**三个层面：**

1. 基础概念的掌握（30%）
2. 基本方法、基本原理的掌握（50%）
3. 算法（20%）

第一章

***第一层面：***

* 数据: 用以**描述客观实体**，并**能被计算机识别**的符号集。
* 数据元素: 通常**具有完整确定意义**的，**描述现实某客观实体**的最小数据集。
* 数据对象: 具有**相同性质**的数据元素的集合，即同类数据的集合。
* 数据结构: **给定数据对象**及其**上面定义的操作**（运算、关系）所共同形成的系统。
* 数据结构研究的方面：①**逻辑关系（结构）** ②**存储（物理）关系** ③**运算关系**
* 常见的逻辑关系： ①**线性关系** ②非线性关系：**树形关系**、**图关系**
* 常见的存储（物理）关系：①**顺序结构** ②**链接结构** ③**索引结构**（顺序结构和链接结构的结合） ④**散列（哈希）结构**
* 数据结构评价的主要标准：

①作为问题的参数的函数来计算**时间效率**

②作为问题的参数的函数来计算**存储需要量**

第二章

***第一层面：***

* 算法：算法是解决某个问题的**指令的有限集合**，此外每个算法都必须符合**有穷性**、**确定性**、**可行性**以及**输入/输出**等准则。
* 过程：事物发展或事物进行所经过的程序，即次序，一般基于时间表征。

过程基于其**目的要求**可分为**一般过程**和**函数过程**。

* 有效算法：**以多项式时间为界限**的算法称为有效算法。
* 难解性问题：如果一个问题**不存在多项式时间算法**，则称它是难解的。
* 算法设计需要遵循的基本规则和准则：①**正确性** ②**易读性** ③**健壮性** ④**高效性**（效率与低存储量的需求）
* 常见的算法描述基本方法：①（具体的）**计算机程序设计语言** ②**自然语言** ③**PDL语言**（也称为伪程序设计语言，或伪码语言） ④**流程图**（框图，或N-S图）

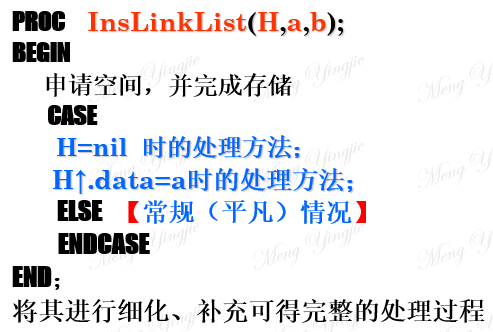
第三章

***第一层面：***

* 线性表：一个线性表（linear list）是n≥0个数据元素a1、a2、……、an的**有限序列**，序列中除第一个和最后一个以外，每个元素**有且仅有一个直接前驱和直接后继**。
* 链表：**通过指针联系起来的结点的整体**。
* 静态链表：**以整型变量的值作为存储链接的指针值**，从而联系起来的结点的整体。
* 表头结点：给链表**增加的一个附加结点**，该结点一般**放置于链表的最前面**，但该结点不用以存储数据元素。
* 存储密度：

***第三层面：***

* 基于链接结构的插入与删除问题



第四章

***第一层面：***

* 栈：栈（stack）是一个**下限为常数**（栈底），**上限可变化**（栈顶）的**向量**（**或者反之**）。栈也称为堆栈或堆阵。
* 队列：队列（queue）是一个**下限和上限只能增加而不能减少**的**向量**（**或者反之**）。
* 栈、队列与线性表的异同：

同：

① **数据对象**相同 ②**数据逻辑关系**相同 ③**数据存储结构**相同

异：

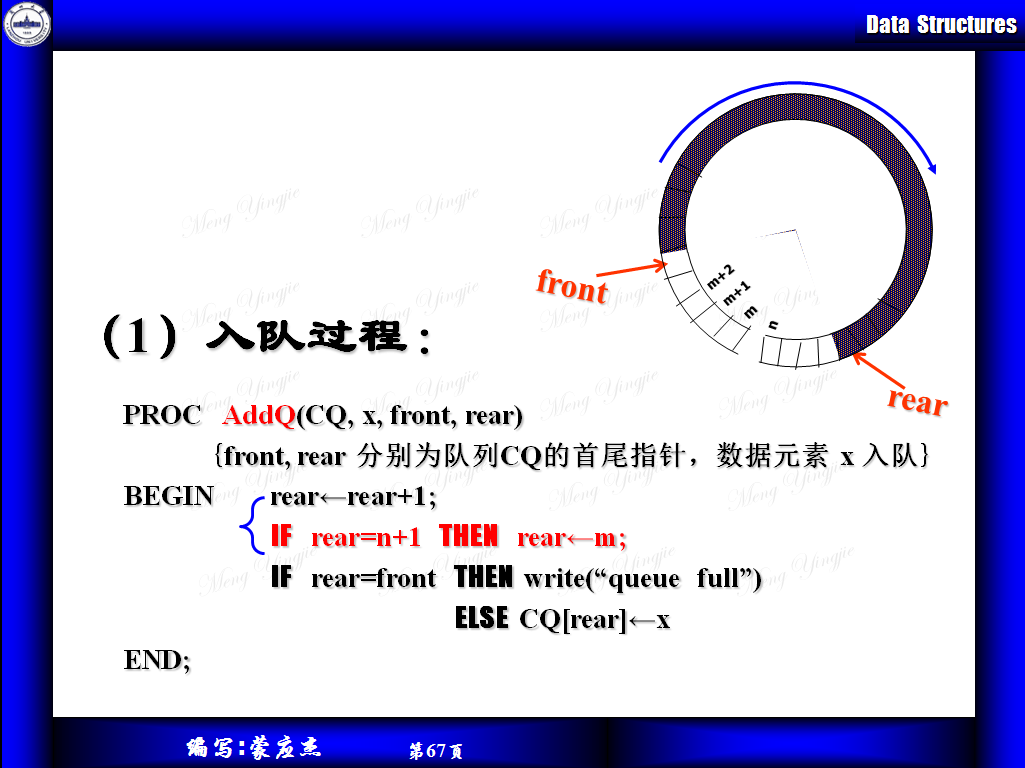
栈：①只能在栈顶进行插入和删除 ②不能进行线性表的操作中除插入和删除以外的操作

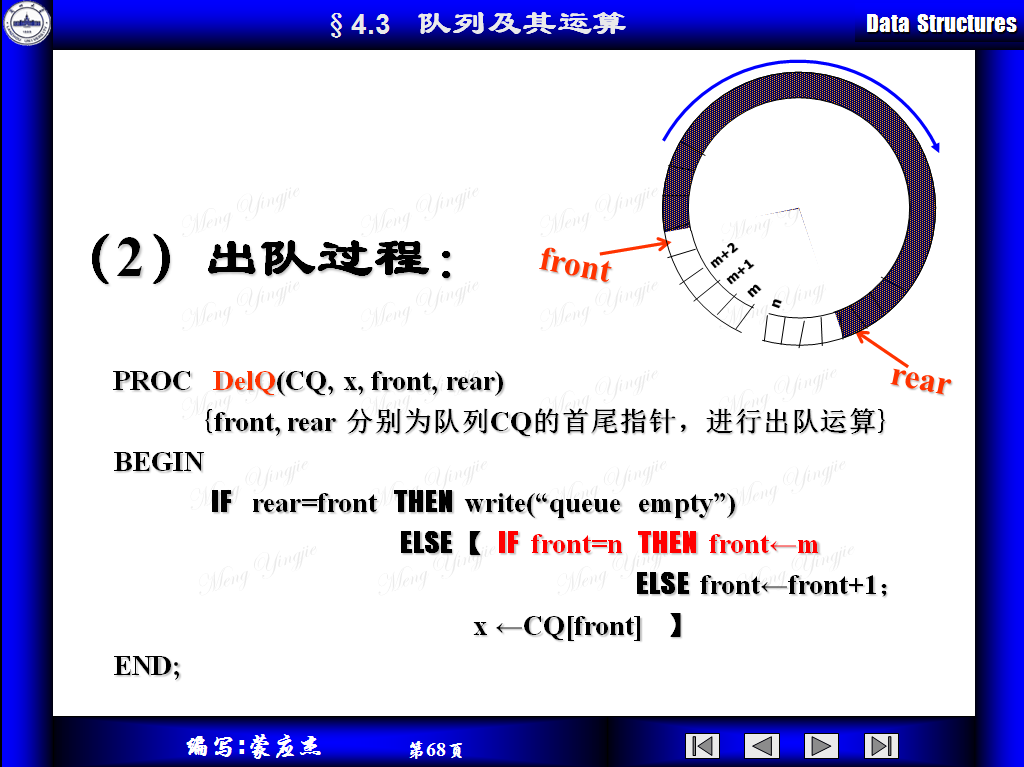
队列：①只能在一端插入，另一端删除 ②不能进行线性表的操作中除插入和删除以外的操作

