叉树的第$i(i\geq 1)$层上至多有2^i-1个结点。

$$\sum_{i=1}^{h} 2^{i-1} = (2^0 + 2^1 + 2^2 + \ldots + 2^{h-1}) = 2^h - 1$$

补充：

等比数列的求和公式是

$$1 + a + a^2 + a^3 + \ldots + a^n = \frac{1-a^{n+1}}{1-a}$$

性质3 包含n个元素的二叉树的高度至少为$\lceil \log_2(n+1) \rceil$

根据性质2，高度为h的二叉树最多有2^h-1个结点，因而$n \le 2^h-1$，则有$h \ge \log_2(n+1)$。

由于h是整数，所以$h \ge \lceil \log_2(n+1) \rceil$。

性质4 任意一棵二叉树中，若叶结点的个数（度为0）为n_0，度为2的结点的个数为n_2，则必有$n_0=n_2+1$

无后的结点个数等于儿女双全的结点数+1

设二叉树的度为1的结点数为n_1，树中结点总数为n，则$n=n_0+n_1+n_2$ ①

（∵二叉树中只有度为0、1、2三种类型的结点）

设分支数为B（树枝），n个结点的二叉树，除了根结点外，每个结点都有一个分支进入，则B=n-1；分支是由度为1或者度为2的射出的，又有$B=2n_2+n_1$；

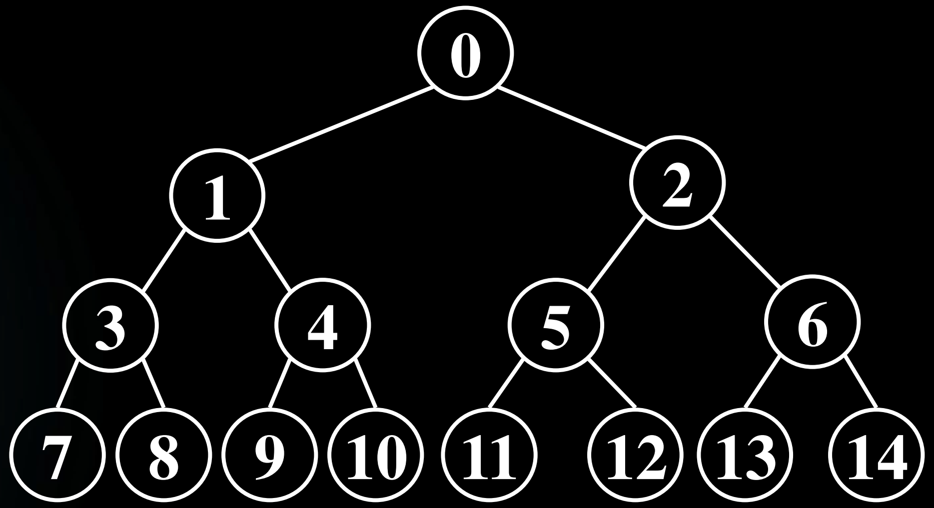
则有：$n-1=2n_2+n_1$ ➔ $n=2n_2+n_1+1$……②

由① ②可得到：

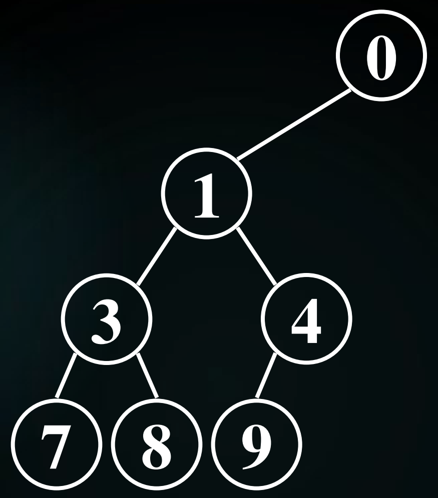
$n_0+n_1+n_2=2n_2+n_1+1$ ➔ $n_0+n_2=2n_2+1$ 即$n_2=n_0-1$。

定义 高度为h的二叉树恰好有2^h-1个结点时称为满二叉树

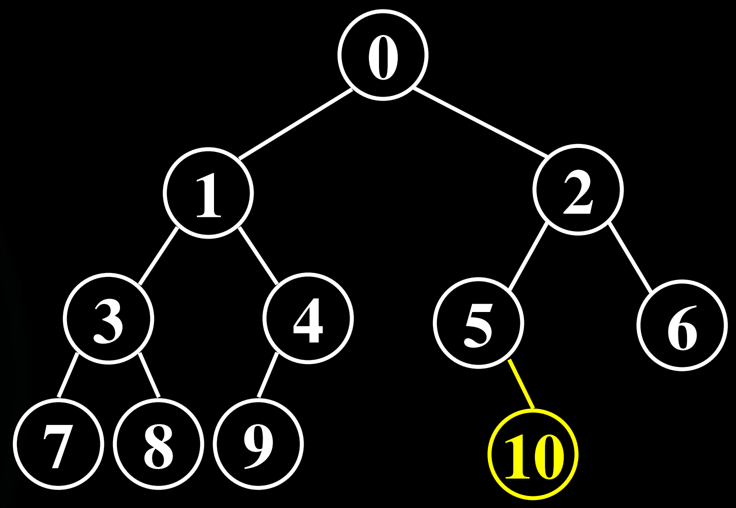
性质2 高度为h的二叉树上至多有2^h-1个结点。



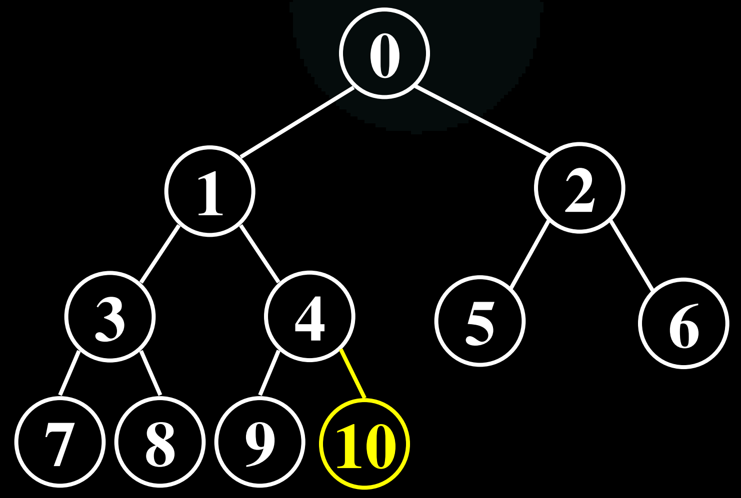
定义 一棵二叉树中，只有**最下面两层结点**的度可以小于2，并且最下一层的**叶结点**集中在**靠左**的若干位置上。这样的二叉树称为**完全二叉树**



非完全二叉树



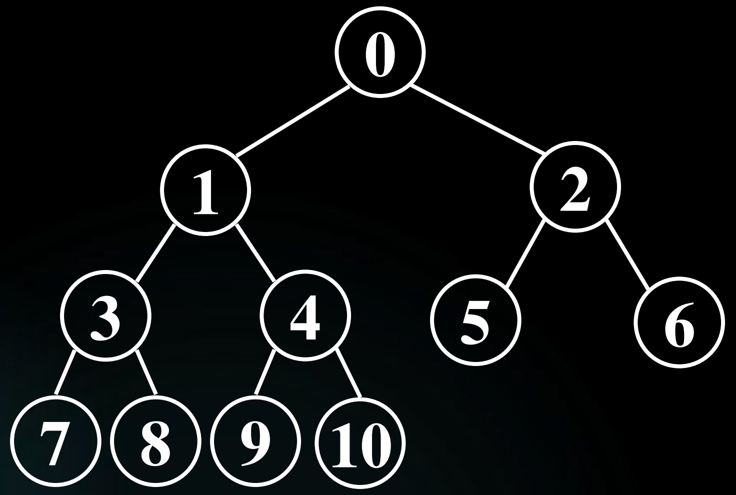
非完全二叉树



完全二叉树

完全二叉树的性质

性质5 具有n个结点的完全二叉树的高度为$\lceil \log_2(n+1) \rceil$



根据完全二叉树定义：除最后两层外，其他层结点都是满度（2）

↓

删除最后一层，得到的是一个满树

高度为h的二叉树恰好有2^h-1个结点时称为满二叉树

↓

删除最后一层，剩下结点树是$2^{h-1}-1$

二叉树的第i(i≥1)层上至多有2^{i-1}个结点

↓

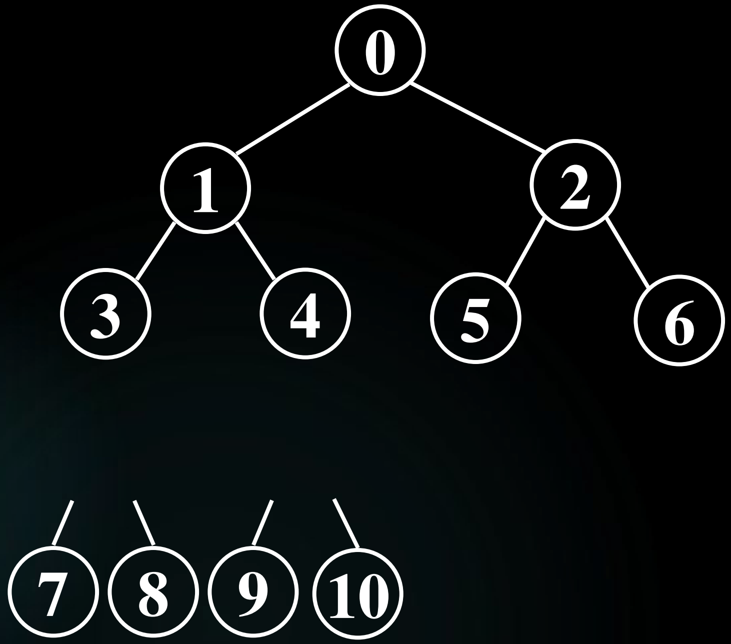
h层结点数$[1, 2^{h-1}]$

↓

完全二叉树结点个数 $2^{h-1} \leq n \leq 2^h-1$

完全二叉树的性质

性质5 具有n个结点的完全二叉树的高度为$\lceil \log_2 (n+1) \rceil$



完全二叉树结点个数 $2^{h-1} \le n \le 2^h - 1$

$$\log_2(n+1) \le h \le \log_2 n+1$$

h是整数

$$\lceil \log_2(n+1) \rceil \le h \le \lfloor \log_2 n + 1 \rfloor$$

$$h = \lceil \log_2(n+1) \rceil = \lfloor \log_2 n + 1 \rfloor$$

已知完全二叉树高度，求解树叶

n个结点的二叉树中完全二叉树最矮

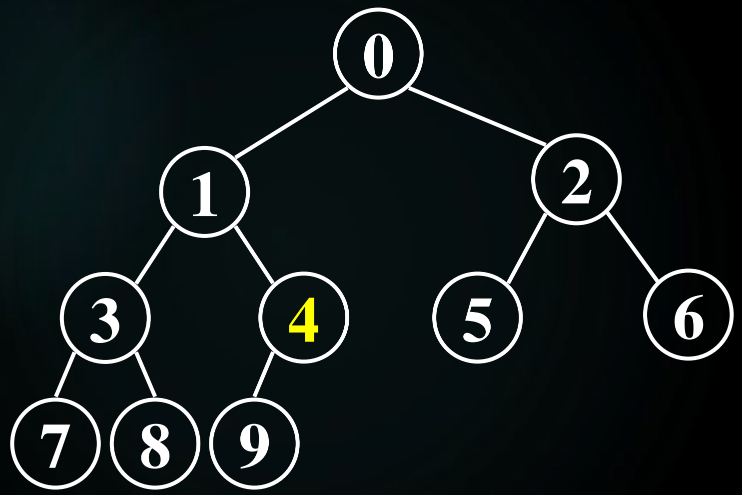
性质6 假定对一棵有n个结点的完全二叉树中的结点，按从上到下、从左到右的顺序，从0到n-1编号，设结点序号为i，则有以下关系成立：

