1. **基本概念 0.3**
2. **常见数据表示存储（如三元组、链表）和运算（给定一棵树，如何变为二叉树）方法 0.5**
3. **算法设计 0.2**

**第一章（一）**

数据：1 通用：用以载荷信息的物理符号 2 计算机：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，描述现实世界某客观实体的最小数据集

数据对象：具有相同属性的数据元素的集合

数据结构：给定数据对象及其上面定义的操作（运算、关系）所共同构成的系统

DS研究的主要方面：数据的逻辑、存储和运算关系。

常见的逻辑关系：线性关系、树形关系、图关系

常见的存储关系：顺序关系、链接关系、散列关系、索引关系

DS的评价标准：1、作为问题参数的函数来计算时间效率，即时间复杂度

2、作为问题参数的函数来计算存储需要量，即空间复杂度

**第二章 算法概论（一）**

算法：解决某个问题的指令的有限集合，都必须符合有穷性、确定性、可行性及输入\输出问题

过程：事物发展或事物进行所经过的程序，即次序

过程的分类：根据信息处理的要求及目的性，过程可分为一般过程（关注过程）、函数过程（关注结果）

有效算法：以多项式时间为界限的算法

难解性问题：不存在以多项式时间为界限的算法的问题

算法设计的四条标准：正确性、易读性、健壮性、高效性

算法常用的描述方法：（具体的）计算机程序设计语言、自然语言（具有二义性）、PDL语言、流程图

**第三章 线性表（一、三）**

**一、**

线性表：是n >= 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，一般放置于链表的最前面，但该结点不用存储数据元素

存储密度：数据本身所占存储量 / 整个结构所占存储量

**三、**

链表的插入：单链表：为数据元素b申请空间，并完成数据存储；以数据元素a为依据获取插入位置；插入。 两个指针P, SAVE , S存储b

TYPE pointer = ↑node；

node = RECORD

data : datatype;

next : pointer；

END

link = pointer;

链表的删除：注意结点回收，P, SAVE两指针

**第四章 栈和队列（一、三）**

**一、**

栈：是一个下限为常数，上限可变化的向量（或者反之）

队列: 是一个下限和上限只能增加而不能减少的向量（或者反之）

栈与线性表异同:

  同：栈、线性表的数据对象及其数据元素间逻辑关系相同，故它们都属于线性结构； 异： 栈的运算是线性表的一个子集,而且这个子集还要加以约束,即若限定线性表的运算只有插入和删除那么可以认为栈是只允许在表的一端进行插入和删除。

队列与线性表异同:

同：队列、线性表的数据对象及其数据元素间逻辑关系相同，故它们都属于线性结构；

异：队列的运算是线性表的一个子集,而且这个子集还要加以约束,即若限定线性表的运算只有插入和删除那么可以认为队列只允许在表的一端进行插入，在另外一端进行删除。

假溢出：存储空间未满，由于存储方法或管理模式导致的数据元素无法存入的一种现象。

**三、**

循环队列的入列：一维数组定义，空间编址范围m..n。front 不存元素，rear存元素

VAR CQ : Array[m..n] OF datatype;

front, rear : integer;

PROC  AddQ ( CQ, x, front, rear)

｛front, rear 分别为队列CQ首尾指针，数据元素x入队｝

BEGIN rear ← rear+ 1

IF rear = n + 1 THEN rear ← m; //结束的地方，rear移到开始

IF rear = front THEN write( "queue full" )

ELSE CQ[rear] ← x;

END;