* 假溢出：由于**空间分配或管理不当**，导致的**空间还有**，**数据却无法装入**的现象。

***第三层面：***

* 循环队列的出入队：





第五章

***第一层面：***

* 串：一个由零个或多个字符组成的**有穷序列**称为串。
* 串的模式匹配：把**从目标串中查找模式串**的过程称为串的模式匹配。

第六章

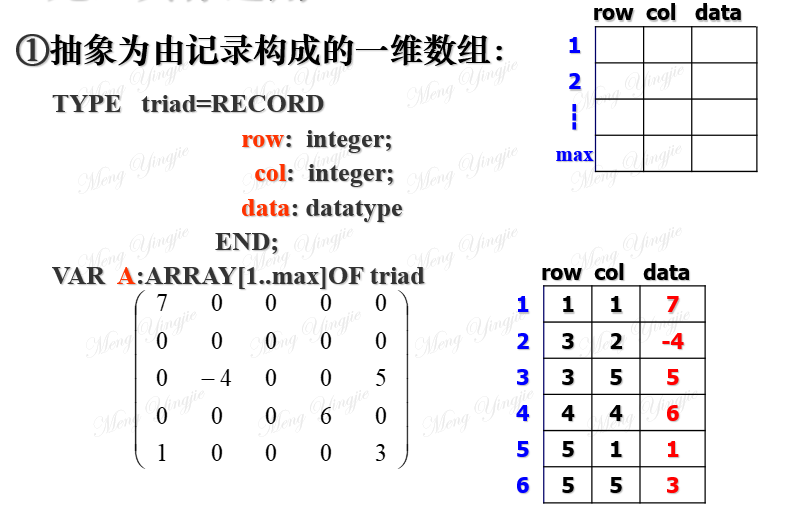
***第一层面：***

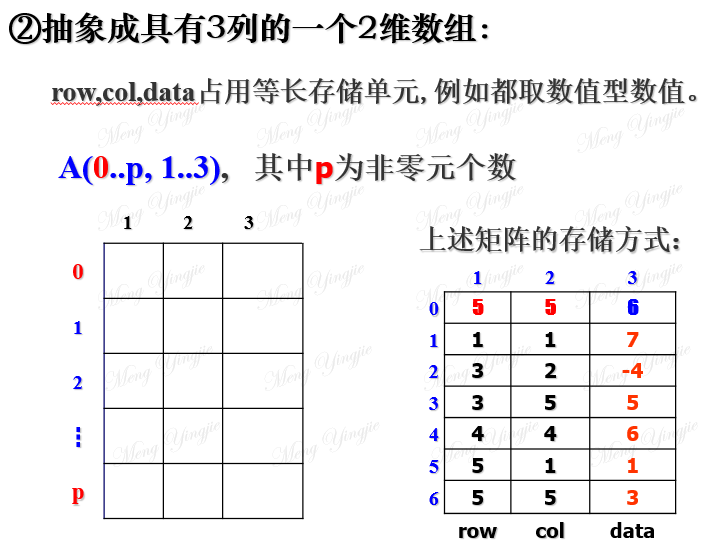
* 数组：1维数组是个**向量**，它的每个元素是该结构中不可分割的**最小单位**；n（n>1）维数组是个向量，它的每个元素是n-1维数组，且具有相同的下限和上限。
* 稀疏数组：在一个数组当中**和某元素比较**而言，**不相同的元素很少**时，我们称此数组为稀疏数组。
* 稀疏矩阵：是稀疏数组的典型例子。
* 广义表：是**零个或多个原子或子表**所组成的**有限序列**。一般简称为表（或列表）。
* 稀疏矩阵与特殊矩阵的差异：**零元素的分布有无规律**

***第二层面：***

* 稀疏矩阵的存储方法：

①三元组：**将矩阵的全部非零元素抽象为一个线性表，每个元素含行、列、值三个数据项。**





**检索可依据行下标值进行二分法搜索，存取时间O(log2p),存储空间需要O(3p)个单元。**

**三元组的缺陷：**

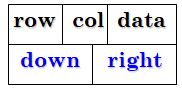
**不利于矩阵元素变化：**

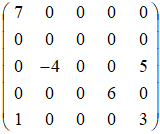
**由非零变为零引起删除运算。**

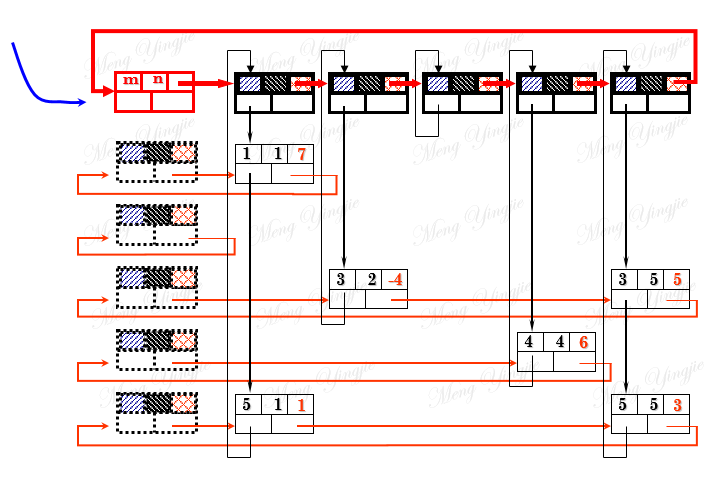
**由零变为非零引起插入运算。**

②十字链表：**对于非零元素依据行、列特性,每一行和每一列建立一个循环链表。**

**结点结构如下：**



稀疏矩阵的十字链表存储结构如下：



第七章

***第一层面：***

* 树：树（tree）T，是满足如下性质的有限个结点组成的非空集合：

①T中**有且仅有**一个称为**根**的结点；

②除根结点之外，**其余结点**分成m（m>0）个**不相交的集合**T1，T2，…，Tm，其中每个Ti都是树，而且都称为T的子树。

* 二叉树：二叉树(binary tree)T是满足如下性质的结点的有限集合：

T是**空集**；或者T包含**一个根结点**且**其余结点分成两个不相交的集合**，并分别被称为根结点的左子树和右子树。

* 遍历：对于给定的数据结构，**系统的访问**该结构中的每个结点,且每个结点仅被访问一次的操作过程称～。
* 二叉树的遍历：对于给定的二叉树，系统地访问二叉树中的每个结点，且每个结点仅被访问一次的操作过程称～。
* 二叉树的遍历规则：

1. **层次策略**：

自上而下：从左到右、从右到左

自下而上：从左到右、从右到左

1. **深度策略**：

约定先左后右次序后可分为：先根（DLR）、中根（LDR）和后根（LRD）

* 线索：将二叉树的**空指针**利用起来，用于表示**某线性关系下前驱或后继**时，这种指针称为线索。
* 线索树：**带线索的**二叉树称为线索二叉树，简称线索树。
* 线索化：给二叉树**加线索**的过程称线索化。

二叉排序树：二叉排序树或者是空二叉树，或者是满足如下性质的二叉树：

①若它的左子树非空，则**左子树上所有结点的值**均小于根结点的值；

②若它的右子树非空，则**右子树上所有结点的值**均大于根结点的值。

* 霍夫曼树（哈夫曼树）：Huffman最早提出了**构造具有最小加权路径长度**最小的二叉树的算法，此算法称Huffman算法，按此算法构造的具有最小加权路径长度的二叉树称Huffman树(或最优二叉树)。