（1）当i=0时，该结点为二叉树的根。

（2）若i>0，则该结点的双亲的序号为$\lfloor (i-1)/2 \rfloor$

（3）若2i+1＜n，则该结点左孩子的序号为2i+1，否则该结点无左孩子

（4）若2i+2＜n，则该结点右孩子的序号为2i+2，否则该结点无右孩子



设结点i的层号是k，k层结点编号范围

$$2^{k-1}-1 \le i \le 2^k-2$$

设结点i的孩子编号取决于

- 与i同层，在i之后结点个数$af(i)=2^k-2-\mathrm{i}$
- 与i同层，在i之前结点的孩子个数

$$bc(i)=2\times(\mathrm{i}-2^{k-1}+1)$$

i的左孩子编号=$i+af(i)+bc(i)+1=2i+1$

(1) 先序遍历（VLR）

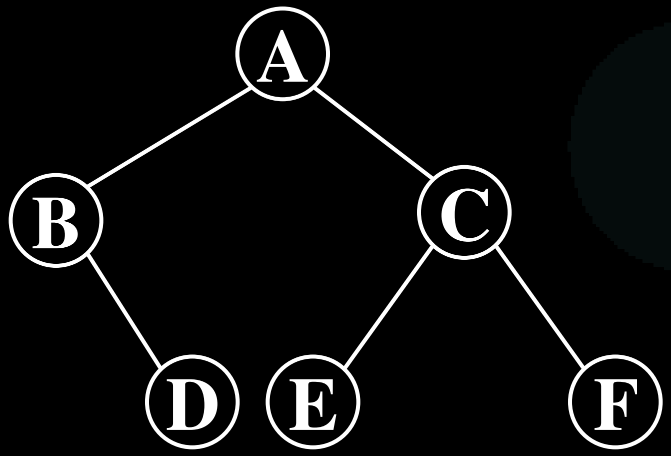
IF二叉树为空，则什么也不做

ELSE

访问根结点；

先序遍历（左子树）；

先序遍历（右子树）。



先序遍历序列： A B D C E F

(2) 中序遍历（LVR）

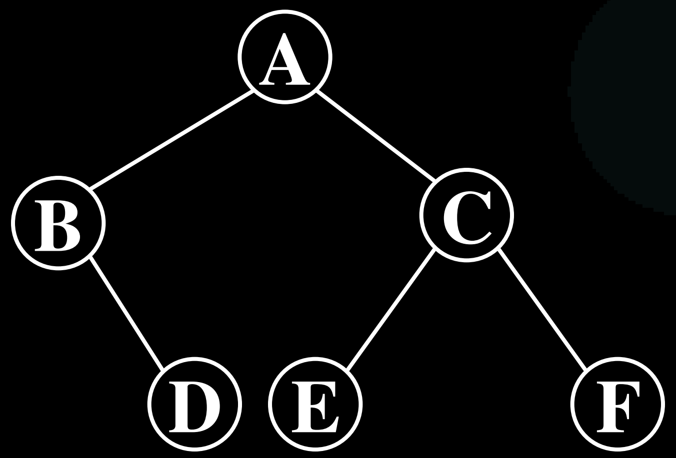
IF二叉树为空，则什么也不做；

ELSE

中序遍历（左子树）；

访问根结点；

中序遍历（右子树）。



中序遍历序列：B D A E C F

(3) 后序遍历 （LRV）

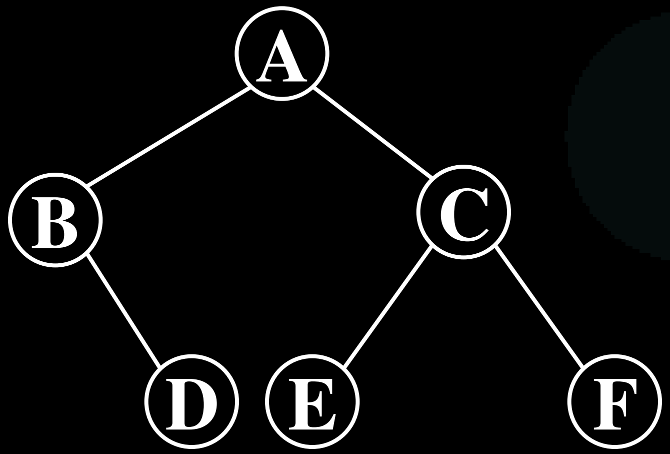
IF二叉树为空，则什么也不做；

ELSE

后序遍历（左子树）；

后序遍历（右子树）；

访问根结点。

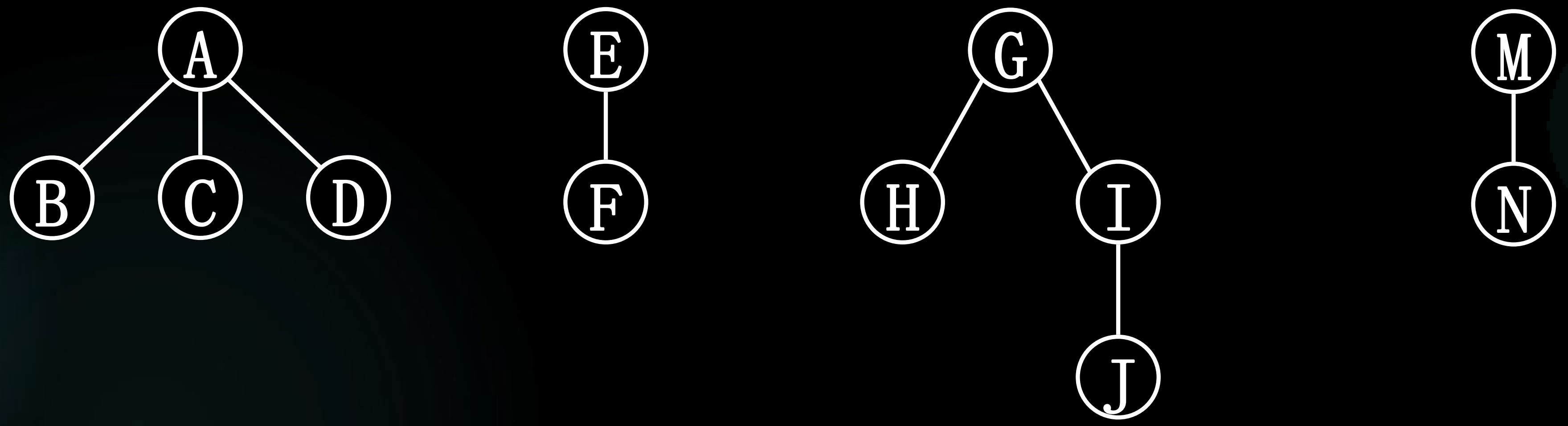


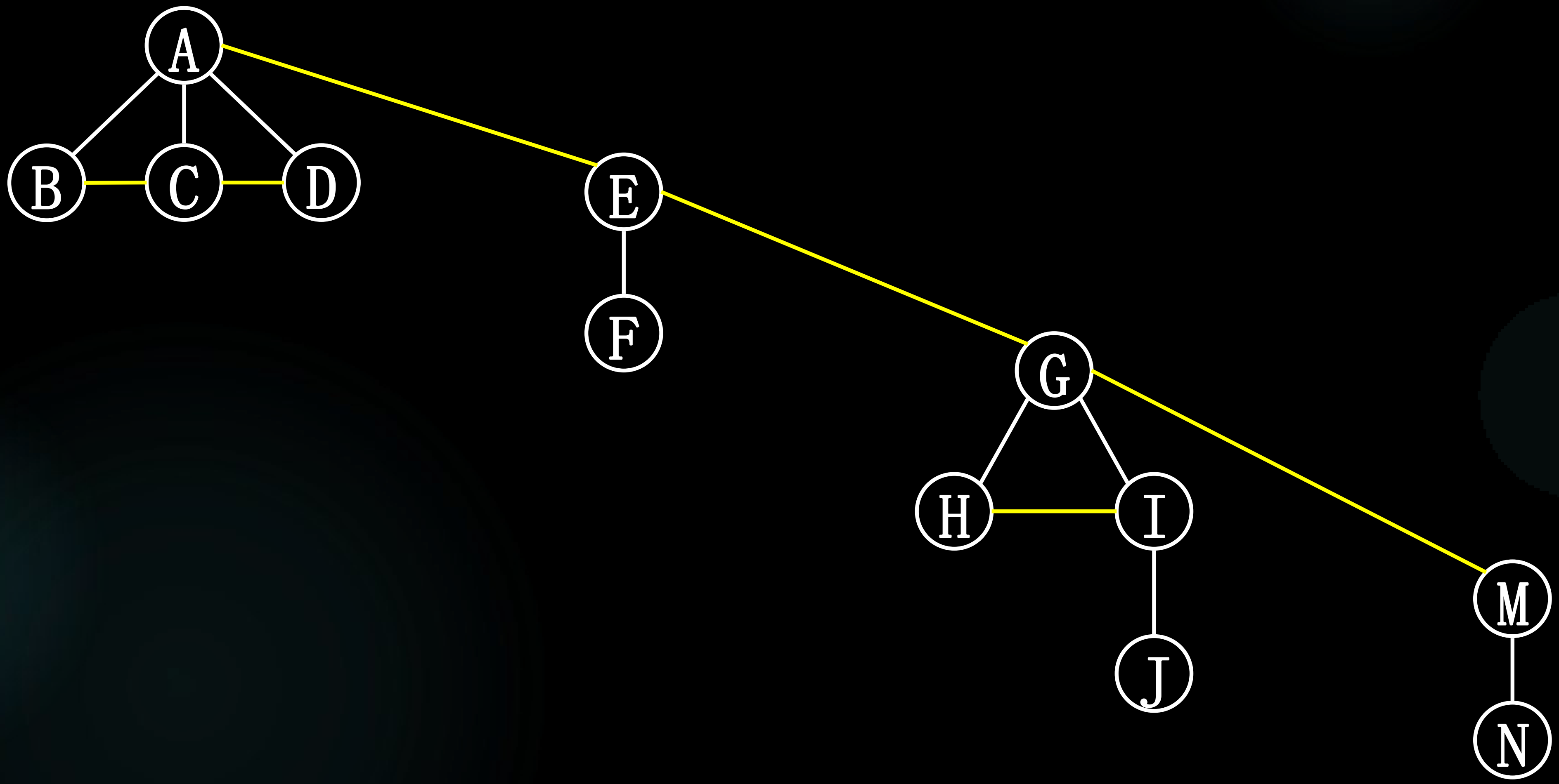
掌握遍历的原理、根据三种遍历序列的部分显示，画出二叉树。遍历算法的应用例题，课件补充应用算法