**第五章 串（一）**

串：一个由零个或多个字符组成的有穷序列

串的模式匹配：把从目标串中查找模式串的运算过程称为串的模式匹配。

**第六章 数组和广义表（一、二）**

**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：在一个矩阵中和某元素比较而言,不相同的元素很少时,我们称此矩阵为稀疏矩阵（是稀疏数组的典型例子，零元素分布无规律）**

**广义表：是零个或多个原子或子表所组成的有限序列**

**二、**

**稀疏矩阵的存储**

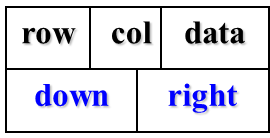
**三元组：**

**1、一维数组表示，抽象为线性表，每个元素由 row col data构成。线性表**

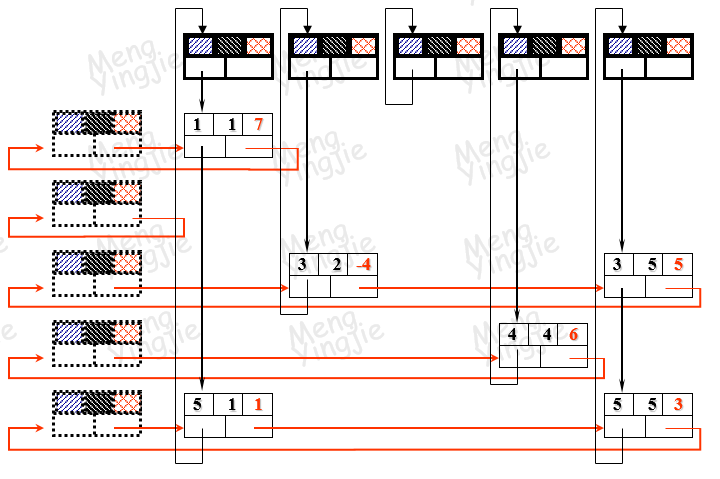
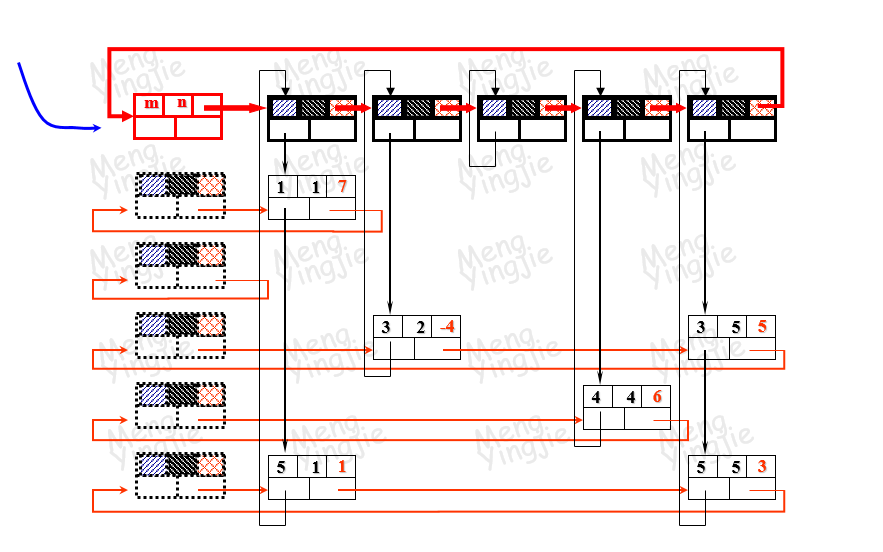
**row col data**

**1 1 1 7**

**2 2 4 4**

**3 … … …**

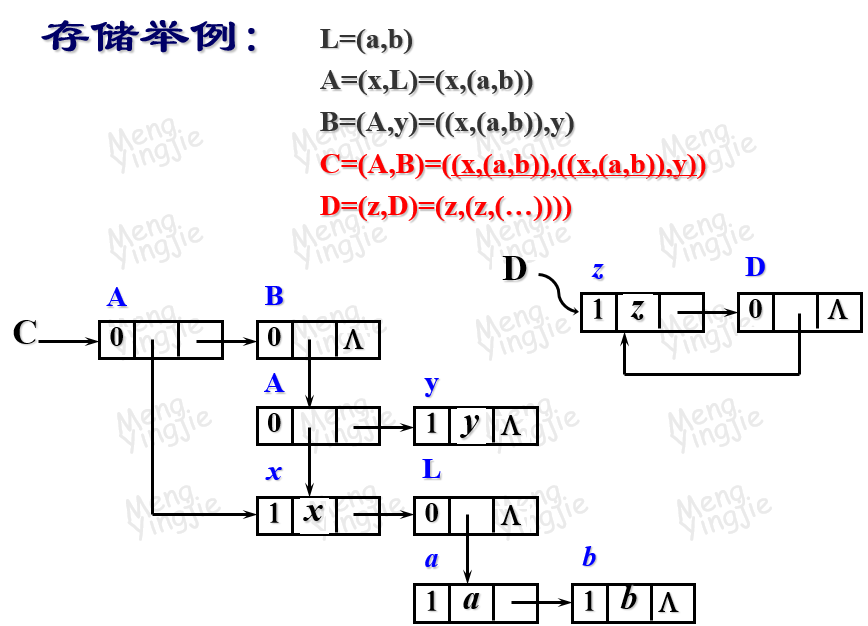
**2、二维数组表示，**A ( 0..p, 1..3 )，其中p为非零元个数，在最顶端可加入总的行和列，data存总的元素数量。