Huffman算法的基本思想：

① 给定一组权值集合{w1,w2…wn},据此构成n棵二叉树组成的森林F。注：F中的每棵二叉树只有一个带权值为Wi（1≤i≤n）的根结点。

② 将F={T1，T2…Tn}按根结点的值由小到大进行排序。

③ 取出T1和T2组成一棵二叉树T，再将T插入到F中，并使F依据根结点的值有序。

④ 反复执行③直到F={T}为止。

***第二层面：***

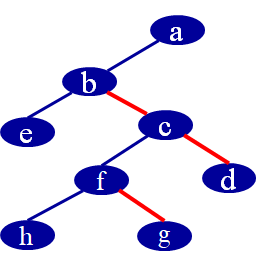
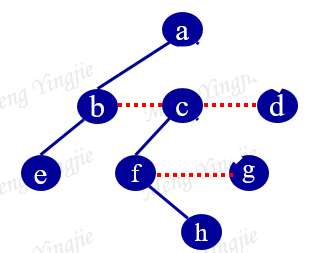
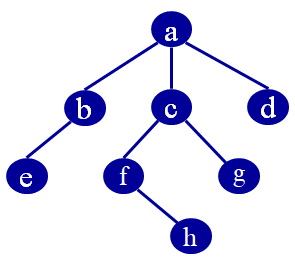
* 树、森林与二叉树的相互转化：

①树转为二叉树：

原理：根据树的孩子兄弟存储策略演化而来

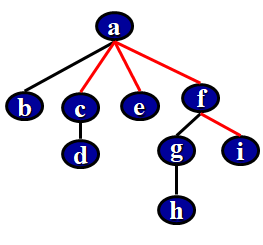
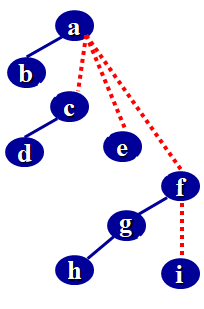
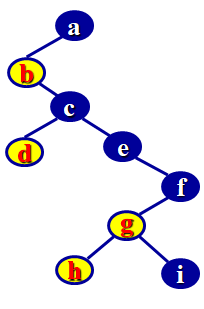
转换规则：

1. **加线**：在各**兄弟节点**之间加一条虚连线
2. **抹线**：将所有双亲结点与除第一个左孩子以外的所有孩子的关系抹掉（**除了老大之外都不要了**）。
3. **调整**：将原来左孩子的兄弟作为它的右孩子（原有连线向左倾斜，新加连线向右倾斜）。



②二叉树转为树：

1. **加线**：若某结点x是双亲结点的左孩子，则将该结点的右孩子以及当且仅当连续地沿着此右孩子的右链不断搜索到的所有右孩子都与x的双亲结点连接起来。
2. **抹线**：抹掉所有结点与所有右孩子的关系（所有结点与右孩子断绝父子关系）。
3. **调整**：原来沿着x的右链搜索到的所有结点变成x的兄弟。