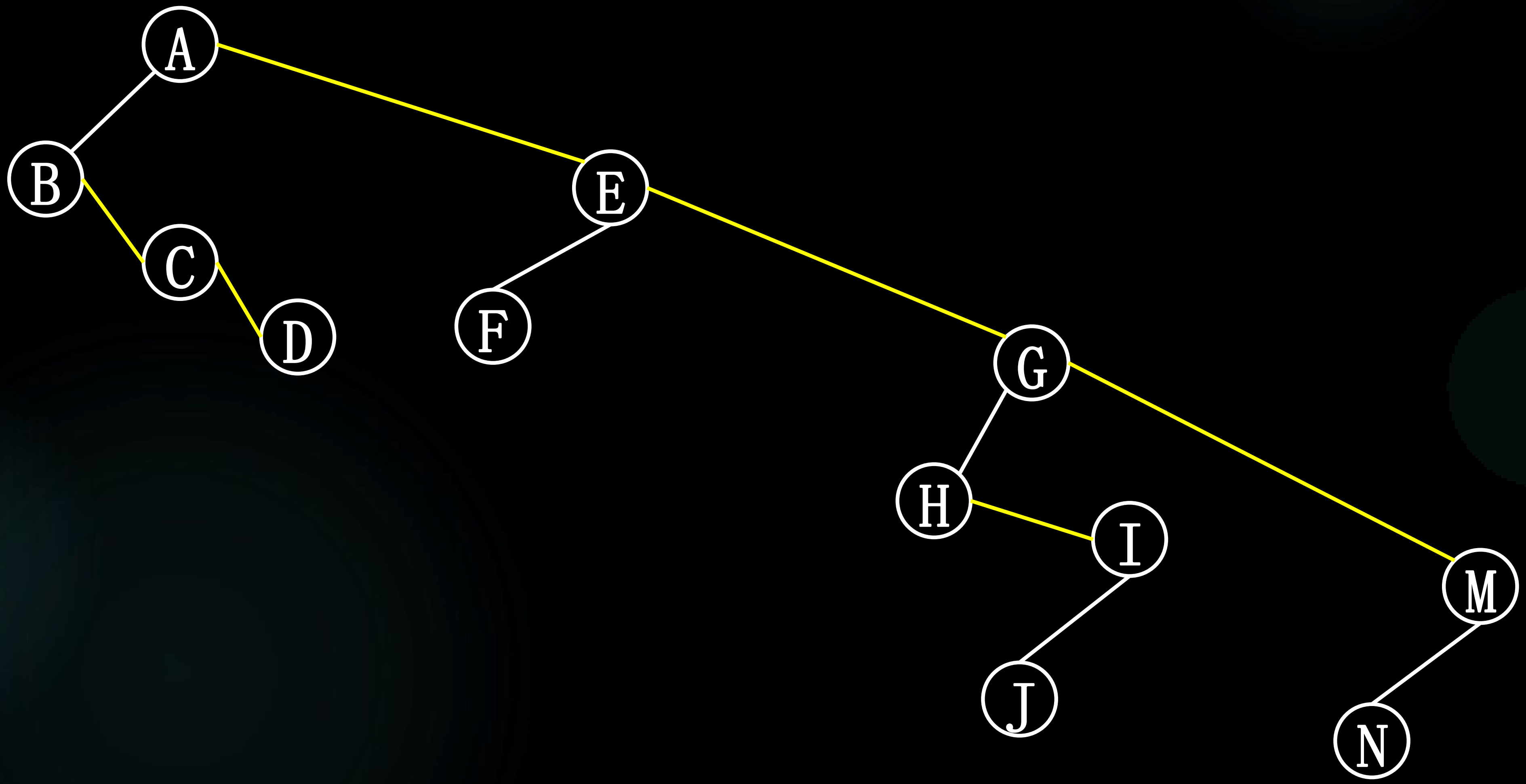
后序遍历序列：**D B E F C A**

二叉树的线索化，线索数与分支数的关系

森林转换成二叉树的原理

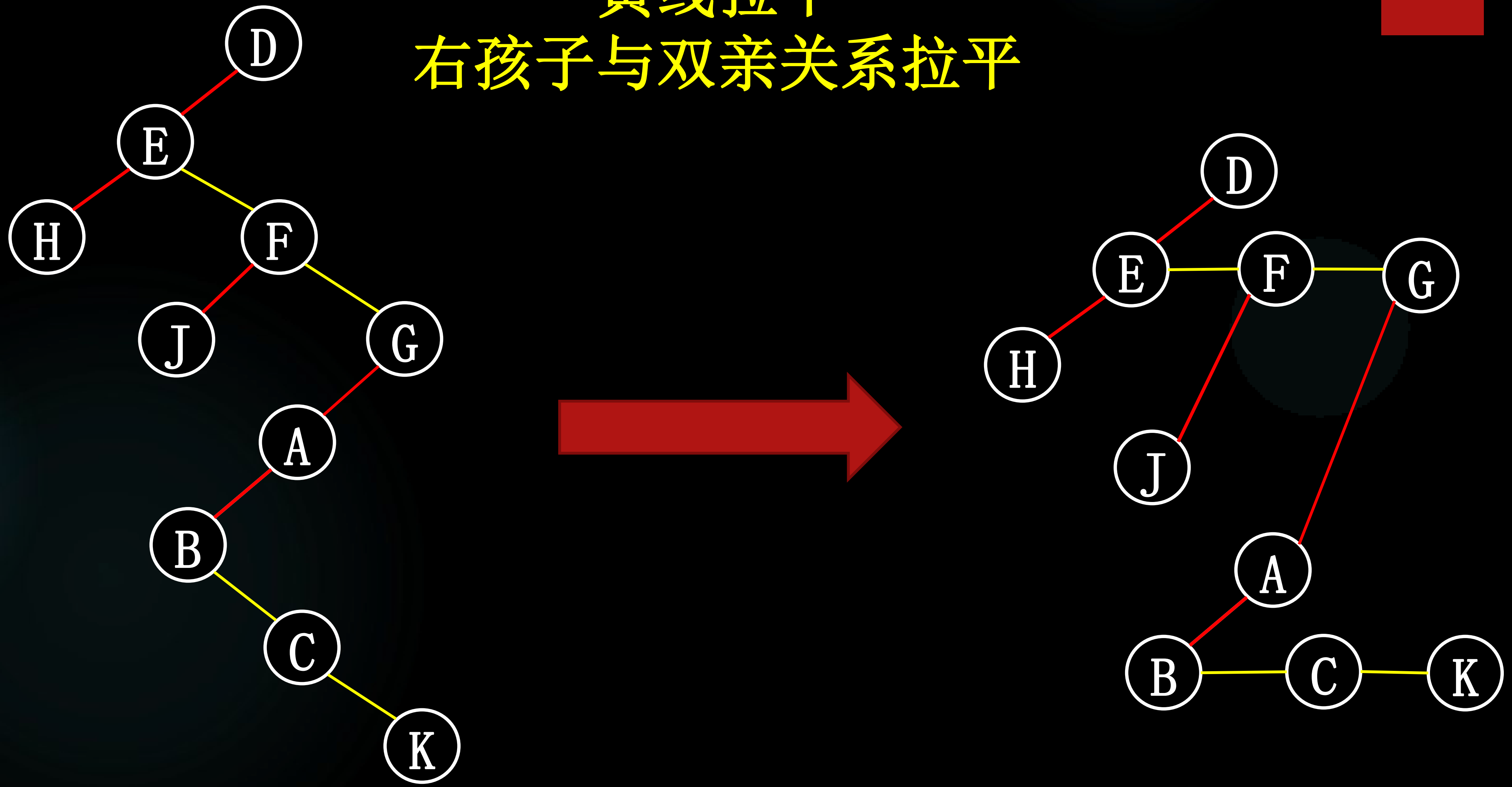

A
B
C
D
E
F
G
H
I
J
M
N


A
B
C
D
E
F
G
H
I
J
M
N

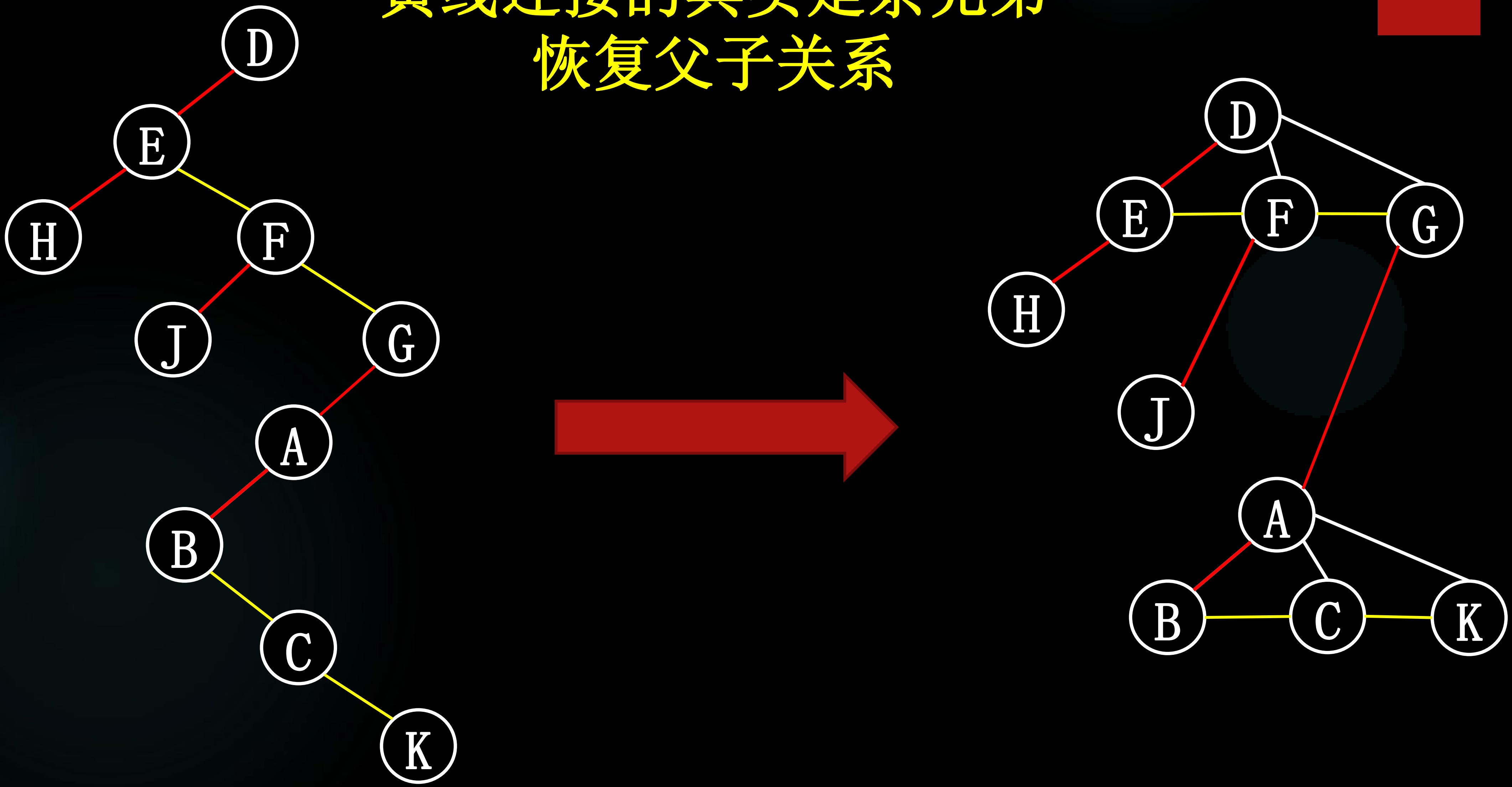

A
B
C
D
E
F
G
H
I
J
M
N

二叉树转换成森林

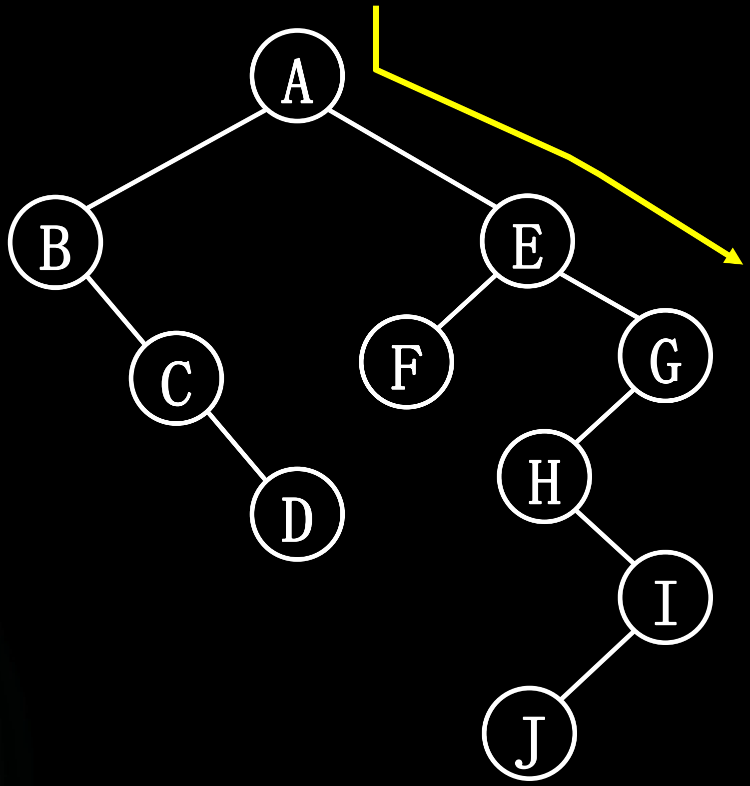
黄线拉平
右孩子与双亲关系拉平



黄线连接的其实是亲兄弟
恢复父子关系

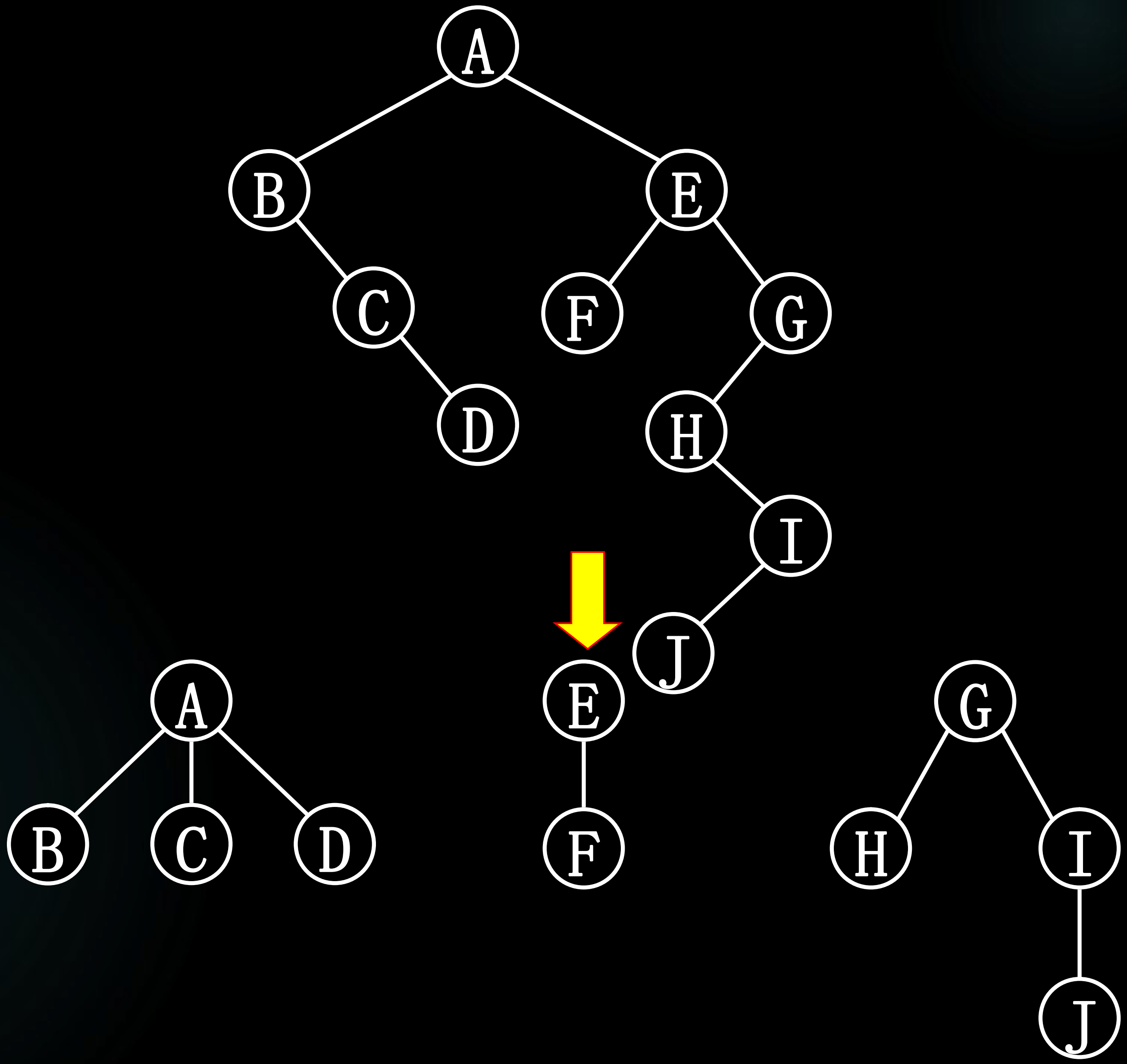


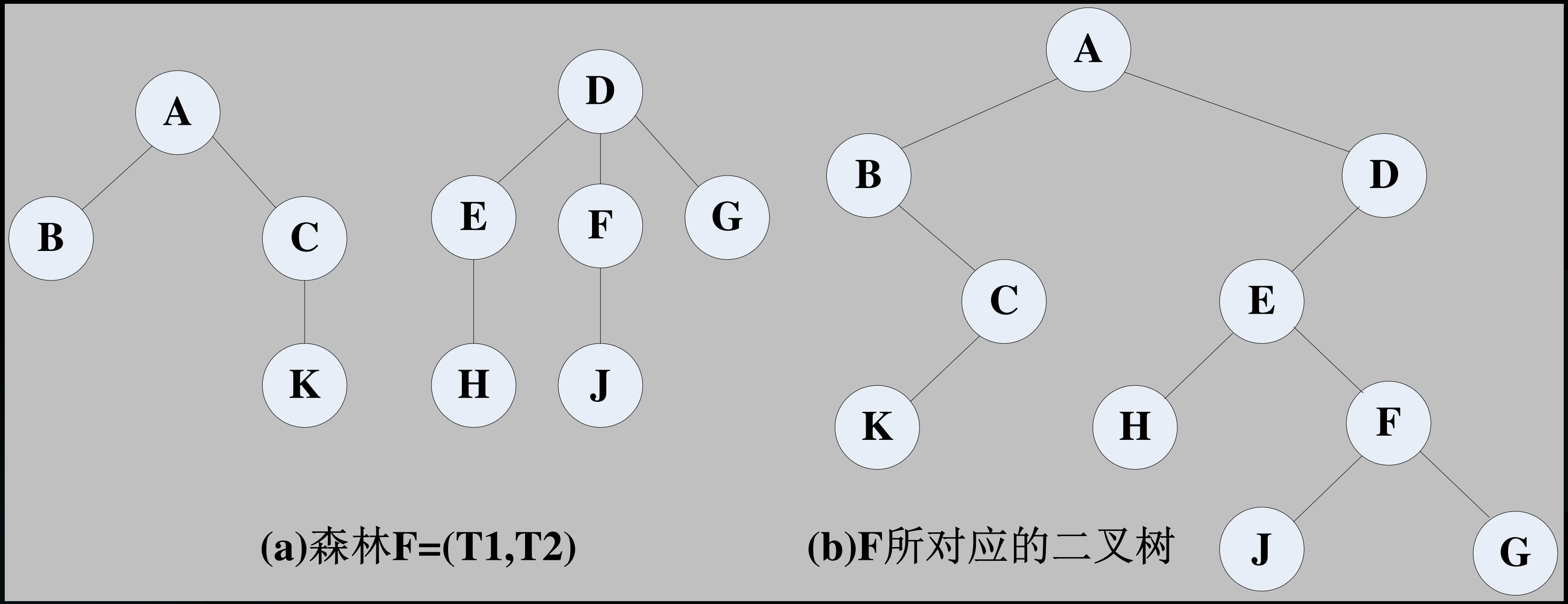
一棵二叉树B转换成的森林中有多少棵树？



经过3个结点，
故森林中3棵树

一棵二叉树转化成的森林中所具有的树的数目，等于二叉树从根结点开始沿右链到第一个没有右孩子的结点所经过的结点数目。


A
B
E
C
F
G
D
H
I
J
A
E
G
B
C
D
F
H
I
J

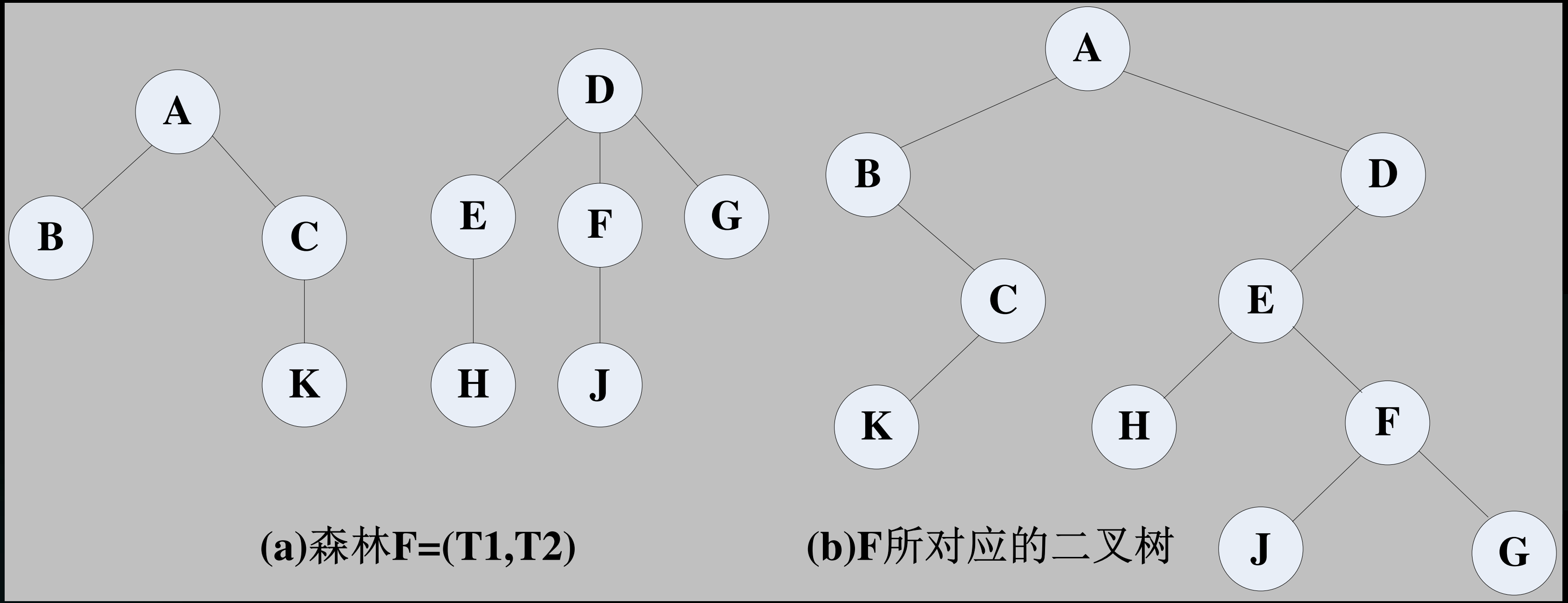


对上图（a)的森林的先序遍历的结果是：

A B C K D E H F J G

它等同于对(b)的二叉树的先序遍历。

对森林的先序遍历等于对每棵树先序遍历的简单拼接



对上图（a)的森林的中序遍历的结果是：

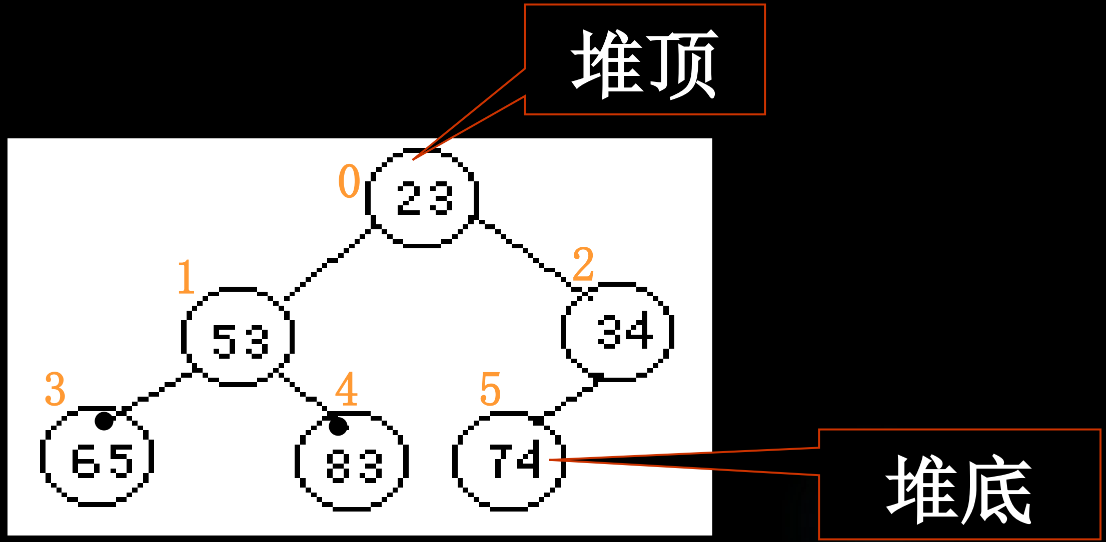
B K C A H E J F G D

它等同于对(b)的二叉树的中序遍历。

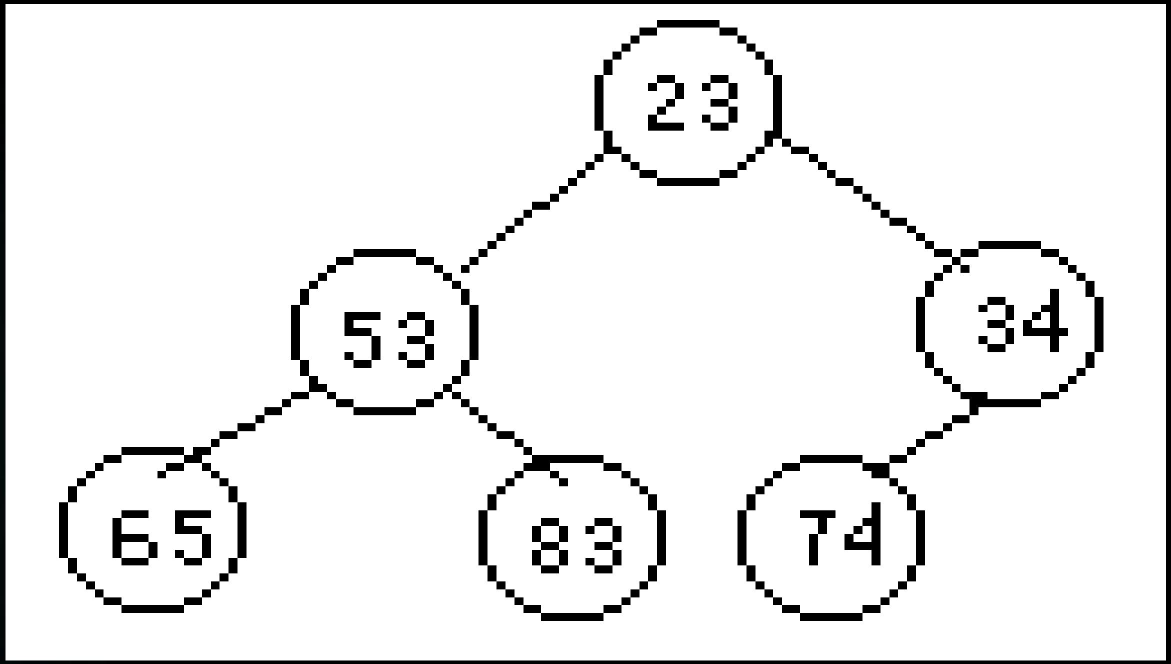
对森林的中序遍历等于对每棵树中序遍历的简单拼接

堆

- 一个大小为n的堆是一棵包含n个结点的完全二叉树，该树中每个结点的关键字值大于等于其双亲结点的关键字值