**十字链表：每一行和列建立循环链表**

**对表头结点再建立循环链表，再加入表头结点的表头结点，只链接最上面的表头。**

**广义表定长结点的存储：**

1. **顺序存储：n维数组**
2. **链表存储：atom info link atom = 0，此结点为子表；atom = 1，结点为原子。**

****

**第七章 树形结构 （一、二、三）**

**一、**

**1、树：**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。

2、二叉树：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。

3、遍历：对给定的数据结构，系统地访问该结构的每个结点，且每个结点只访问一次

4、对二叉树 \ 树的遍历：对于给定的二叉树 \ 树，系统地访问二叉树 \ 树中的每个结点，且每个结点仅被访问一次的操作过程

5、二叉树的遍历规则：层次策略：自下而上（从左到右…），深度策略：根节点左右子树访问次序不同，共六种，其中有先序、中序、后序。先序：访问根结点，遍历左子树，遍历右子树。

6、线索：将二叉树的空指针用于表示某线性关系下前驱或后继时，这种指针叫线索

7、线索化：给二叉树加线索的过程

8、线索树：带线索的二叉树

9、二叉排序树：空二叉树；或者是满足如下性质的二叉树：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；若它的右子树非空，则左子树上所有结点的值均大于根结点的值

10、霍夫曼树（最优二叉树）：按照霍夫曼算法构建的具有最小加权路径长度的二叉树

**二、**

1、树、森林、二叉树的相互转换

树转为二叉树：加线：左右孩子相连 抹线：将根节点下从右到左抹线 调整

二叉树转为树：加线：若x是父节点的左孩子，将沿x右孩子的所有结点和x的父节点相连，直到没有右孩子 抹线：抹去所有结点和其右孩子的连线 调整

森林转为二叉树：i）每个树转为二叉树 ii）二叉树再链接，依据森林中树的次序，后一棵作为前一颗二叉树的根节点的右子树

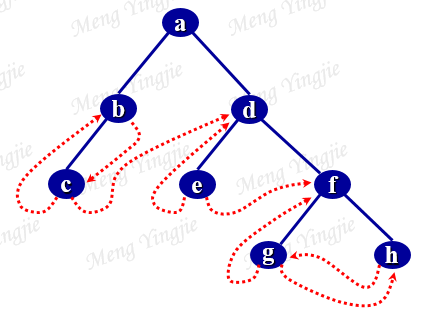
二叉树还原为森林：i）抹线：根节点和其右孩子 ii）二叉树还原为树

2、二叉树的遍历，前序、中序、后序

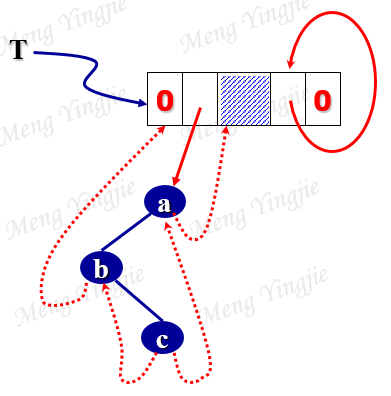
1. **二叉树线索化：一般空的左指针指向直接前驱、空的右指针指向直接后继，前驱和后继指的是遍历得到的顺序的，并非二叉树的父、子结点，通过tag表明是否为空。tag = 0指孩子，tag = 1为线索**