③森林转为二叉树：

1. 树转换：将每一棵树转为二叉树。
2. 二叉树连接：依据森林中树的次序，依次将转换得来的二叉树，后一棵作为前一棵二叉树的根结点的右子树。

④ 二叉树还原为森林：

1. 抹线：沿着二叉树的根结点的右子树的右链不断搜索到的结点之间的连线全部抹去。
2. 还原：将得到的每一个二叉树还原为树。

* 二叉树的前、中、后三种遍历方法：

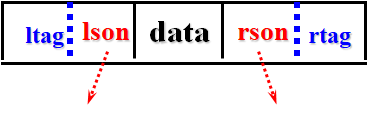
①先根遍历：a．访问根结点 b．遍历左子树 c．遍历右子树

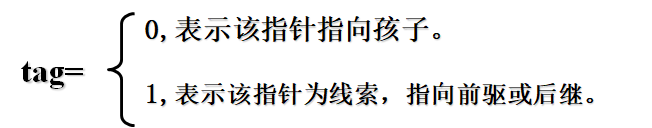
②中根遍历：a．遍历左子树 b．访问根结点 c．遍历右子树

③后根遍历：a．遍历左子树 b．遍历右子树 c．访问根结点

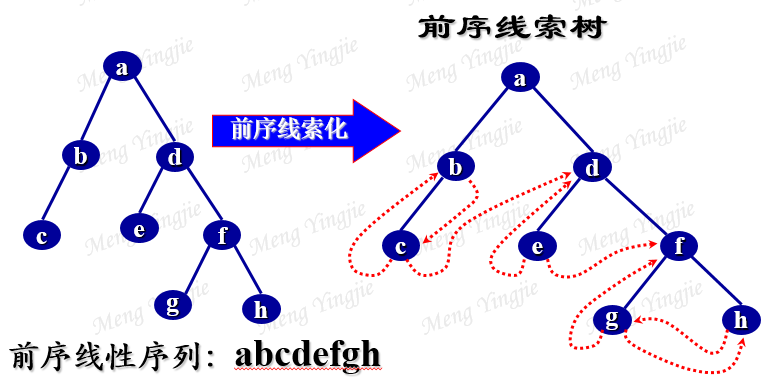
* 二叉树的前、中、后三种遍历方法的线索化过程：

1. 加线索：通过加标志位区分线索和孩子。先输出遍历结果再根据结果加线索。

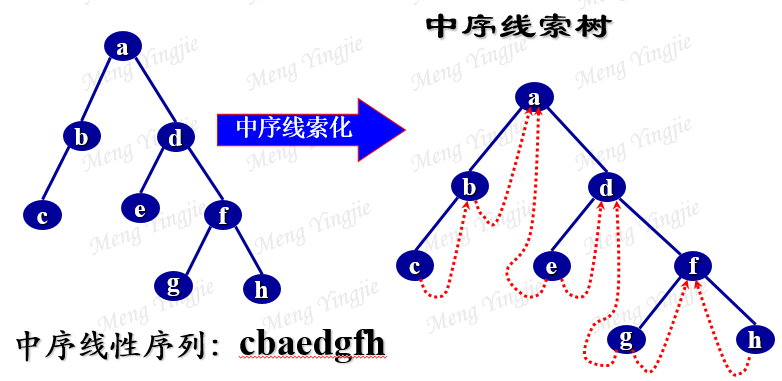




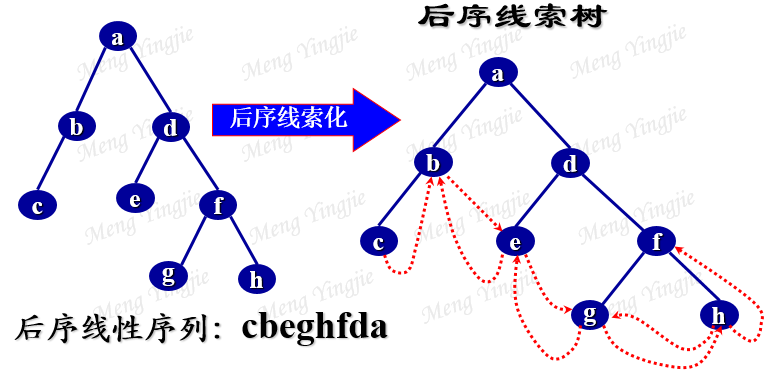
1. 三种遍历的线索化过程示例：
2. 前序遍历：



1. 中序遍历：

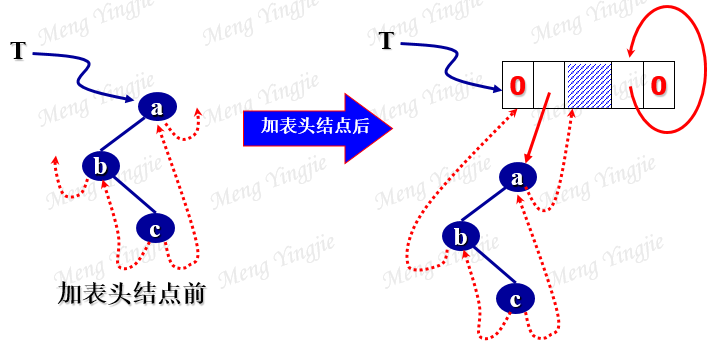


1. 后序遍历：



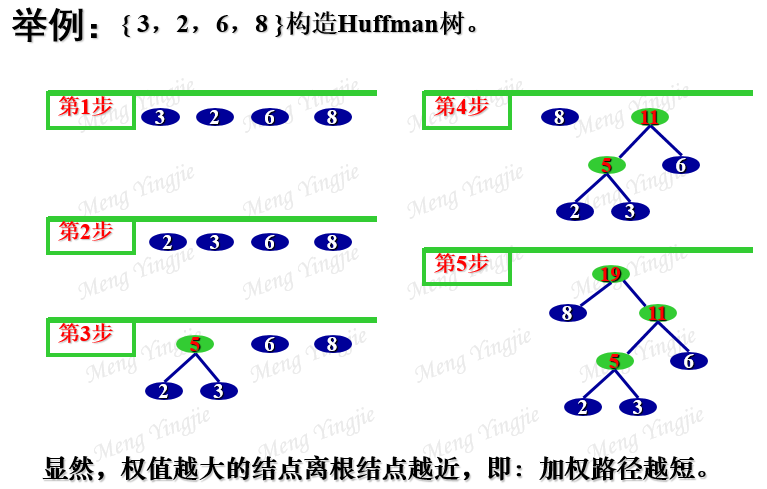
1. 线索树的改进：

可以将链表中的表头结点推广到线索树中。即可以给线索树增加一个表头结点。以中序线索树为例：

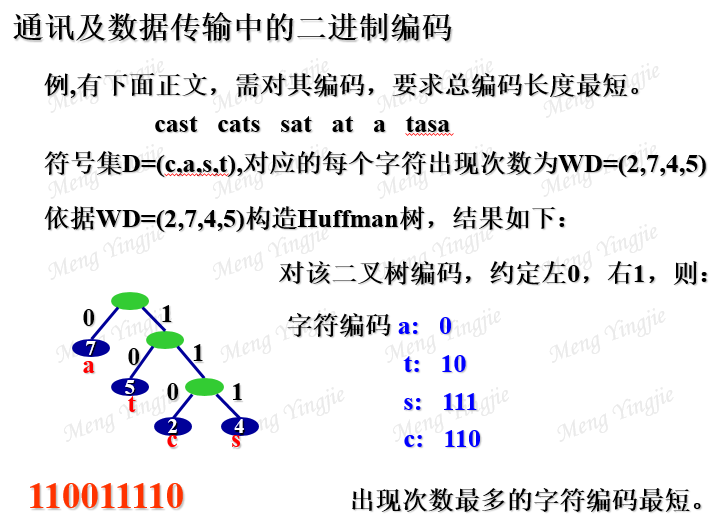


* 二叉排序树的构造：将S插入二叉排序树T中，若T为空则S直接作为T的根；若不为空则与T进行比较，如果S小于T则进入T的左子树，否则进入右子树。
* 霍夫曼树的构造及编码：

1. 构造：

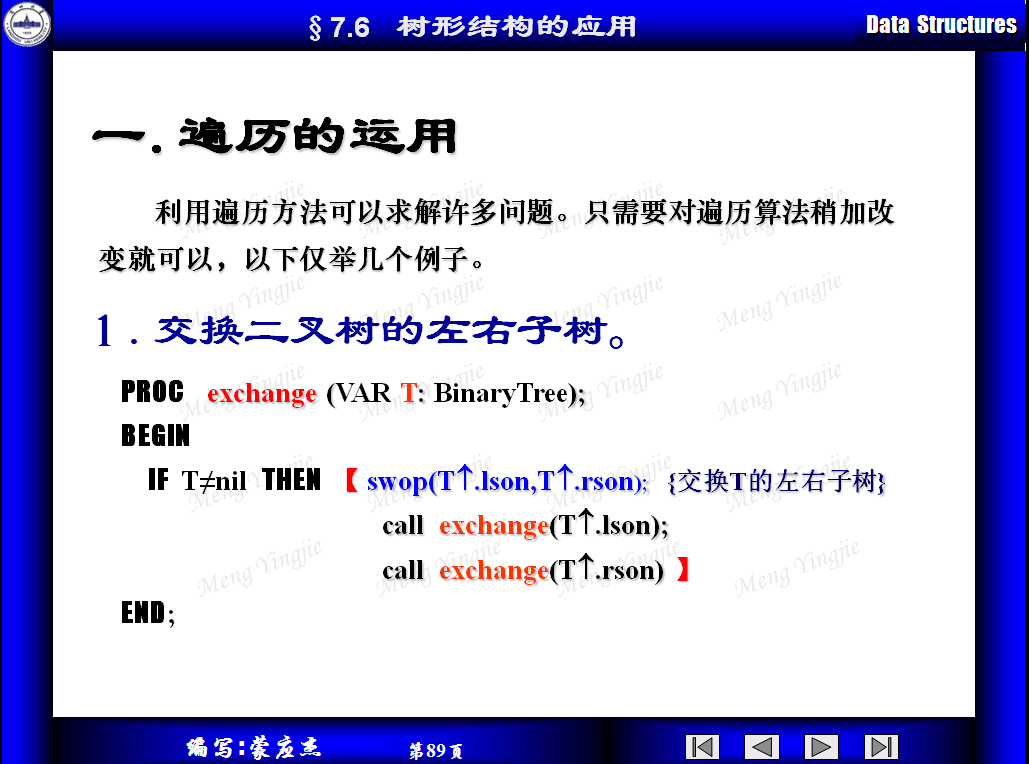


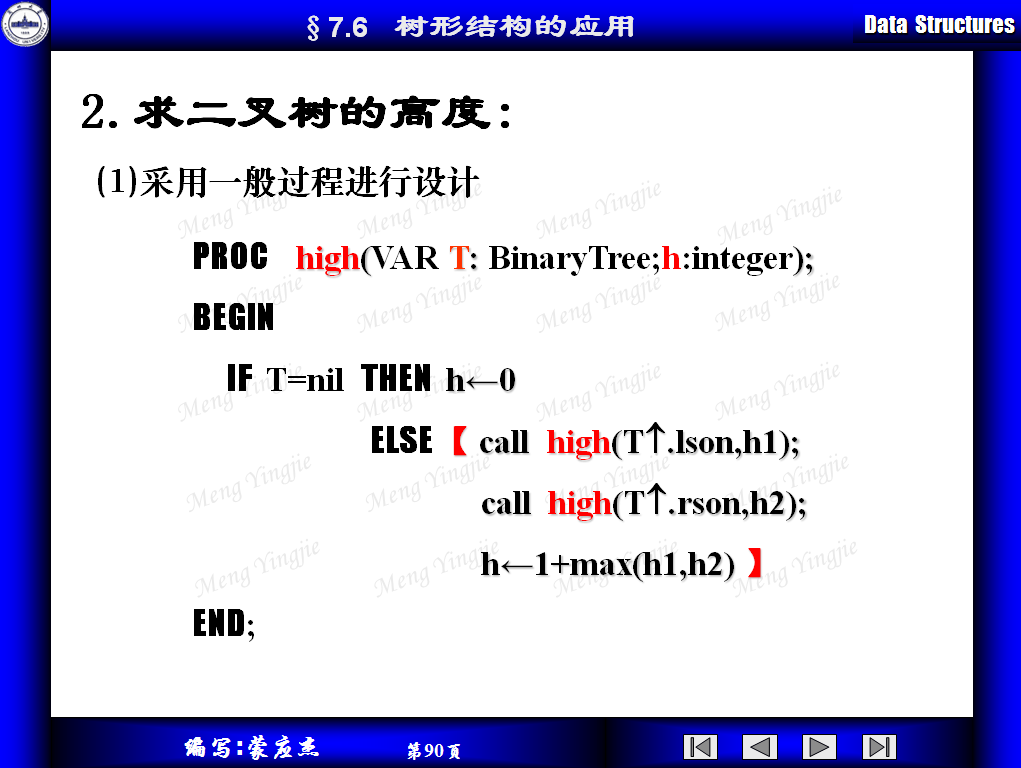
1. 编码：



***第三层面：***

* 遍历方法的应用（交换左右子树，求二叉树高度，统计二叉树叶子结点的个数等）







第八章

***第一层面：***

* 图：由n（n≥1）个结点v1，v2…vn构成的数据G称为图，若结点集V={v1，v2…vn}上定义的称为后继的关系E是非自反的。可以表示为G={V，E}V称为顶点集，E称为边集。
* 图的遍历：给出图G和其中的**任意一个顶点v0**，从v0出发**系统地访问**G中**所有的顶点**，且每个顶点**仅被访问一次**，这一过程称为图的遍历，或遍历图。需要**辅助数组**记录该结点是否被访问过。
* 图的遍历规律：