- 完全二叉树的根称为堆顶，它的关键字值是整棵树上最小的-最小堆
- 最大堆



判断序列（23，53，34，65，83，74）是最小堆吗



判断方法
先将序列画成完全二叉树形式
判断最小堆条件是否成立

双亲≤子女

给定序列不是最小堆
如何改造成唯一的最小堆？

1. 将序列画成堆（完全二叉树）的形式

2. 将堆调整为最小堆：

从⌊(n-2)/2⌋到0

结点i的双亲的序号为⌊(i-1)/2⌋

从完全二叉树最后一个叶子的双亲开始往前访问直到根结点

每访问一个结点，判断是否满足最小堆条件 双亲≤子女

IF 不满足，将该结点与最小孩子交换（向下调整）

交换完后再判断该结点是否满足最小堆条件

IF 不满足，继续向下调整，直到满足

3.将获得的最小堆中元素按照层次遍历顺序依次编号，并表示为一个序列

优先权队列中插入一个新元素的算法步骤：

1.首先将新元素插入到优先权队列的最后

2.检查新元素插入后，队列是否保持优先权队列的特点，若不能则需调整：

调整过程是由下向上，与双亲结点比较

若双亲结点大则新元素上浮，双亲结点下沉。

注：这一过程中与AdjustDown相反的比较路径

AdjustDown中调整结点与其孩子比较，不断下沉

本算法中调整结点与双亲比较，不断上浮

优先权队列中取出一个堆顶元素的算法步骤：

1.首先堆顶元素取出

2. 将堆底元素覆盖堆顶元素，删除堆底元素

3. 对堆顶元素进行向下调整

例 1： 向优先权队列中插入一个新元素24

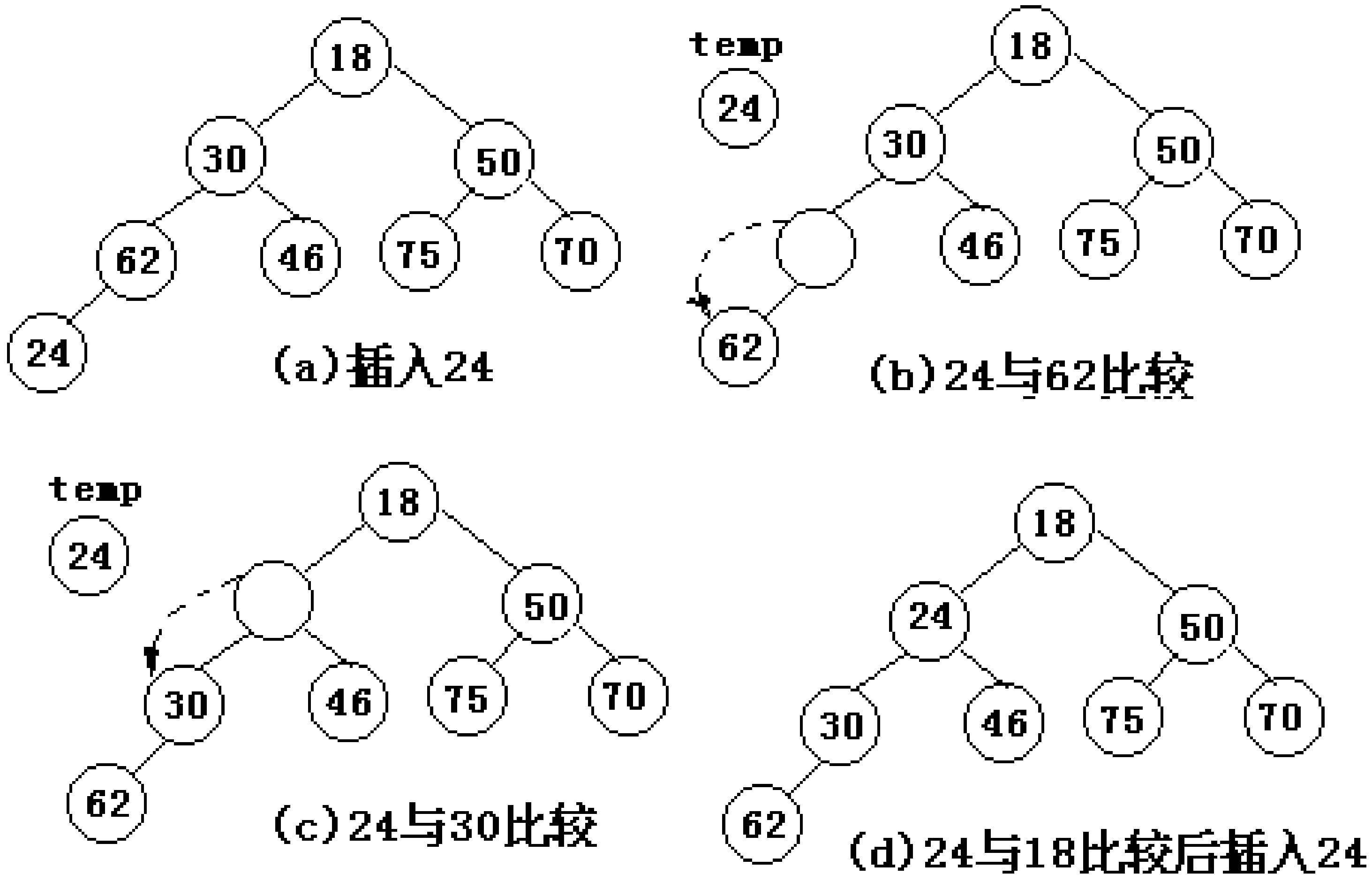


图 5.19 向上调整算法示例

哈夫曼算法可以描述如下：

(1) 用给定的一组权值$\{w_1,w_2\ ,\ldots,w_n\}$生成一个森林$F=\{T_1$，T_2，…，$T_n\}$,其中每棵二叉树T_i只有一个权值为w_i的根结点，其左、右子树均为空。

(2) 从F中选择两棵根结点权值最小的树作为新树根的左、右子树，新树根的权值是左、右子树根结点的权值之和。(约定左子树根权值小)