**ltag lson data rson rtag 改进加入表头结点**

**按中序线索化：利用遍历，把访问改为线索化，使用递归无法记录前一个结点信息。使用非递归。**

****前序线性序列：abcdefgh

**增加改进结点后，中序为例**

****

1. **霍夫曼树的构造**

**给定一组权值，从小到大排序，取出最小的两个构成二叉树，再将形成的根节点再次插入进行排序，重复。必须是左边或右边小，另一边更大。（和二叉排序树的区别，排序树：左子树小，右边大）**

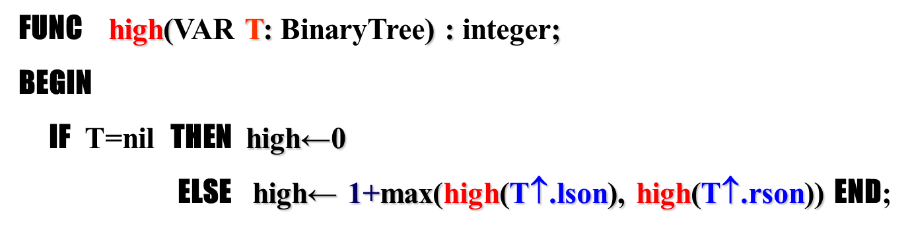
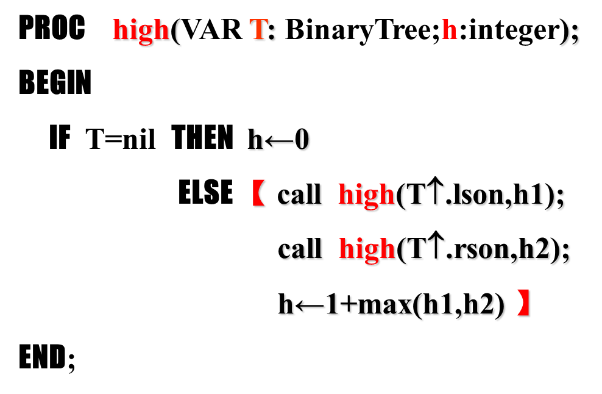
1. **霍夫曼树的编码：对一串单词，构造符号集D，计算每个字母的出现次数，用出现次数构造最优二叉树，边，和左子树相连边为0，右子树为1。**
2. **二叉排序树的构造：左子树小，右子树大。按照中序遍历二叉排序树，得到线性有序序列。按照插入一个结点的方法构造，比根节点大，进入右子树，小，进入左子树。**

**三、**

**遍历方法的应用**

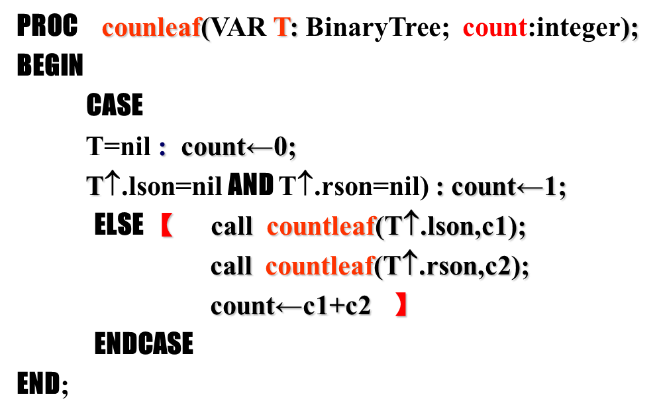
1. **交换左右子树的二叉树：每个结点的左右子树都要交换**

****

1. **求二叉树的高度**

**采用函数过程最恰当。**

1. **统计二叉树的叶子结点个数**

****

**第八章 图结构（一、二、三）**

**一、**

**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。可表示为**G=(V,E)，V称为顶点集，E称为边集。

**图的遍历：图G和其中任意一个结点v0，从v0出发系统地访问G中所有的顶点，且每个顶点只访问一次的过程**

**最小生成树：**可系统访问图的所有顶点，再加上遍历过程中经历的边，构成的使边的权值之和最小的一棵树（子图）为最小生成树

**AOV网：**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优先关系,此有向图称为顶点表示活动的网络，即AOV网

**AOE网：带权有向图，顶点表示事件，有向边表示活动，权值表示活动持续事件，这样的带权有向图为边表示活动的网络，及AOE网。（activity on edge network）**

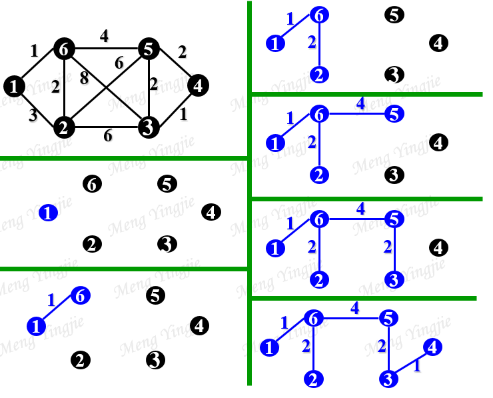
**关键路径：任务计划作业图上的需要时间最长的路径（可有多条），它决定总任务完成时间。**