①深度优先搜索(DFS,depth-first search)：搜索中，结点扩展的次序**向某一个分支纵深推进**，到底后再回溯。

②广度优先搜索(BFS,breadth-first search)：搜索中，对**所在层次**的所有结点逐个**依次**进行**扩展**后，再推进到下一个层次进行扩展。

* 最小生成树：可系统访问所有顶点，图的所有顶点再加上遍历过程中经历的边所构成的子图实际上是一棵树，我们称做生成树。**边的权值之和最小**的生成树称为最小生成树。
* AOV-网：若在有向图G中，**顶点表示活动或任务**，**有向边表示活动或任务之间的有限关系**，则此有向图称为**顶点表示活动**的网络（AOV-网，Activity On Vertex Network）。
* AOE-网：若在带权有向图中**顶点表示事件**，**有向边表示活动**，**权表示活动持续的时间**，则此有向图称为**边表示活动**的网络（AOE-网，Activity On Edge Network）。
* 关键路径：任务计划作业图上的**需要时间最长**的路径（可有多条）。它决定完成总任务的时间。
* 拓扑排序：**寻找拓扑序列**的有效手段，就是进行拓扑排序。

***第二层面：***

* 图的深度、广度遍历原理和方法：

1. 深度策略：

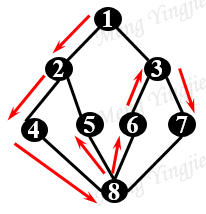
①访问出发顶点v0，

②选择一个v0邻接到的未被访问过的结点u，

③从u开始进行深度搜索。

如果从任一顶点出发再也无法抵达一个未曾访问过的顶点，则本次搜索就算结束。

回退策略：每当抵达一个其所有相邻顶点均已被访问过的顶点x时，就回退到最后访问过的那个顶点——该顶点有一个与其邻接的未曾访问过的顶点y，并从y再开始进行深度遍历。



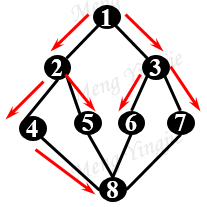
访问结果：v1、v2、v4、v8、v5、v6、v3、v7

1. 广度策略：

②访问出发顶点v0。

②访问v0邻接到的所有未被访问过的结点v1 , v2 , … , vt 。

③如此进行下去，直到无法找到未被访问的结点时，则本次搜索就算结束。



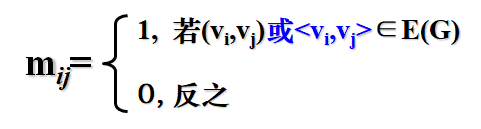
访问结果：v1、v2、v3、v4、v5、v6、v7、v8

* 图的存储方法：

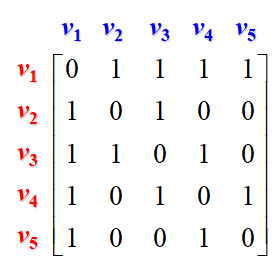
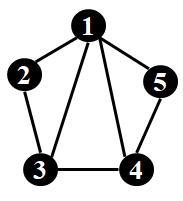
①邻接矩阵：表示顶点之间邻接关系的矩阵称邻接矩阵(adjacency matrix)。

1. 图非网时：

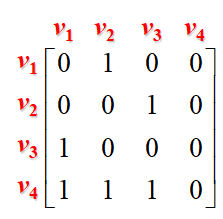
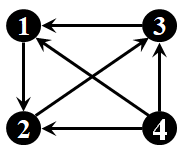
设图有n≥1个顶点，则图G的邻接矩阵定义为n阶方阵M，M满足下列性质：



对于无向图：邻接矩阵为对称矩阵、矩阵第i行之和是顶点vi的度

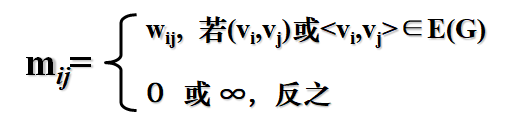


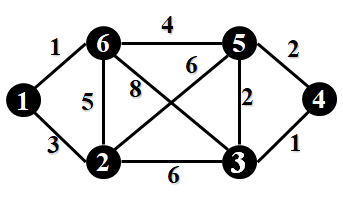
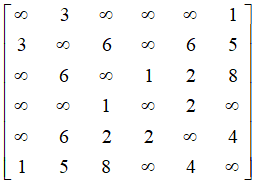
对于有向图：邻接矩阵为非对称矩阵、矩阵第i行之和是顶点vi的出度、矩阵第i列之和是顶点vi的入度