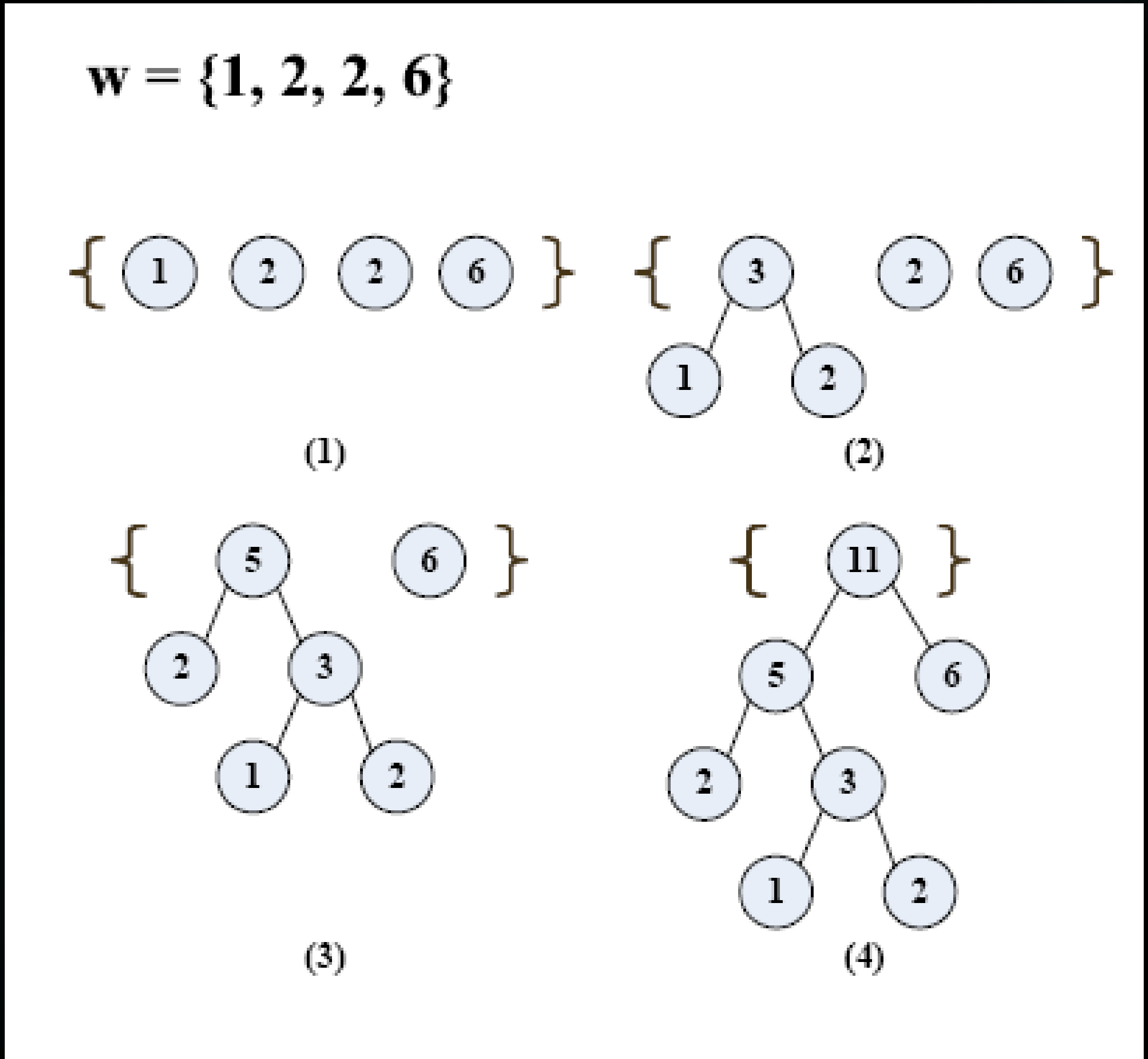
(3) 从F中删除这两棵树，将新二叉树加入F中。

(4) 重复(2)和(3)，直到F中只包含一棵树为止。此树即为哈夫曼树（重复执行几次？） **n-1**

构造哈夫曼树的过程

最高高度

WPL求法



约定左小右大

3. 哈夫曼编码

可以利用哈夫曼树得到前缀编码，即哈夫曼编码。
方法如下：1. 用权值构造哈夫曼树

2. 约定左分支为0，右分支为1。即左 0右 1

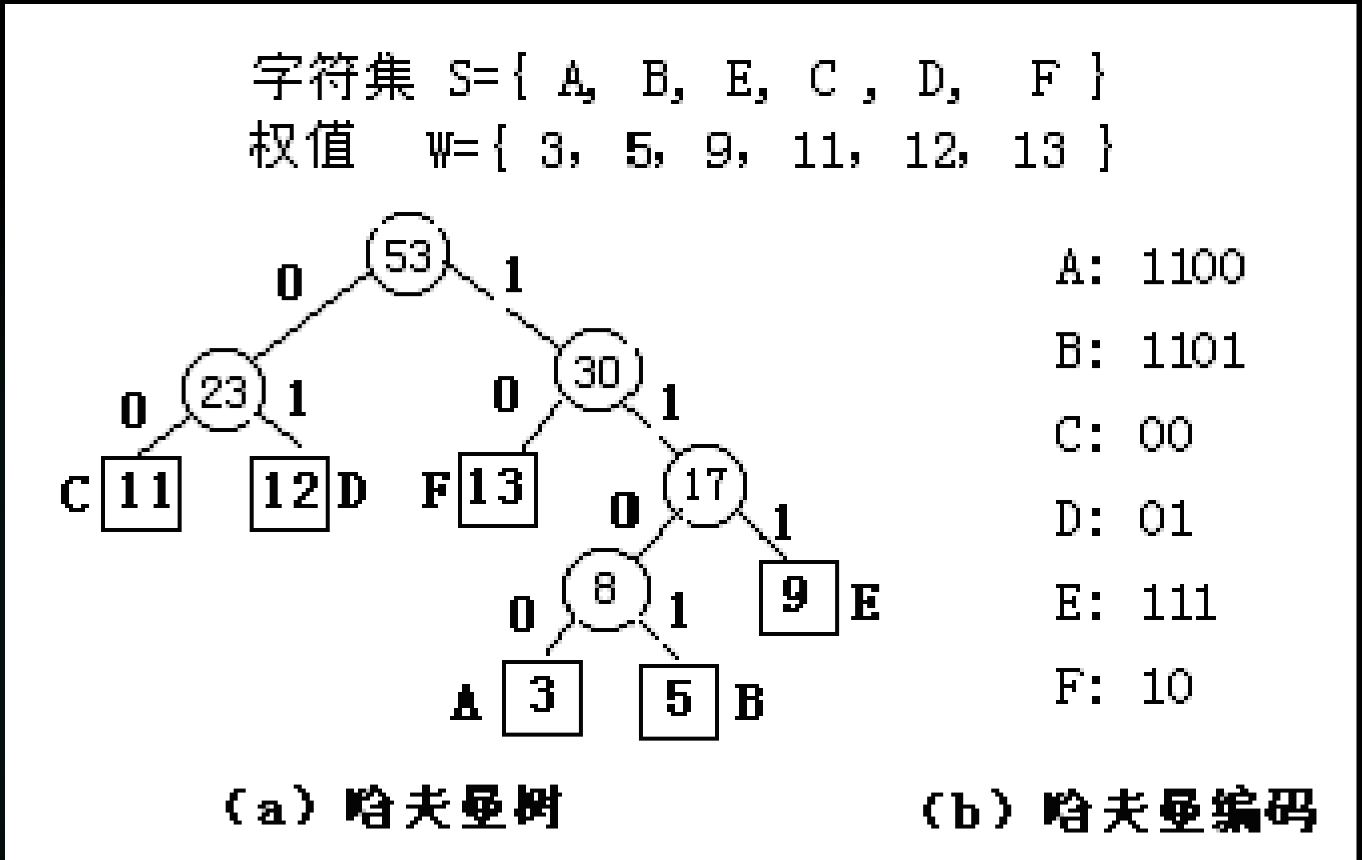


图5.22 哈夫曼编码示例

电文：

ABF

编码：

1100 1101 10

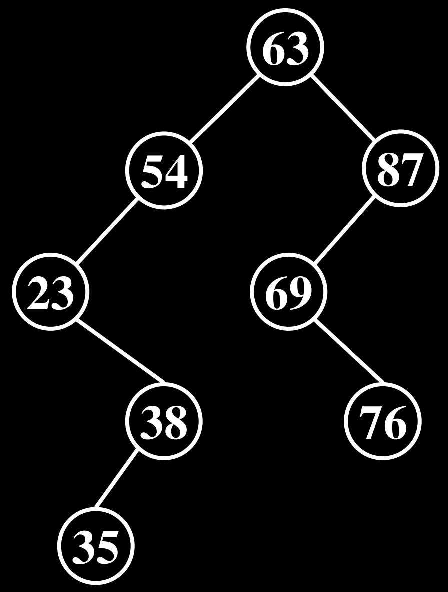
已知编码长度范围，已知两个字符编码，求最多还可以对多少字符编码

目录

对半搜索失败时最多比较次数（已知元素个数）

对半搜索成功平均搜索长度

目录



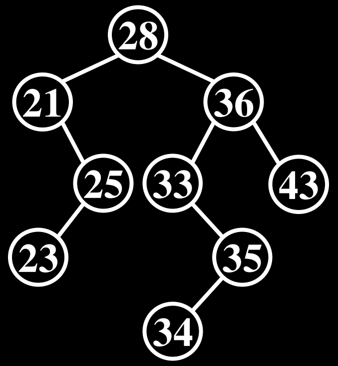
中序遍历:23 35 38 54 63 69 76 87

性质 若以中序遍历一棵二叉搜索树，将得到一个以关键字值递增排列的有序序列。

二叉搜索树
上删除元素

删除叶子结点

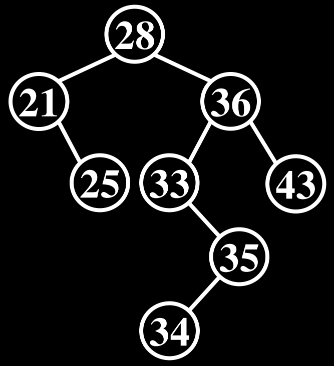
23



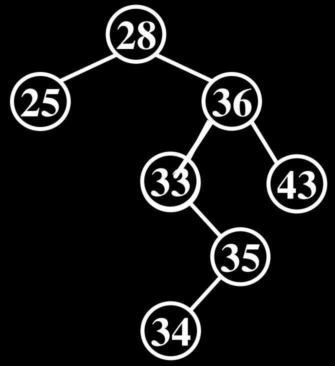
删除有两个孩子的结点

删除只有一个孩子的结点

21

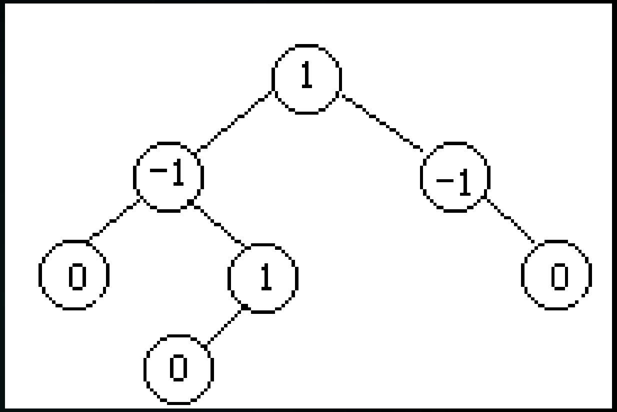


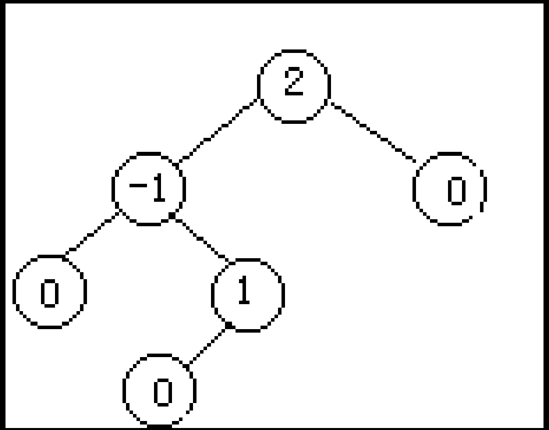
28



二叉平衡树的定义

- 定义　二叉平衡树又称AVL树
  它或者是一棵空二叉树，或者是具有下列性质的二叉树：
  （1）其根的左、右子树高度之差的绝对值不超过1；
  （2）其根的左、右子树都是二叉平衡树。



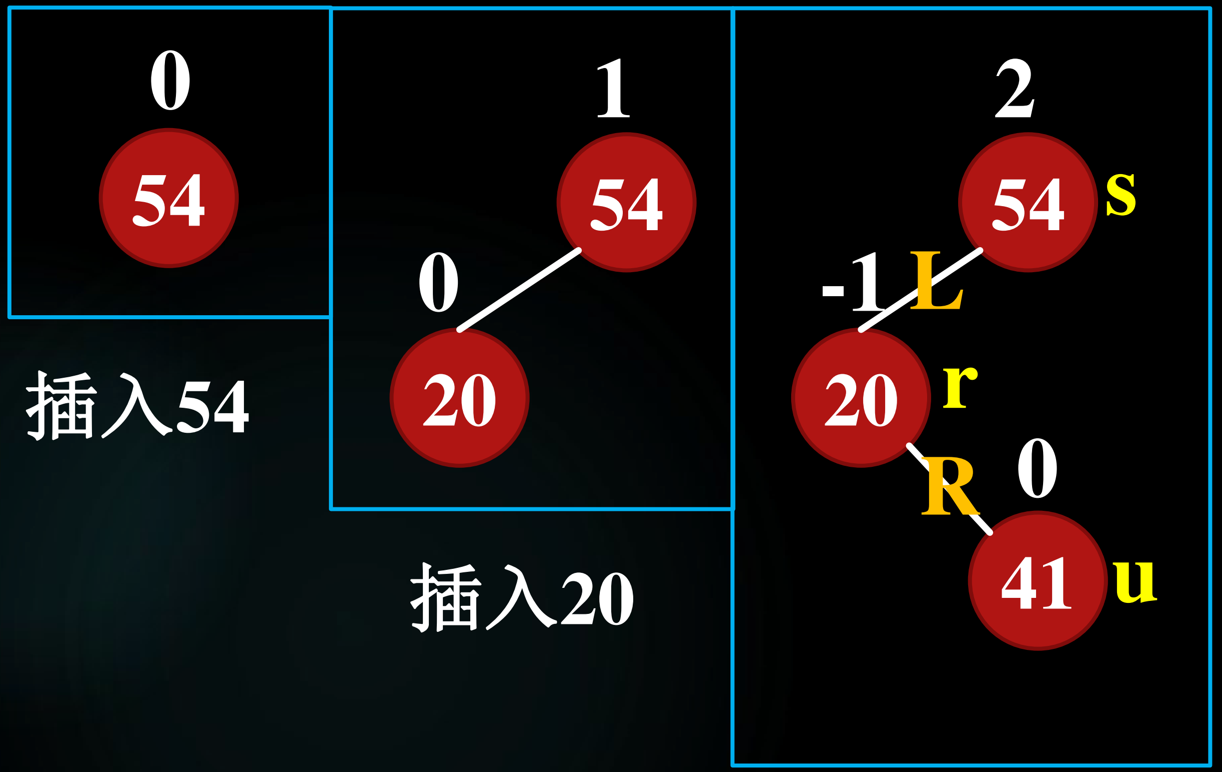


- 结点的平衡因子定义为该结点的左子树的高度减去右子树的高度
- AVL二叉搜索树既是二叉搜索树又是AVL树，具有平衡性和排序性。

AVL构造步骤

1. 先进行二叉搜索树的插入操作
2. 修改平衡因子
3. 取出最小不平衡子树
4. 在最小不平衡子树上标记s、r、u
5. 对s、r、u三个结点进行拆树和平衡
6. 将平衡后的子树放回原树中
7. 检查：平衡性、有序性

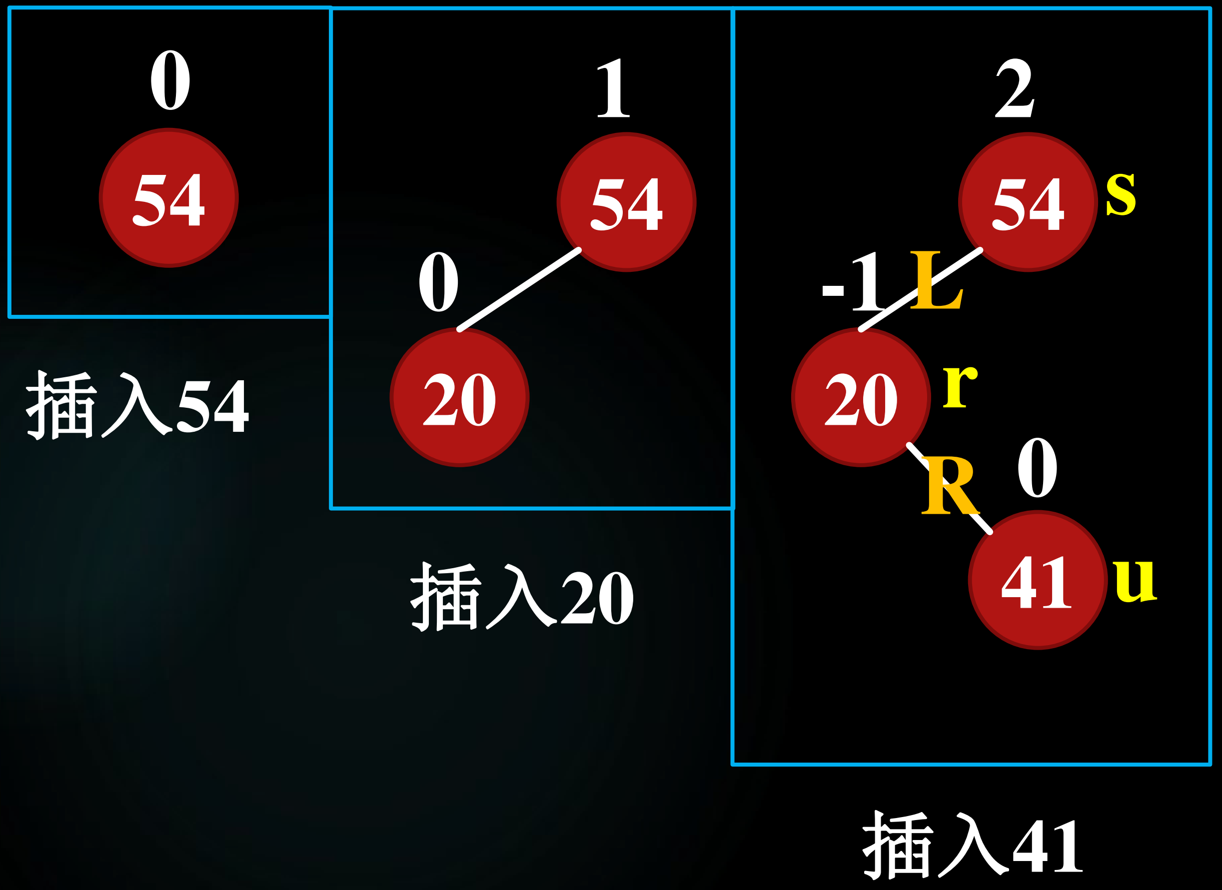
输入关键码序列：54，20，41，80，62，73
画出二叉平衡树的建立过程



插入54
插入20
插入41

最近不平衡祖先怎么找？
1. 从插入新结点位置往根结点方向遇到的第一个平衡因子（尚未更新）非零的结点
2. 插入后更新平衡因子，离插入结点最近的平衡因子为2或-2的结点