**拓扑排序：寻找它拓扑序列的过程**

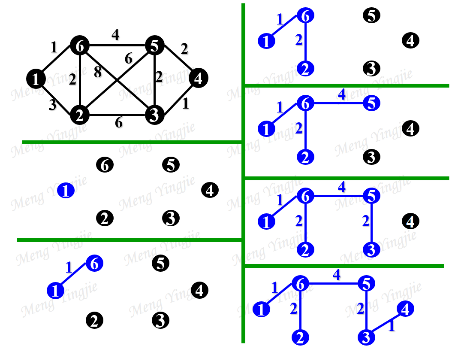
**二、**

**1、图的DFS、BFS方法和思想**

**2、图的存储表示：a）邻接矩阵：图非网，无向图时，矩阵内用0或1表示，0没有这个边。有向图，<vi，vj>，i行，j列，同样0和1表示。 b）邻接表：对图的每一个顶点建立一个链表。无向图：链接与vi邻接的所有结点；有向图：链接以vi为起点的所有结点。**

**3、构造最小生成树的方法和原理：**

**a）Prim（普利姆）算法：取图中的一点v0，加入集合U中（ U = {v0} ），从v0开始，在图剩下的点中，寻找从U中的点到这些点权值最小的边的结点加入U中，直到所有结点加入到了U中。**

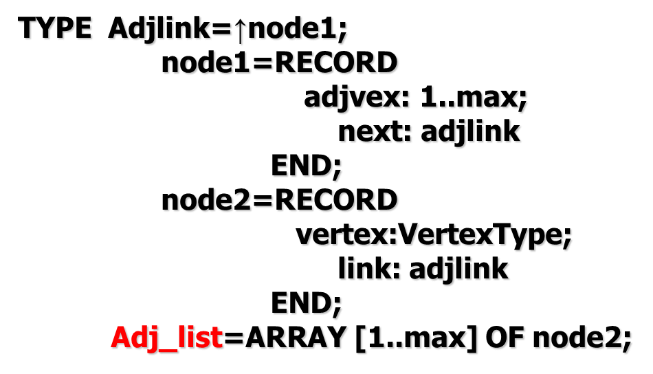
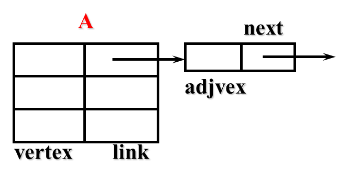
****