1. 网的邻接矩阵：

设图有n≥1个顶点，则图G的邻接矩阵定义为n阶方阵M，M满足下列性质：

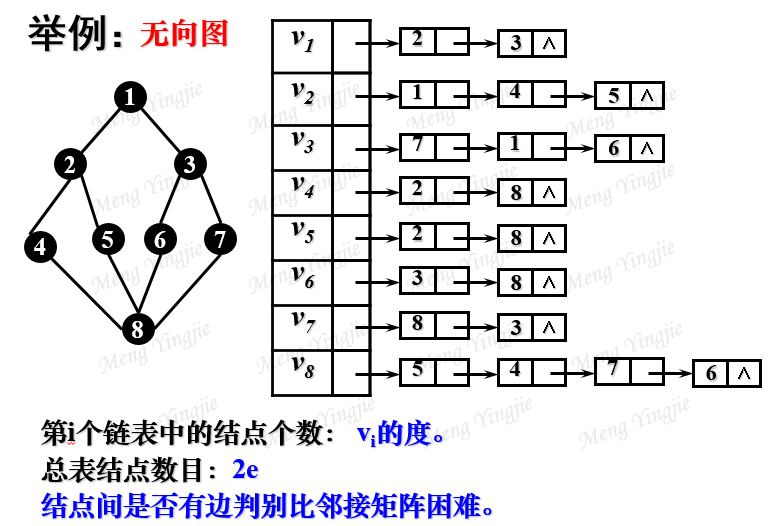




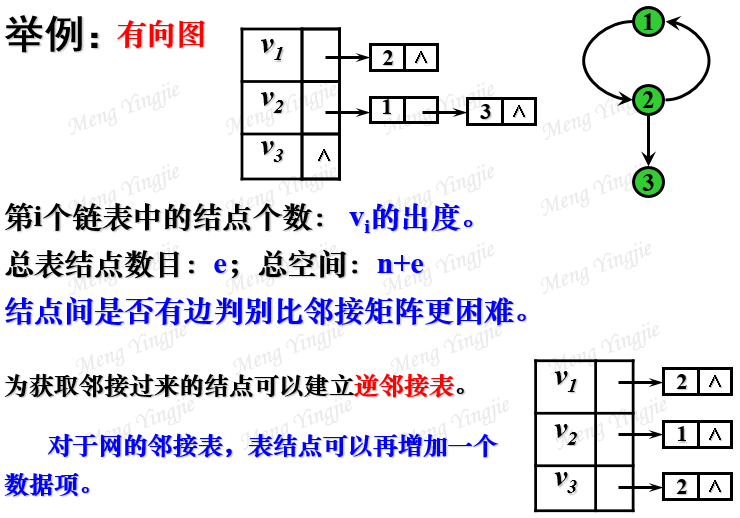
②领接表：在这种表示中，对图的每个顶点建立一个链表，也称邻接链表表示。

即有n个结点时有n个链表，第i个链表中的结点：

在无向图中：是与vi邻接的所有结点的收集。

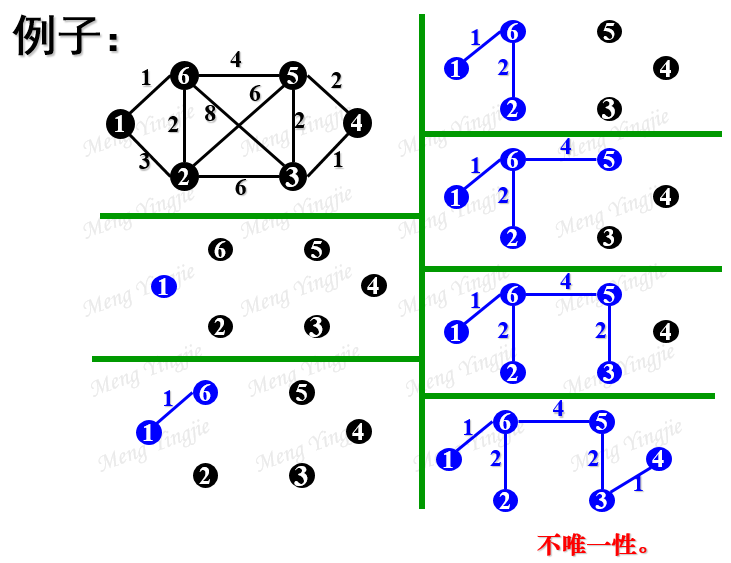


在有向图中：是以vi为始点的所有终点的收集。



* 最小生成树的构造：

1. Prim算法：设V={1,2…n}是图G的顶点集合，Prim算法由一个初值为{v0}的集合开始，它每次生成一条边，逐渐长成一个最小代价的生成树。

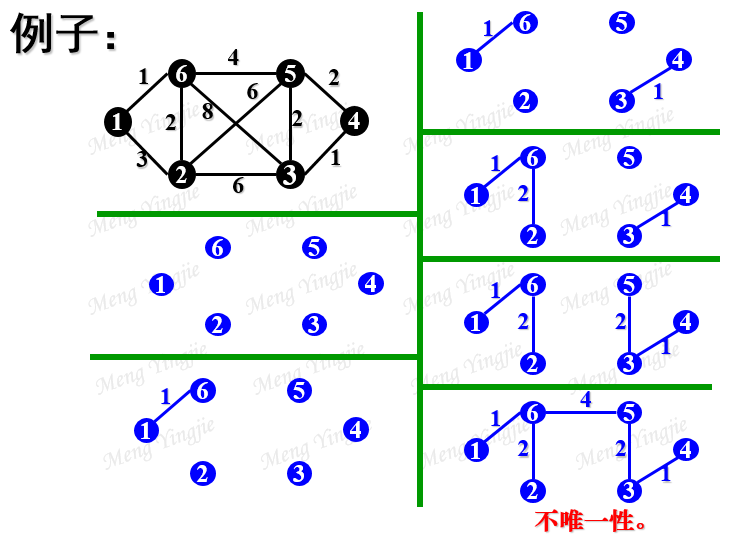


1. Kruskal算法：对于图G={V,E}该算法初始化从T={V,**Æ**}开始此时生成树是由n个连通分量（即n个孤立顶点）组成的一个子图—目标使得n个连通分量逐渐成为一个连通分量，具体做法：

按照权值递增的顺序逐个考虑E中的每条边：

①若该边连通了在两个不同连通分量中的顶点，则将该边填加到T中。

②重复①，一旦T中包含了n-1条边，则终止运算。



第九章

***第一层面：***

* 排序：设含有n个记录的集合为R={r1,r2…rn}，其对应的关键字集合为K={k1,k2…kn}，**给定关系α**，按照关系α针对关键字集合K**对R进行运算**，使得R有如下序列：{**ra1, ra2 ,…, ran**},我们将这个操作过程称为排序。本质是**在关系α下寻找R的一种排列**的过程。
* 排序的分类：依据排序过程中**数据所处位置**（内存、外部存储器），可以将排序分为内部排序和外部排序

内部排序：将**整个**待排序的文件**装入内存**，并在其中进行**排序**,这种排序过程称为内(部)排序(internal sorting)。

外部排序：由于数据规模过大，排序过程**不能在内存中一次完成**，需要**不断进行内外存数据交换**才能完成排序，这样的排序过程称外(部)排序(external sorting)。

* 排序的稳定性：在排序关系下，假设排序前ri在rj之前，**排序之后领先关系不变**，则称此排序过程和排序方法是稳定的，否则是不稳定的。
* 堆：设L是长度为n的表，当1≤i≤⌊n/2⌋时，其数据元素满足：**L(i)≤L(2i)且L(i)≤L(2i+1)**，则称L是一个堆。其中将L(1)称为堆顶，L(n)称为堆底。

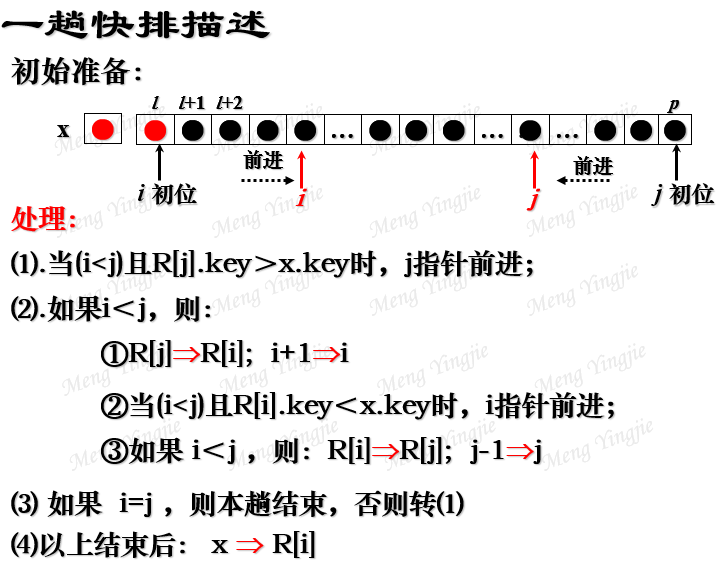
***第二层面：***

* 一趟排序的基本原理：

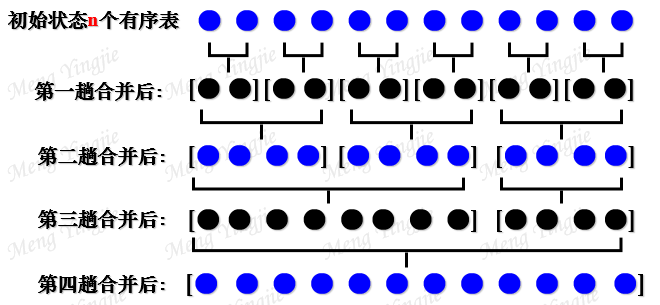
①直接插入：将要排序的源文件F=R1,R2...Rn视为两部分 F’=R1,F”=R2...Rn。对F’和F”重复如下工作：

1. 从F”中取出一个记录Ri,并将Ri从F”中删除。
2. b.将Ri插入到F’并使F’线性有序。

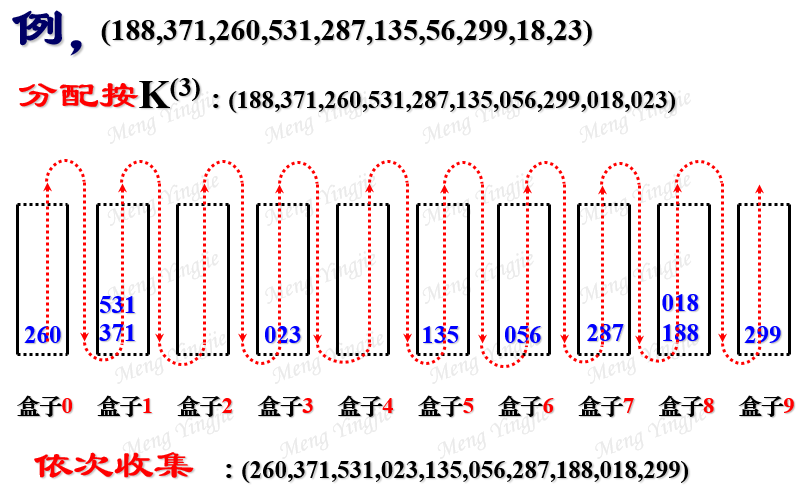
②快速排序：在待排序的n个记录中抽取一个记录r作为轴心元素，以r为标准将所有记录分为两组，第一组中各记录的关键字都小于r的关键字，第二组中个记录的关键字都大于r的关键字，并把r排在这两组中间。