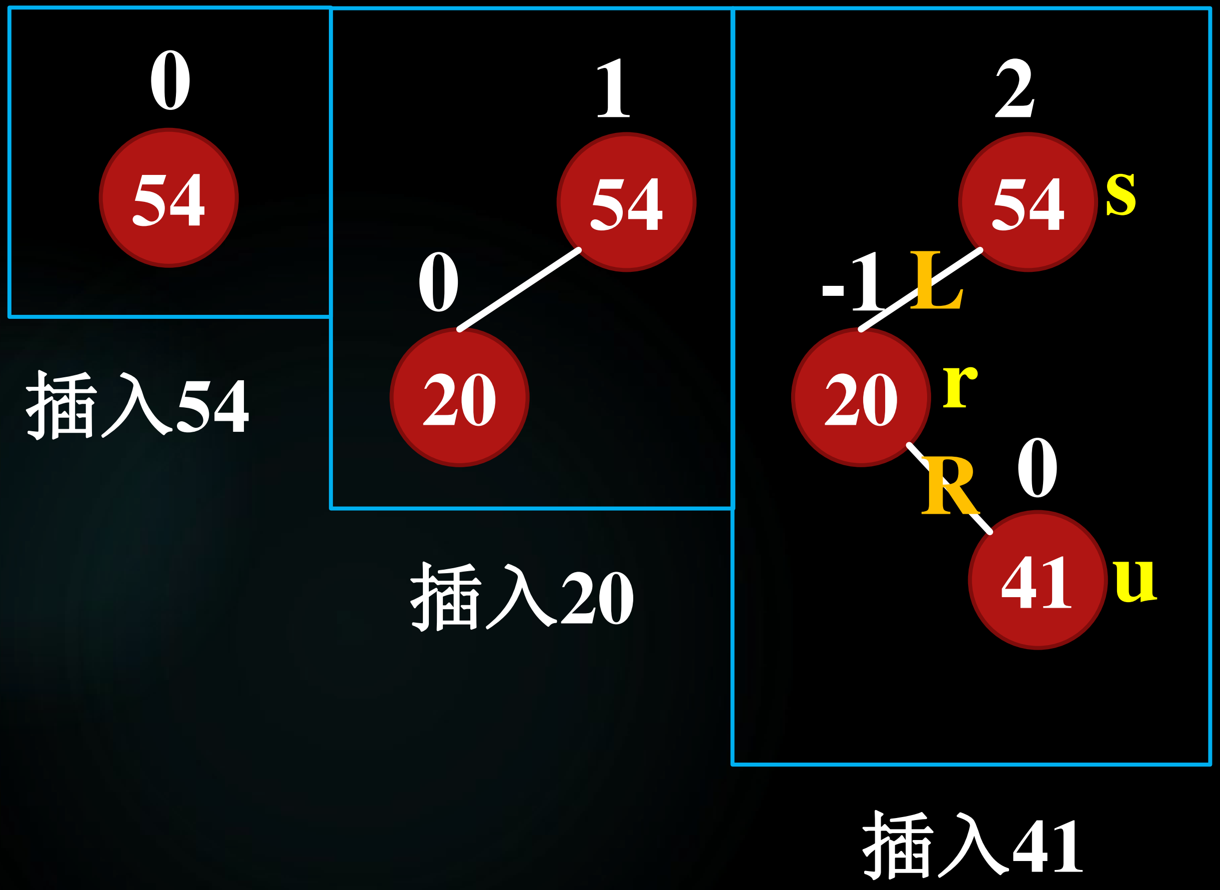
输入关键码序列：54，20，41，80，62，73
画出二叉平衡树的建立过程



如何标记s、r和u？

1. 最近不平衡祖先为s
2. 从s到插入的新结点路径上，紧靠s的后两个结点分别是r和u

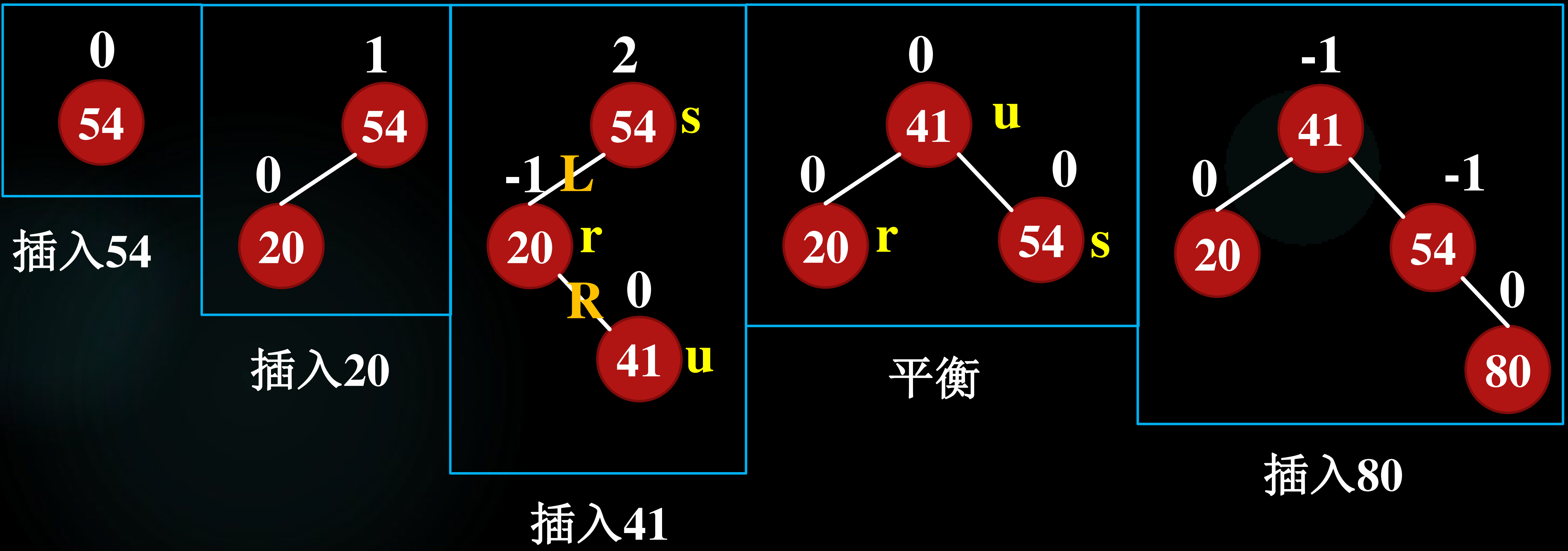
输入关键码序列：54，20，41，80，62，73
画出二叉平衡树的建立过程



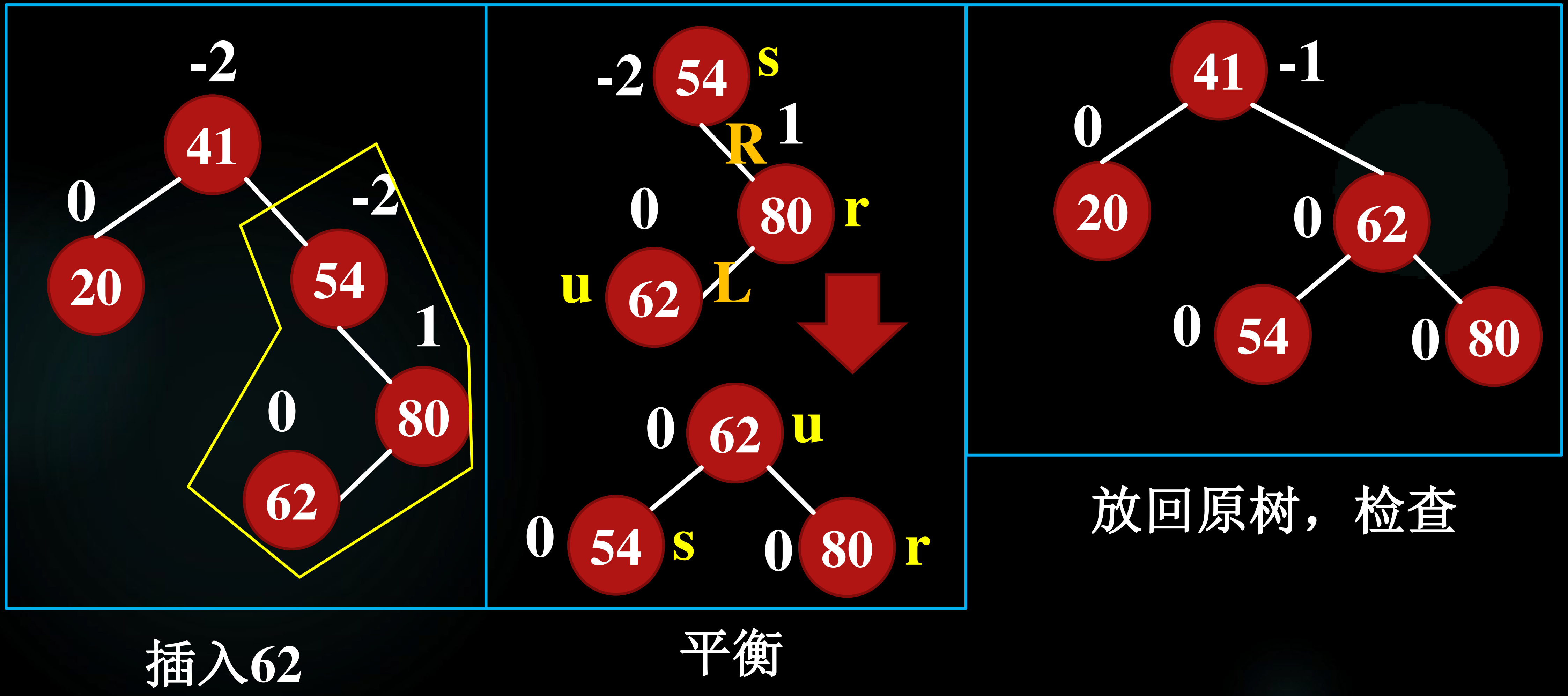
如何标记L和R？

1. s与r之间的路径如果为左分支，则为L，否则为R
2. r与u之间的路径如果为左分支，则为L，否则为R

输入关键码序列：54，20，41，80，62，73
画出二叉平衡树的建立过程

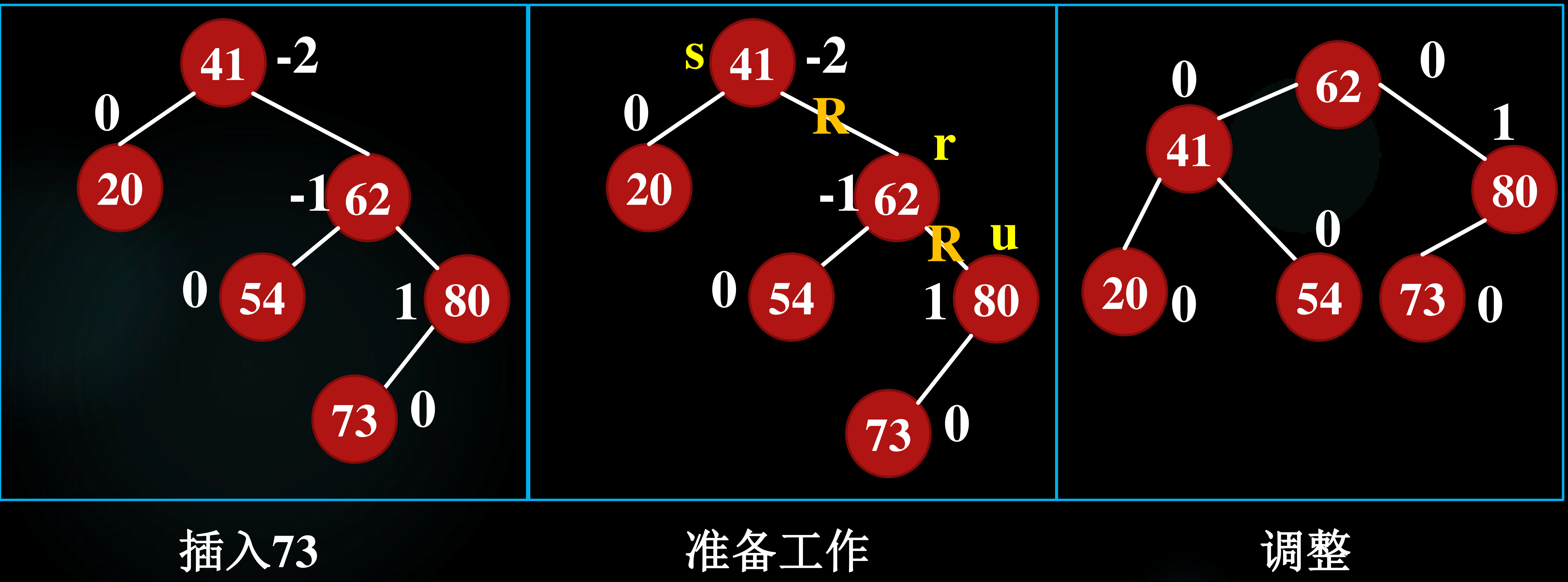


输入关键码序列：54，20，41，80，62，73
画出二叉平衡树的建立过程



放回原树，检查

输入关键码序列：54，20，41，80，62，73
画出二叉平衡树的建立过程



B-树的定义

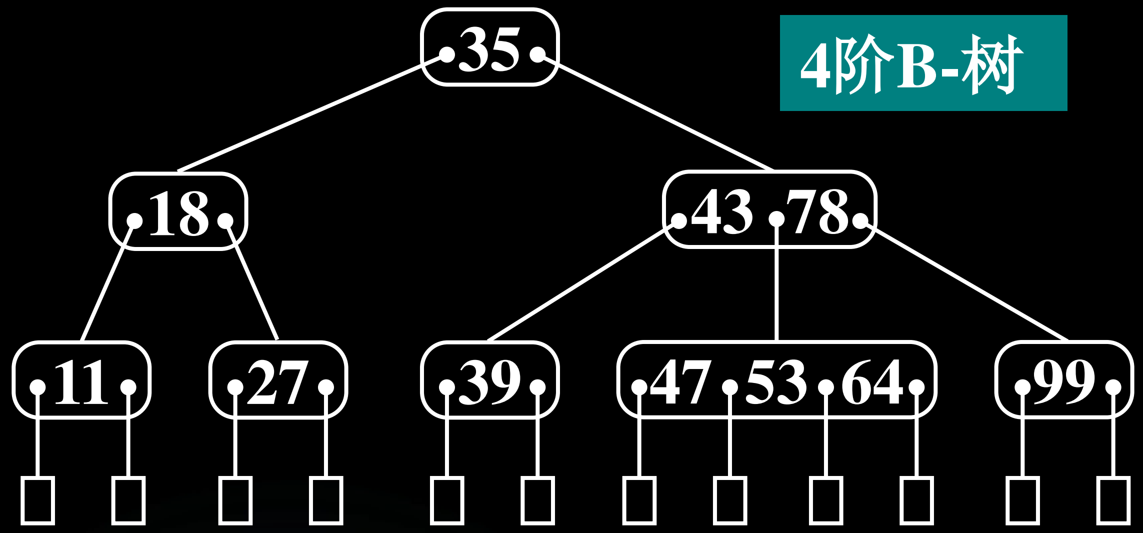
定义　一棵m阶B-树是一棵m叉搜索树，它或者是空B-树，或者是满足下列特性的树：