**b）Kruskal算法（克鲁斯卡尔）：去掉图中的所有边，从权值最小的边开始考虑，使边能把两个分开的连通分量相连。**

**三、**

**邻接表的建立：**

**类型定义：**

****

**vertex 存储结点数据**

**有向图： 无向图：**

**第九章 排序（一、二、三）**

**一、**

**排序：**设有n个记录的集合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: (r1, r2, …, rn)，的操作过程

**基于数据规模排序分为：内排序、外排序**

**内部排序法：整个待排序文件装入内存来排序**

**外部排序法：需要不断进行内外存数据交换才能完成的排序**

**排序的稳定性：在排序的关系下，若排序前ri在rj前，排序后，领先关系不变，则此排序过程和排序方法是稳定的**

**堆：设L是长度n的表，当1 <= i <= [ n\2 ]，其数据元素满足L(i) <= L(2i)，且L(i) <= L(2i + 1)，则称L是一个堆**

**二、**

**基于一趟排序**

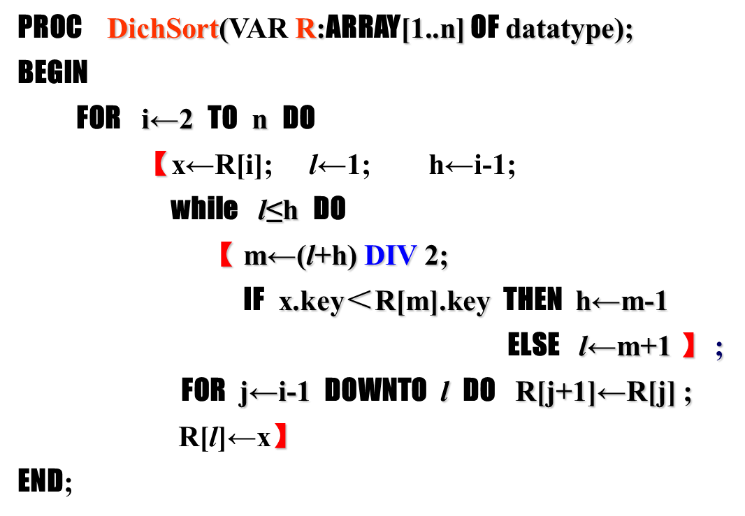
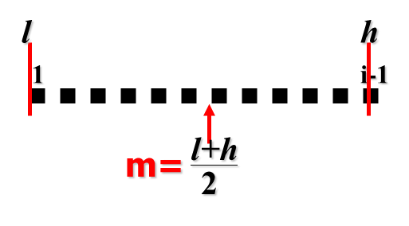
1. **直接插入：选取元素x，x会依次向后移动，若x大于x前一个元素，将x放到这个元素上，这个元素向后比较，若小于比较元素，元素依次向后移动。在一趟排序中，排序了的元素有序**
2. **快速排序：**在待排序的n个记录中任取一个记录r(例如就取第1个)，作为轴心元素；以r为标准将所有记录分为两组，a）第1组中各记录的关键字都小于r的关键字；b）第2组中各记录的关键字都大于r的关键字；并把r排在这两组中间(最终位置)的这一过程称为一趟快排。

i，j分别从两头开始向中间。交换元素，直到i，j相等，为一趟

1. **两组合并：对已有序的文件F1, F2排序，类似于多项式相加，另设F来记录F1, F2排序结果。**
2. **基数排序：**统计小于(或大于)关键字K的个数。例如，若一个关键字K恰好大于27个别的关键字，则这个关键字所对应的记录就应排列在第28个位置上，每个记录需带一个计数器

**三、**

**二分插入（一趟插入）：**

****

**第十章 数据检索（一、二、三）**

**一、**

**检索：在给定数据对象中查找满足某种条件的数据元素的过程**

**基于检索性质的分类（检索的分类）：基于关键字的检索和基于属性的检索**

**基于数据组织的分类（基于检索对象的组织关系和存储关系）：线性结构、散列方法、树索引方法**

**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。

**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。

**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（衡量检索算法的主要标准）

**AVL树：**一棵空二叉树；或：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：**用散列法组织存储的线性表

**碰撞：**有两个不同的关键映射到了同一地址空间的现象

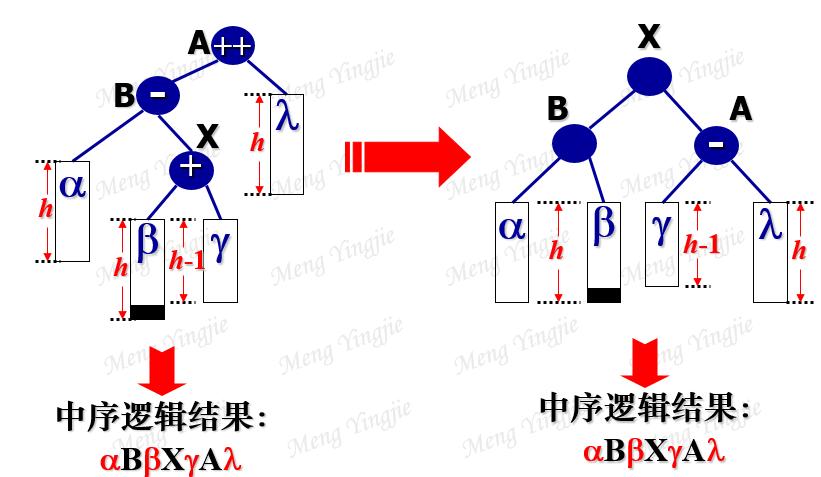
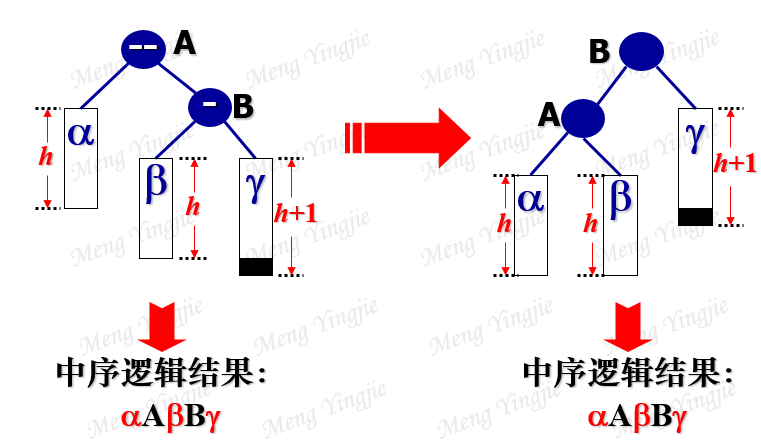
**同义词：**产生碰撞的两个(或多个)关键字称为同义词(相对于它们的散列函数H而言)。