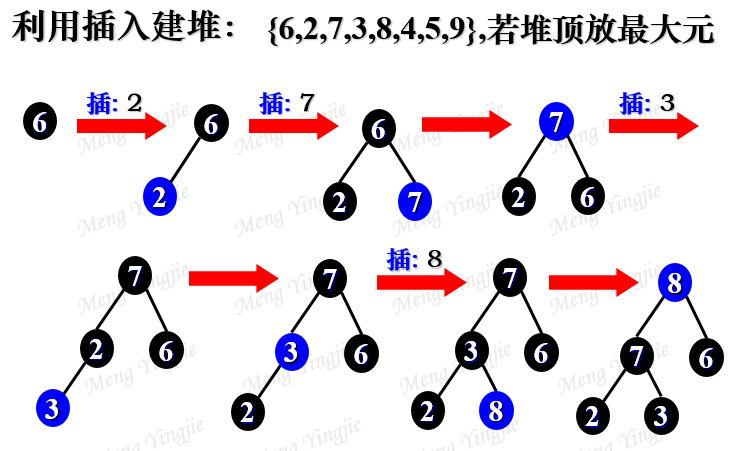
③ 二路归并：将两个子文件进行合并，得到n/2个部分有序的较大子文件，每个子文件含有两个记录，再将这些子文件合并，如此反复，直到最后合并成一个文件时，排序就完成了。

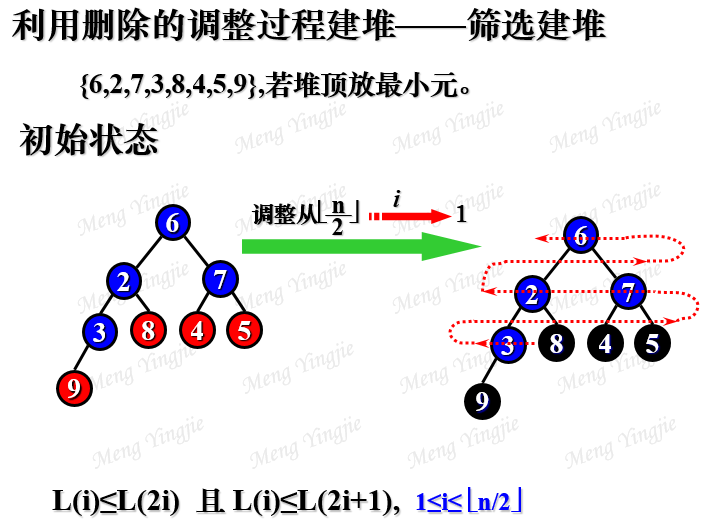


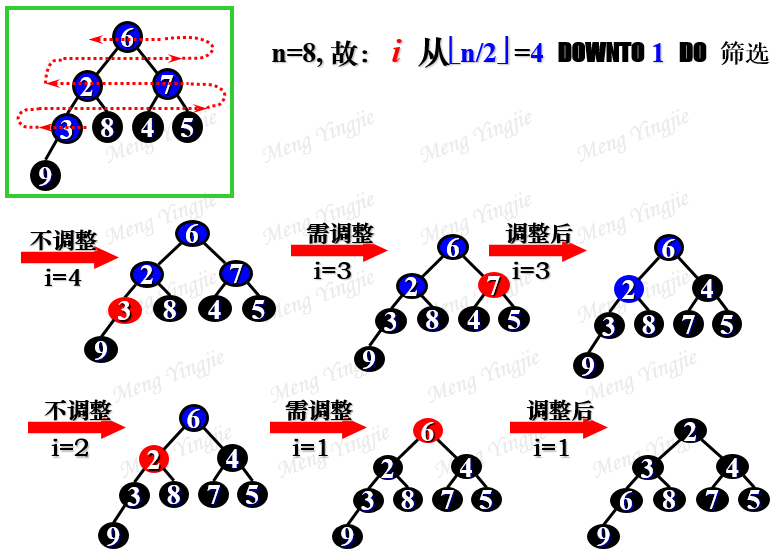
④ 基数（桶）排序：从个位开始放到0至9的盒子中，直到排完最高位结束。



* 堆的构造

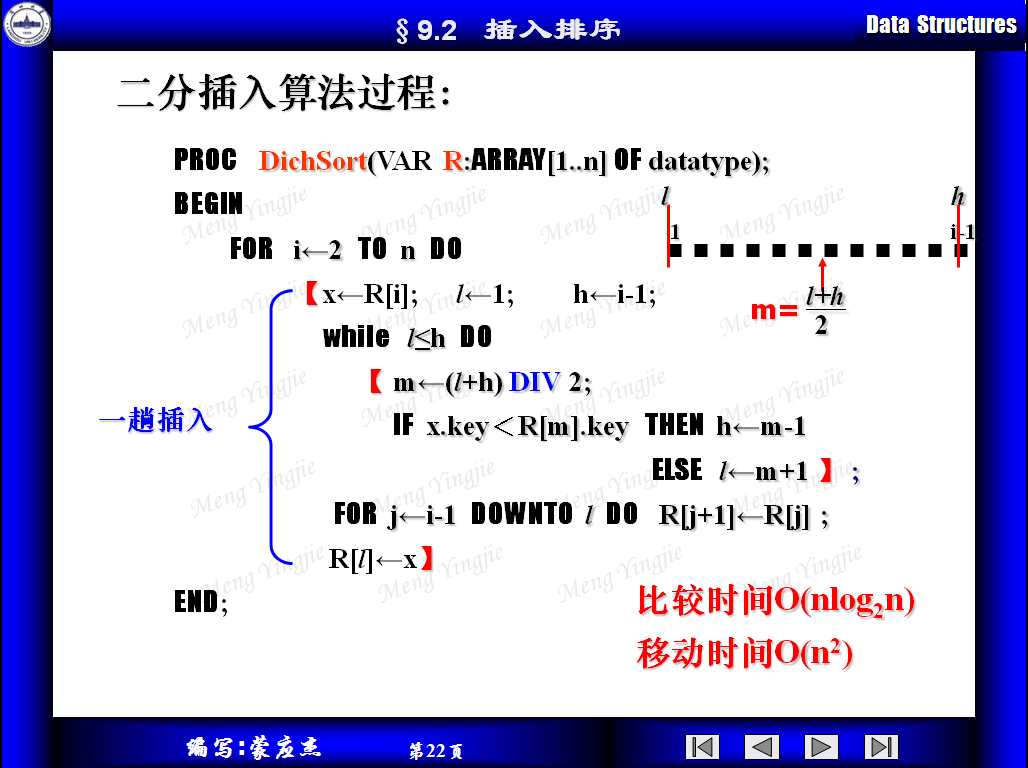






***第三层面：***

* 二分/折半插入排序：



第十章

***第一层面：***

* 检索：在给定数据结构中**查找满足某种条件的数据元素**(或结点、记录)的过程。
* 检索的分类：

①按性质分：

1. 基于关键字的检索：在给定的结构中**找出关键字**等于指定值的结点，即按关键字检索。
2. 基于属性的检索：在给定结构中**找出某属性值**等于指定值的结点。