(1) 根结点至少有两个孩子 根结点可以只有一个元素

(2) 除根结点和失败结点外的所有结点至少有$\lceil m/2 \rceil$个孩子。

(3) 所有失败结点均在同一层上。 其他结点可能不允许只有一个元素，比如6阶-B树

考虑均衡性



4阶B-树

拿到一个B-树，进行检查工作

(1)首先看m是多少

(2)计算⌈m/2⌉是多少

(3)查看每个结点（根结点除外）的孩子数量有没有少于⌈m/2⌉，或超过m

(4)查看每个结点的关键字数量是否是孩子数量-1

从定义中可以得到，一棵m阶B-树中

(1)一个结点最多有m个孩子,m-1个关键字

(2)除根结点和失败结点外每个结点最少有⌈m/2⌉个孩子，⌈m/2⌉-1个关键字

(3)根结点最少有2个孩子

(4) 所有失败结点均在同一层上，失败结点的双亲是叶子结点

B-树的性质

性质 设B-树失败结点的总数是s，那么，一棵B-树包含的**元素总数N**是B-树的失败结点的总数s减一，即

$$N = s - 1$$

定理 N个元素的m阶B-树的高度h有 $h \le 1 + \log_{\lceil m/2 \rceil}(\frac{N+1}{2})$

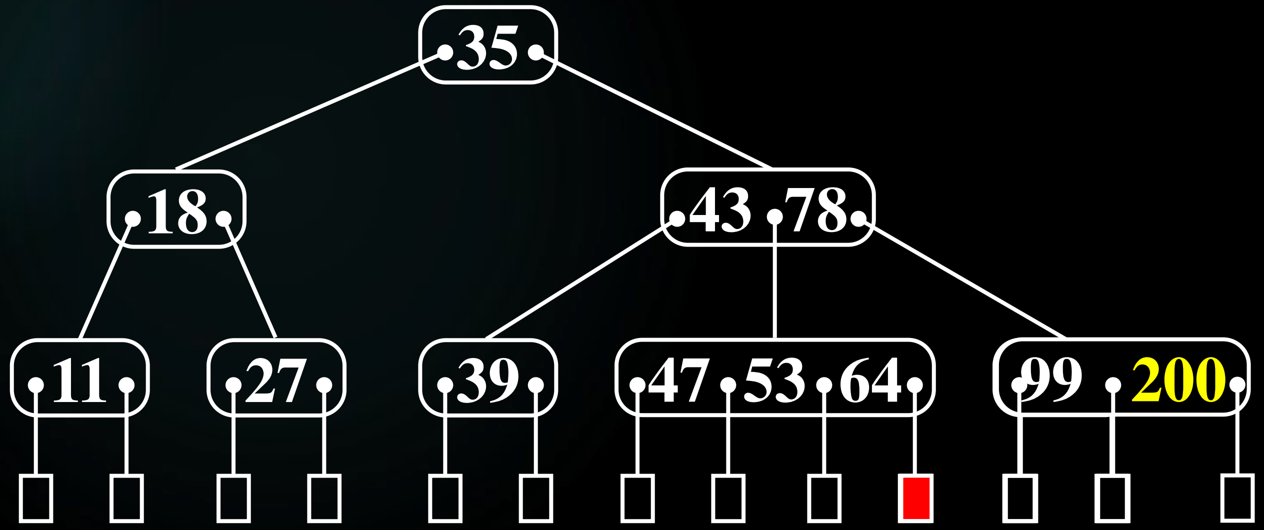
B-树的插入

将一个元素插入B-树的步骤：

（1）查重

（2）查重失败后停止在失败结点处，将新元素插在该失败结点的上一层叶子结点中

（3）如果插入后，该叶子结点中包含的元素个数不超过**m-1**，则插入成功完成，否则需作分裂。



4阶-B树

(1) 插入200

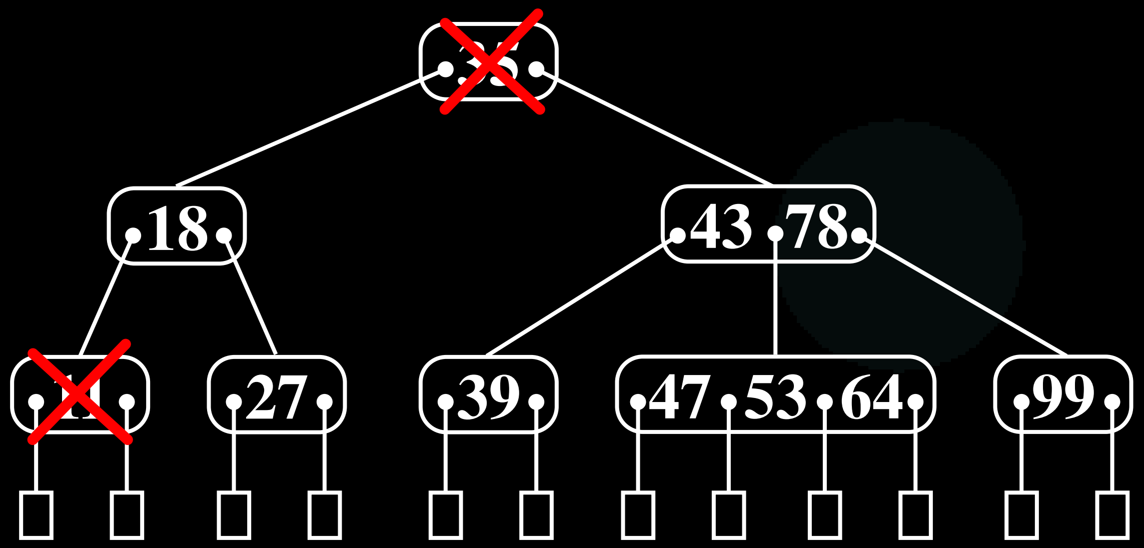
(2) 插入65

B-树的删除

B-树删除元素，分情况：

a. 元素在叶结点上

b. 元素不在叶结点上



目录

冲突：key1≠key2，但h(key1)=h(key2)的现象。

同义词：对同一散列函数，具有相同h值的关键字。

冲突是不可避免的。当冲突发生时，必须对冲突进行处理。

处理冲突的方法有：

（1）拉链法

（2）线性探查法

（3）二次探查法

（4）双散列法

线性探查法的缺点：很快表中所有位置的empty都变成F

易使元素在表中连成一片（线性聚集），探查次数增加，影响搜索效率

改进方法：1、二次探查法

2、双散列法

$h(key)=key\%11$

0	1	2	3	4	5	6	7	8	9	10
	35	24	80	15		13				65

插入35

$h(key)=35\%11=2$

$h_1(key)=(h(key)+1^2)\%11=3$

$h_2(key)=(h(key)-1^2)\%11=1$

插入13

$h(key)=13\%11=2$

$h_1(key)=(h(key)+1^2)\%11=3$

$h_2(key)=(h(key)-1^2)\%11=1$

$h_3(key)=(h(key)+2^2)\%11=6$

二次探测法能改善“线性聚集”，但是当二个关键字散列到同一位置时，则会有相同的探测序列，产生“二次聚集”。

2、双散列法

具备两个散列函数h1和h2

探查序列为：

$h_1(key)$，$(h_1(key)+h_2(key))\%M$，$(h_1(key)+2h_2(key))\%M$，…。

$h_1(key) = key \% 11$ $h_2(key) = key \% 9 + 1$

0	1	2	3	4	5	6	7	8	9	10
			80	15				58		65

插入58

$h_1(key)=58\%11=3$

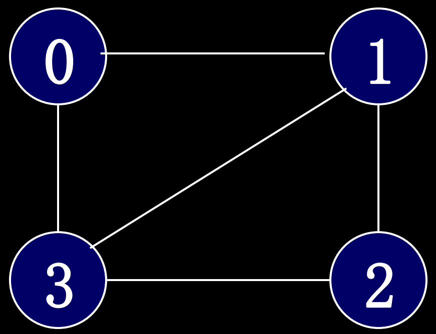
$h_2(key)=58\%9+1=5$

$(h_1(key)+h_2(key))\%11=(3+5)\%11=8$

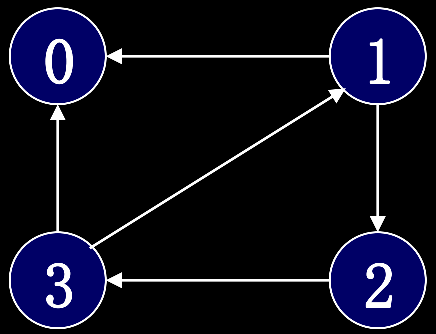
目录

	无向图	有向图
	自回路、多重图	自回路、多重图
	完全图	完全图
	邻接	顶点u邻接到v
	子图、路径	子图、路径
	简单路径、回路	简单路径、回路
	连通图、连通分量	强连通图、强连通分量
	顶点的度	顶点的度、入度、出度
	生成树	生成森林
		有向无环图DAG

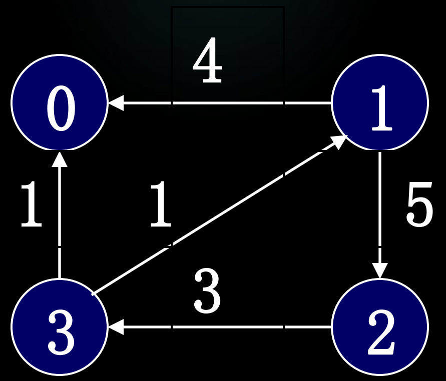
术语、握手定理



（a）无向图G_1



（b）有向图G_2



（c）网G_3

$$
\begin{array}{c|cccc|}
 & 0 & 1 & 2 & 3 \\
\hline
0 & 0 & 1 & 0 & 1 \\
1 & 1 & 0 & 1 & 1 \\
2 & 0 & 1 & 0 & 1 \\
3 & 1 & 1 & 1 & 0
\end{array}
$$

（d）图G_1的邻接矩阵

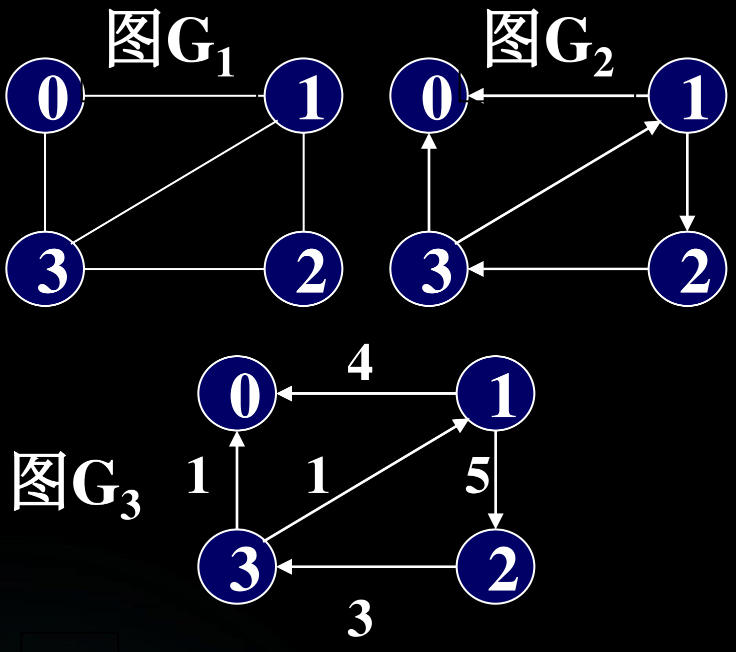
$$
\begin{array}{c|cccc|}
 & 0 & 1 & 2 & 3 \\
\hline
0 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 1 & 0 \\
2 & 0 & 0 & 0 & 1 \\
3 & 1 & 1 & 0 & 0
\end{array}
$$

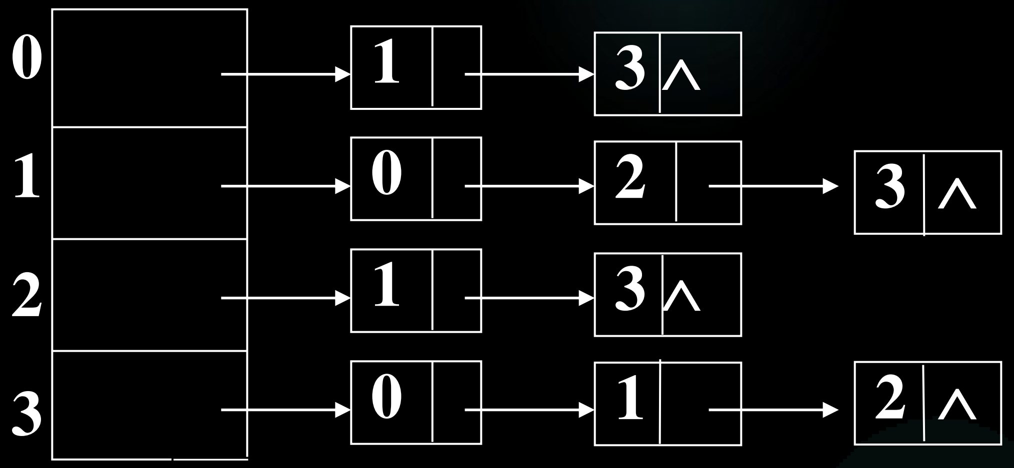
（e）图G_2的邻接矩阵

$$
\begin{array}{c|cccc|}
 & 0 & 1 & 2 & 3 \\
\hline
0 & 0 & \infty & \infty & \infty \\
1 & 4 & 0 & 5 & \infty \\
2 & \infty & \infty & 0 & 3 \\
3 & 1 & 1 & \infty & 0
\end{array}
$$