**堆积：**两个同义词子表结合在一起的现象

**二、**

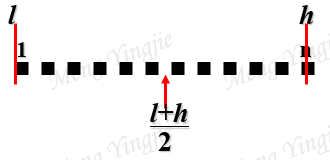
**AVL树的构造（二叉排序树的升级）：调整前后中序结果相同，**选择离插入(或删除)结点最近的不平衡结点(其平衡因子为±2)开始调整，将偏重的结点向下

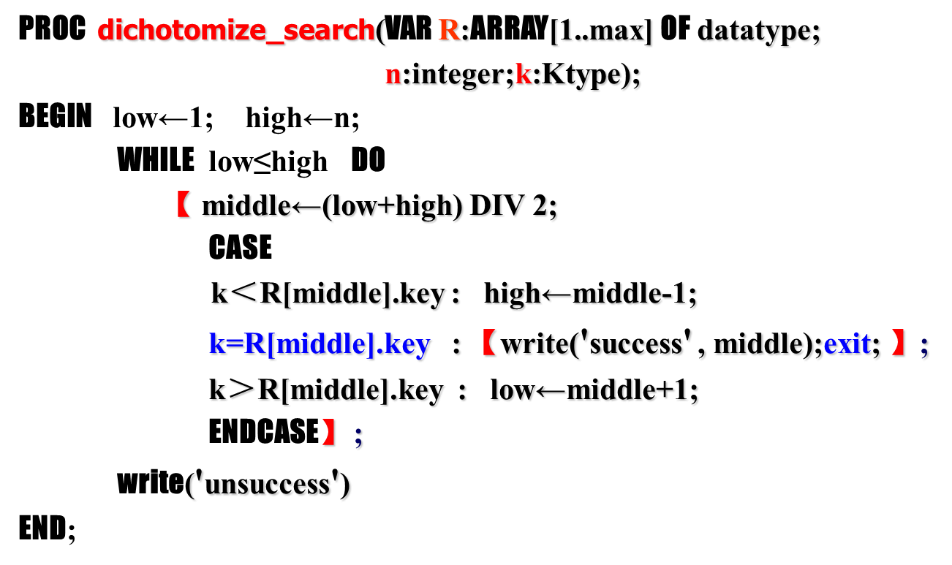
LR RL型，将偏重的x结点作为A B的根节点

****RR 和 LR 型（右子树的右子树、左子树的右子树不平衡）

**三、**

**二分检索：**先取表的中间位置的记录关键字值与所给关键字值进行比较，如果给定值比该记录的关键字值大，则给定值必在表的后半部分；在这后半部分中再取中间位置的记录进行比较，又可舍去这部分中的一半； 重复，直到找到或查完全表而查不到为止。

****需要随即获取区段首、尾及中间位置，所以待检索文件采用向量存储方法

****

**第十二章 文件结构（一）**

**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

**文件的逻辑结构（基于用户角度的分类）：分为**字符流文件，有序的字符流序列，文件基本单位为字节或字。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)

**文件基于存储结构的分类、组织：a）顺序结构，**按照数据到达的时间先后次序进行组织，**该方式组织的为顺序文件b）计算寻址结构：按照散列方式组织，称散列文件 c）带索引的结构：组织数据时需要带一个索引表，索引文件**

**动、静态索引结构：**根据系统运行期间索引结构可否动态变化

静态索引结构：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的索引结构

动态索引结构：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

B-、B+树的本质：平衡的多分树

**B-、B+树作用：组织动态索引结构**

**ISAM文件本质：静态索引结构**

**VSAM文件本质：B+树组织的动态索引结构**