②根据检索对象的**组织关系**和检索对象的**存储组织**模式，检索算法有三类：

1. **顺序表和线性表方法**；
2. **直接访问法(散列方法)**;
3. **树索引方法**。

* 平均检索长度：检索过程中对关键字(或属性)要执行的**平均运算次数**。
* AVL-树（平衡二叉树）：

①一棵**空二叉树**是AVL-树；

②若T是一棵非空二叉树, 其任何结点的**左、右子树的高度相差不超过1**,则T是AVL-树。

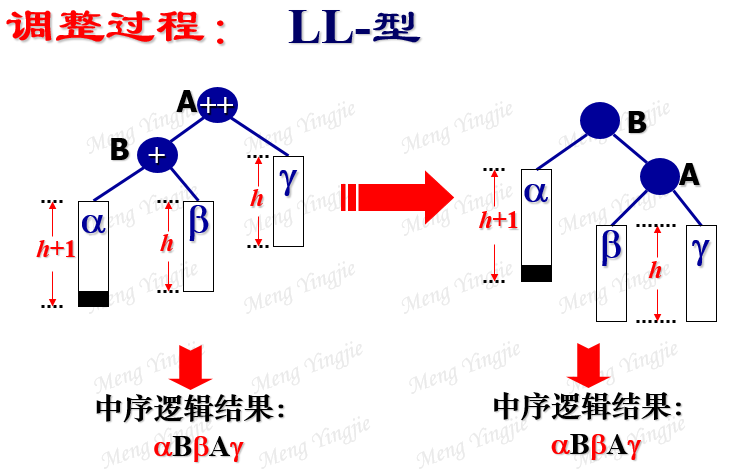
* 散列表：把用**散列法组织存储**的线性表称为散列表。

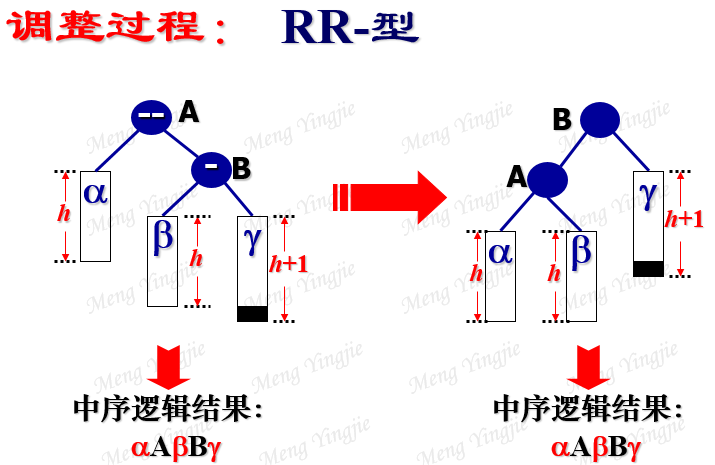
散列法的基本思想：将关键字看成一个变量，通过一定的函数关系，将函数值解释为存储地址，将结点存入这样计算得到的地址单元中，检索的过程是存储过程的逆过程。也称关键字——地址转换法，或关键转换法。

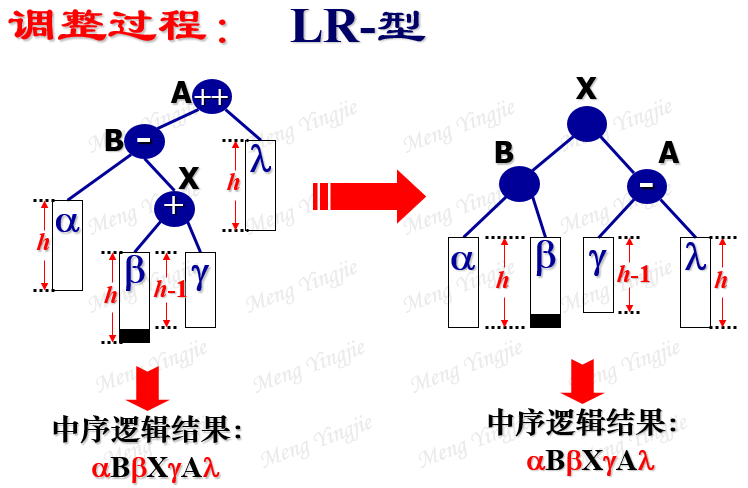
* 碰撞/冲突：依据散列函数H计算出地址，若发现此地址已经被别的结点占用, 即就是说有两个**不同的关键映射到了同一地址空间**，我们把这种现象称为碰撞(或冲突，collision)。
* 同义词：**产生碰撞的**两个(或多个)**关键字**我们称为同义词(相对于H而言)。（k1≠k2，H(k1)=H(k2)）
* 堆积：把**两个同义词子表结合**在一起的现象称为“堆积现象”或“群集现象”。

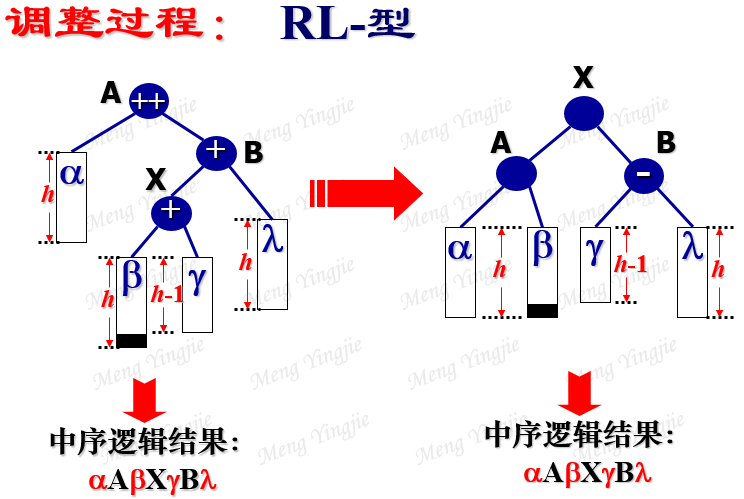
***第二层面：***

* AVL-树的构造：选择离插入或删除结点最近的不平衡点（平衡因子为±2）开始调整.









***第三层面：***

* 二分检索：



第十二章

***第一层面：***

* 文件：**为了进行存取控制、检索和修改而组织在一起**的**数据记录**的**集合**。
* 文件的逻辑组织的分类（用户角度）：

①**字符流文件**：有序的字符流序列，文件基本单位为字节或字。

②**记录文件**：数据记录的集合，文件的基本单位为记录(定长或变长记录)。

* 文件的物理组织的分类（存储结构）：

①**顺序文件**：按照数据到达的时间先后次序进行组织。按该方式（顺序结构）组织的文件称顺序文件。

②**散列文件**：按照散列方式组织文件。该方式组织的文件称散列文件。多适合于随机存取，因此也称随机文件。

③**索引文件**：组织数据时需要带一个索引表。按该结构组织的文件统称为索引文件。主要是利用树形结构组织索引。

* 静态索引结构：静态索引结构是指索引结构**在文件创建**，**初始装入记录时生成**，**一旦生成就固定下来**，在系统运行(例如插入/删除)过程中索引结构并不发生变化，只有当**文件重组时才允许改变索引结构**。
* 动态索引结构：动态索引结构是指文件创建，初始装入记录时所生成的索引结构，在**系统运行**(例如插入/删除)过程中**索引结构**本身也**能够发生改变**。
* ISAM的本质：ISAM（索引顺序存取法）是现今磁盘存取文件的常用组织方法，采用静态结构来组织索引。本质是**静态索引结构**。
* VSAM文件的本质：VSAM（虚拟存储存取法）是B+树应用的一个典型例子。其本质是**动态索引结构**。
* B-树、B+树的本质：B-树、B+树是一种**平衡的多分树**。
* B-树、B+树的作用：由于对B树的查找、插入、删除均能够保持B树的动态平衡，因此B-树及其他一些改式（例如B+树）已经成为**组织索引文件（动态索引结构）的一种标准的有效结构**，已得到了广泛的应用。
* 外部排序的基本方法：

外部排序的具体方法与数据所处的外存设备特征有关，但是最通用的方法是**合并排序**，这种方法基本上由两个阶段组成：

1. **生成初始顺串**：首先用某种有效的内排序方法对文件的各段进行排序，这种经过排序的段通常称为顺串 (或归并段)，当它们生成后即写到外存上，这样外存上就形成了许多初始归并段。
2. **合并初始顺串**：对于第一阶段生成的初始顺串使用某种归并方法(例如2-路归并)，进行多遍归并，最后形成整个文件的单一顺串。