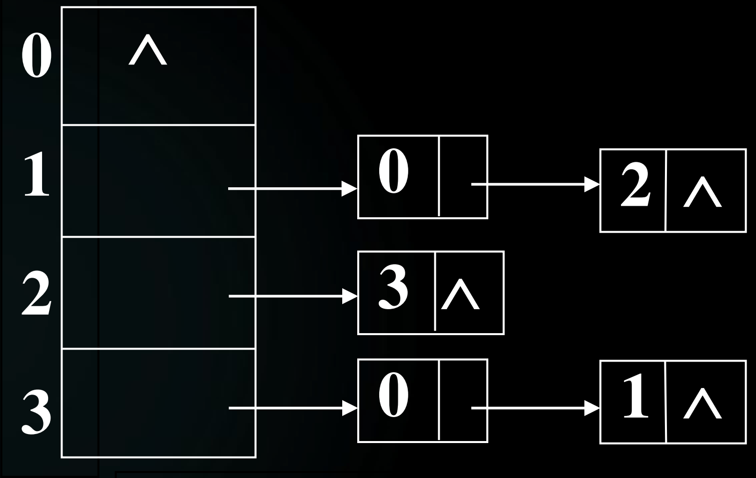
（f）网G_3的邻接矩阵

两种存储方式所需存储空间判断

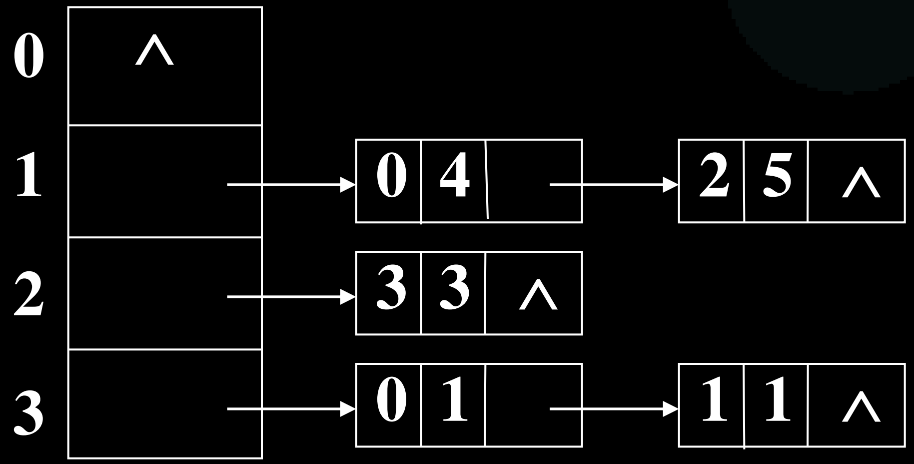




(a)图G_1的邻接表



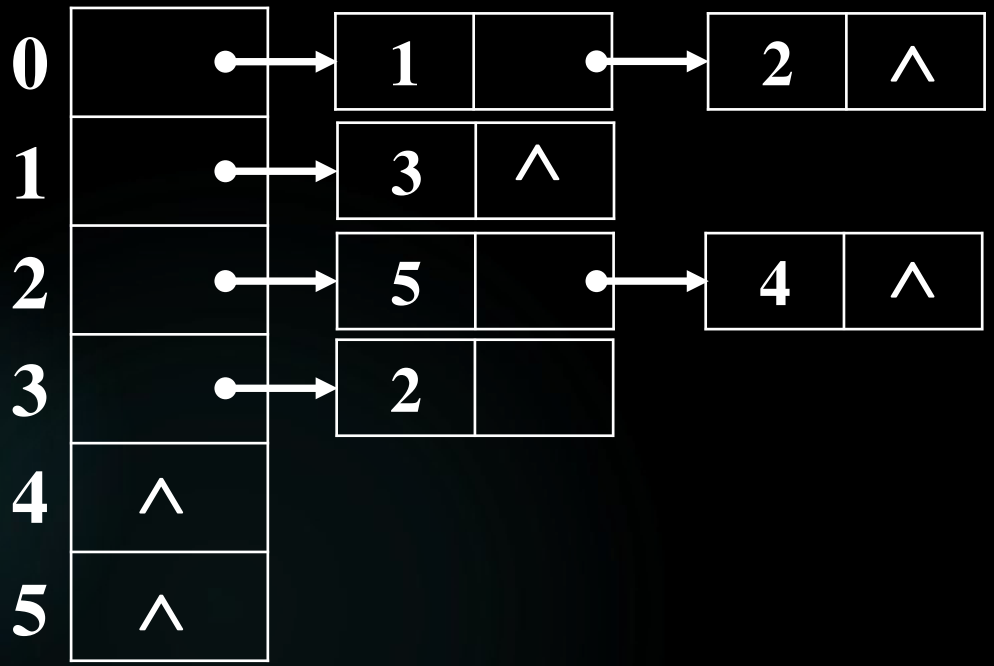
(b)图G_2的邻接表

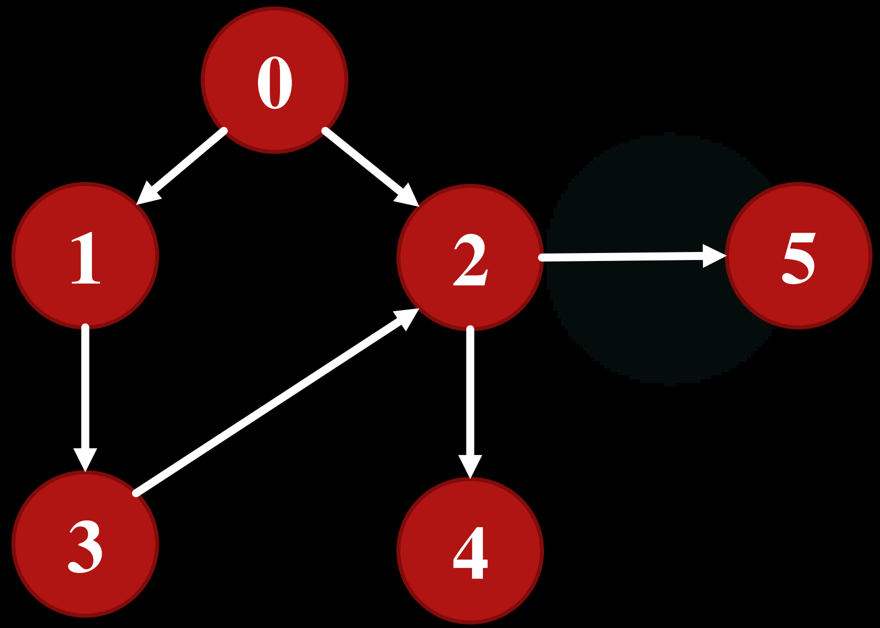


(c)图G_3的邻接表

图的遍历：指从图G的任意一个顶点v出发，访问图中所有结点且每个结点仅访问一次的过程。

沿着边的方向行走



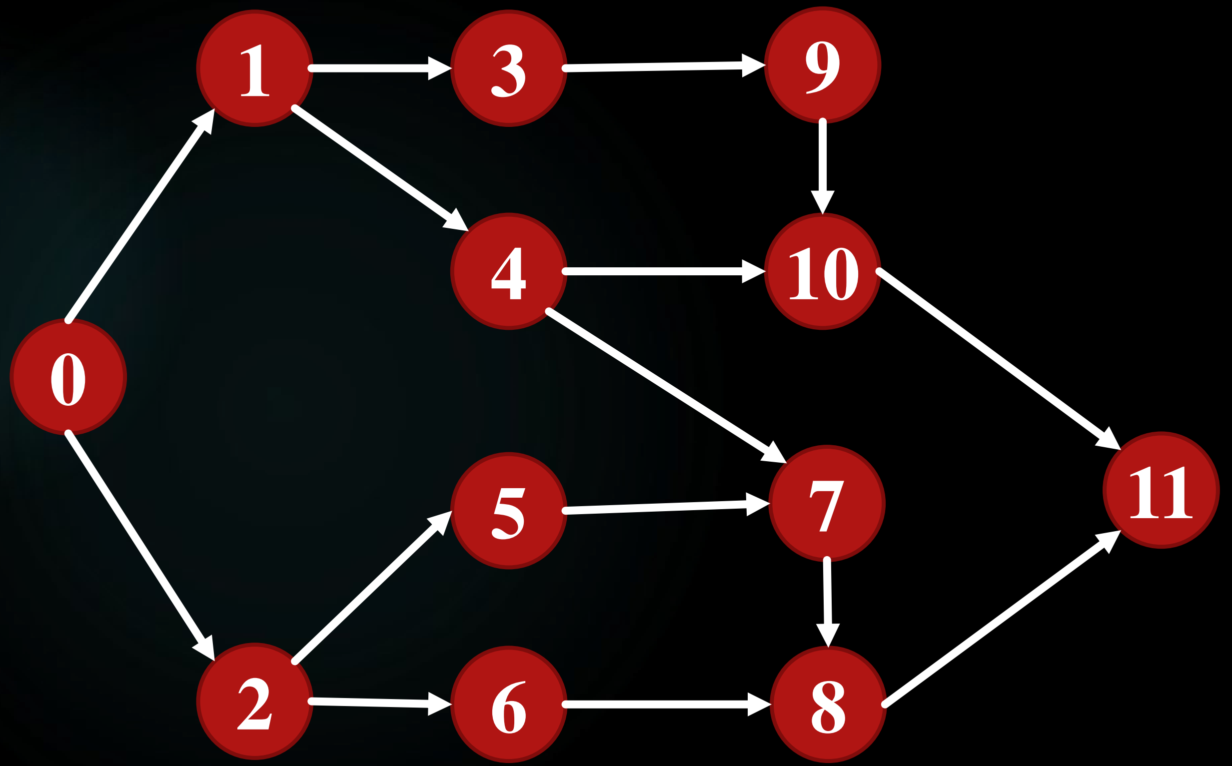


图遍历的方法:深度优先搜索（类似于树的先序遍历）和宽度优先搜索（类似于树的按层次遍历）

时间复杂度

什么是拓扑排序

一个拓扑序列是AOV网中顶点的线性序列，使得对图中任意二个顶点i和j，若在图中，i领先于j，则在线性序列中i是j的前驱结点。

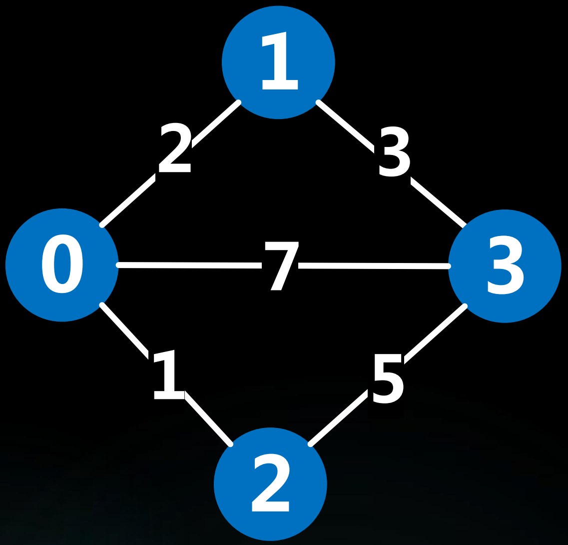


0,1,4,3,2,5,6,7,8,9,10,11
0,1,3,4,9,10,2,6,5,7,8,11

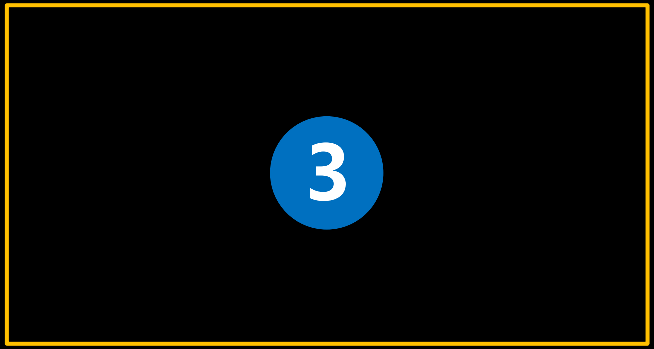
检测拓扑排序的方法

对图中每一条边<i,j>，在序列中i排在j之前

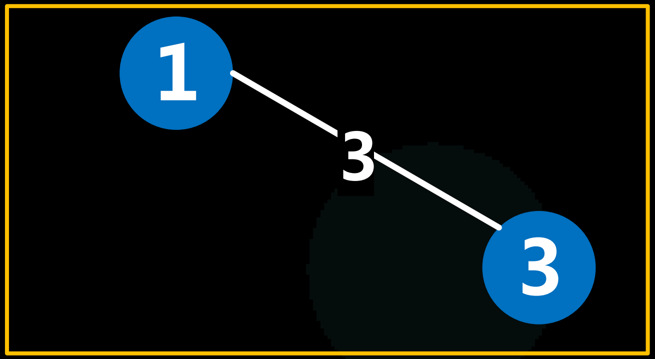
普里姆算法



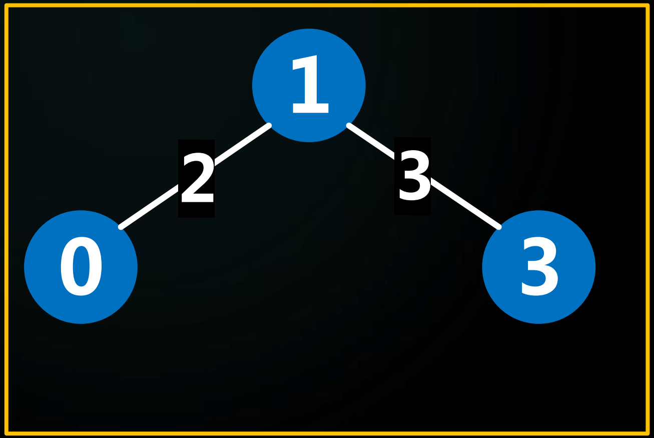
任选一个顶点作为起点



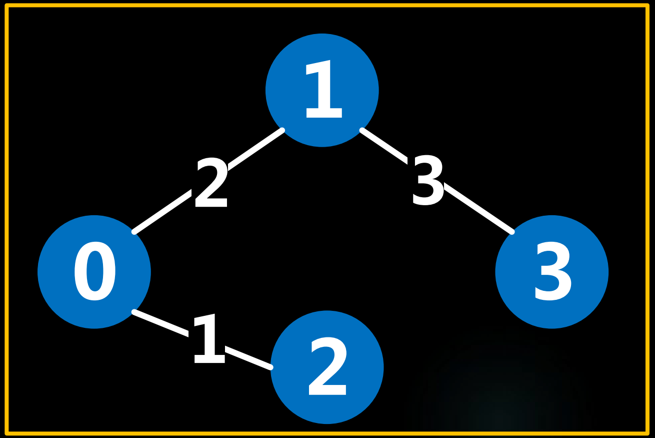
选择一条代价最短的边(3,x)



选择一条代价最短的边(3,x)或(1,x)新添加的边不能造成回路

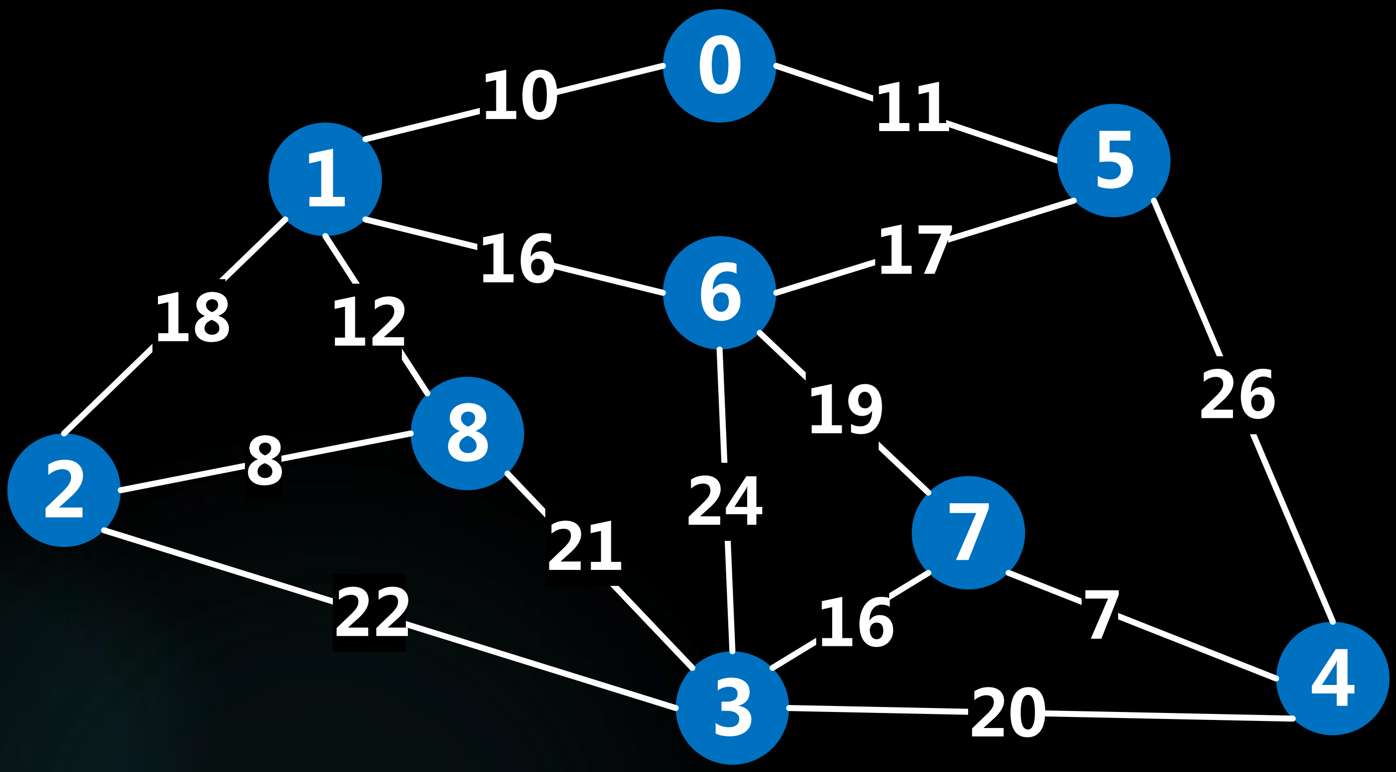


选择一条代价最短的边(3,x)、(1,x)或(0,x)，新添加的边不能造成回路



克鲁斯卡尔的思想

通过找n-1条不构成回路的最小权值边，来得到最小代价生成树。



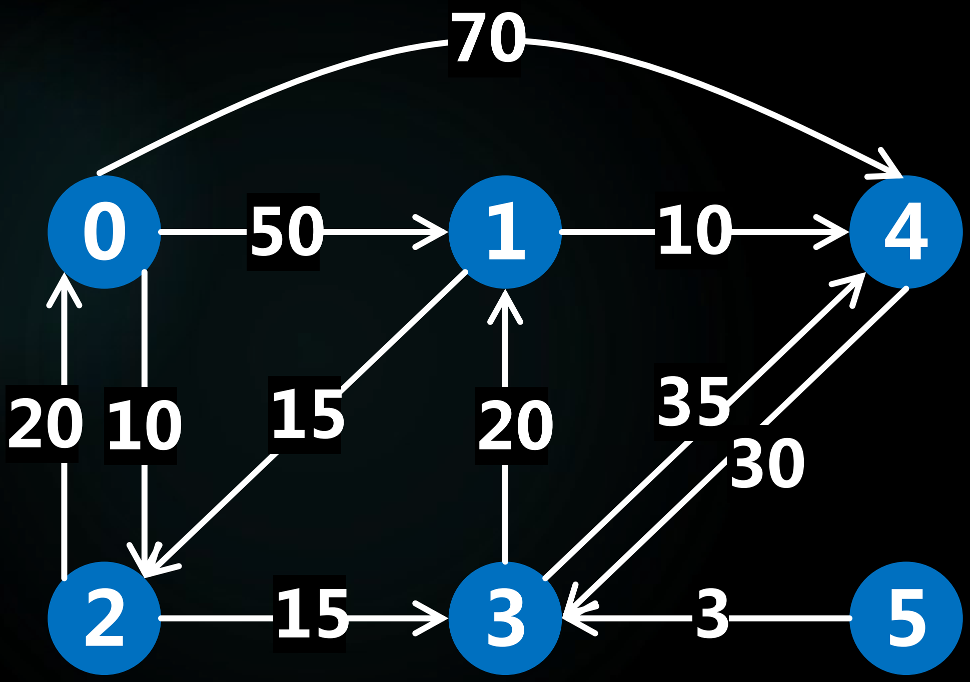
先将边按照权值从小到大排列
初始时最小生成树T=(V,{ })

u	v	权值
7	4	7
2	8	8
1	0	10
0	5	11
1	8	12
1	6	16
3	7	16
6	5	17
1	2	18
6	7	19
3	4	20
8	3	21
2	3	22
6	3	24
5	4	26

单源最短路径算法-迪杰斯特拉

算法思想：……再求得长度次短的
一条最短路……点之间的最短
路径都已求得为止

按照路径长度递增的次序产生最短路径



源点	终点	最短路径	路径长度
0	1	0,2,3,1	45
	2	0,2	10
	3	0,2,3	25
	4	0,2,3,1,4	55
	5	-	∞

目录

内排序的基本概念

掌握各种排序算法

(a) 算法的手工排序过程，即写出各趟排序结果；

(b) 稳定性；

(c) 时间复杂度(最好、最坏和平均)

(d) 已知趟数，判断排序方法；

(e) 已知排序结果，判断排序方法；

(f) 排序算法的比较次数；