第十章 数据检索

一.概论

前面较为系统地介绍了排序的一些基本的方法。

本章集中讨论非数值程序设计中的另一个重要的技术问题——检索 (search,或称为查找seek).

二.相关概念

1.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。

检索的结果只有两种:检索成功、检索失败。

举例，查字典、找人、查成绩等。

2.检索的分类

依据检索的性质检索可以分为两类：

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，即找关键字的检索。

举例是最常见的检索。

检索成功时，往往只得到一个结点即可。

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。

举例：

检索成功时，检索结果往往是一批结点。

3.检索方法的分类

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：

- 顺序表和线性表方法；
- 直接访问法(散列方法)；
- 树索引方法。

对于不同的存储方式，检索的方法是不相同的，反过来，为了提高检索速度，又常常采用某些特殊的存储结构来组织要检索的信息，例如，倒排表、字符树等。因此，在研究检索方法时，首先必须弄清楚这些方法所需的存储方式。

4.一个检索算法的特性

(1).内外有别。

分内检索和外检索。内检索是内存能够容纳全部记录的情形。

(2).静态动态。

静态检索时，表的内容不变(即一个单纯的查找过程)；动态检索时，表中的内容不断地在变动(即表有频繁地插入/删除记录的操作)。

(3).原字变字。

原字系指用原来的关键字；所谓变字是指使用经过变换过的关键字。

(4).数字文字。

指比较时用不用数字的性质。用数字的性质就是象排序算法中那样作各位数的分布计数,而不是直接对关键字进行比较,例如字符树就是使用数字性质。

5.检索算法效率的度量

在前面的各类算法中要对算法评价时，主要从时间和空间两方面来进行，对检索算法同样也从这两方面入手，但更强调时间性能。

衡量检索算法效率的主要标准是检索过程中对关键字(或称键值)要进行的平均比较次数，即平均检索长度(ASL, Average Search Length)。

$$ASL=\sum_{i=1}^{n}(C_i \times P_i),$$

其中，C_i查找第i个的比较次数,
P_i查找第i个的概率

这里的运算在大多数情况下是关键字的比较运算。P_i一般认为是等概率的(即P_i=1/n)

本节结束

一.概论

线性表的检索适应于待检索的文件呈现线性表结构组织的情况，是比较简单的的检索方法。

本节主要讨论：

- 顺序检索；
- 二分检索；
- 分块检索.

二.顺序检索(sequential search)

1.检索方法:

用给定的关键字值与线性表中各结点的相应关键字值进行逐一比较。

找到相等的则检索成功，否则检索失败。

这种方法对顺序分配或链接分配都是适应的。

该方法对于待检索文件的结点无排序要求。

2.向量存储时的算法:

```
PROC Sequential(VAR R:ARRAY[1..max] OF datatype;n:integer;k:Ktype);
BEGIN
        R[n+1].key←k;
        i←1;
        WHILE R[i].key≠k DO
            i←i+1;
        IF i≤n THEN   write('success' , i  )
                ELSE   write('unsuccess')
END;
```

有些教科书采用逆序查找(从最后一个向第1个查找)。

3.简单分析：

检索成功的比较次数： $ASL=\sum_{i=1}^{n}(i)/n=(n+1)/2$

检索不成功的比较次数： n+1.

时间复杂度:O(n)

空间复杂度:O(1)

算法优点：简单易行。

缺点：检索时间长，检索长度与表中结点数成正比。

三.二分检索(dichotomizing search)

1.检索方法:

此方法是将要检索的对象分成两部分，舍弃不包含所需要项目的那一部分，对剩下的部分再用相同的方法进行划分，直到找到所需要的项目或划分部分为空为止。

也称对分检索、折半检索。是一种很常用的方法，例如，查字典。

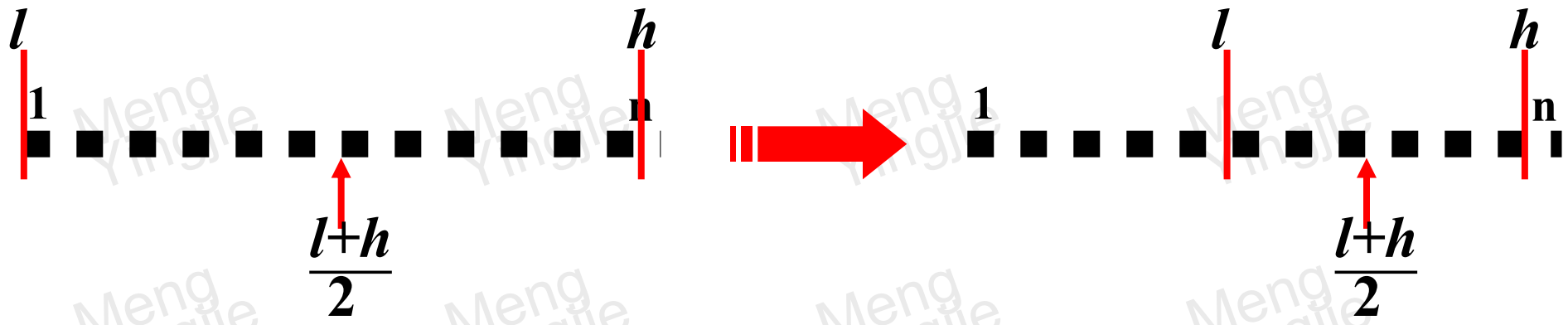
2.二分检索的基本作法:

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；

依次重复，直到找到或查完全表而查不到为止。

操作示意图：



二分检索是一种较快的检索方法，但为了能够按此过程进行，待检索的文件必须有序，另外需要随即获取区段首、尾及中间位置，所以待检索文件必须采用向量存储方法。

3.算法过程：

```
PROC dichotomize_search(VAR R:ARRAY[1..max] OF datatype;
                        n:integer;k:Ktype);
BEGIN  low←1;   high←n;
       WHILE low≤high DO
            【middle←(low+high) DIV 2;
              CASE
              k＜R[middle].key :  high←middle-1;
              k=R[middle].key  :  【write('success' , middle);exit; 】 ;
              k＞R[middle].key :  low←middle+1;
              ENDCASE】 ;
       write('unsuccess')
END;
```

4.简单分析：

设每个记录检索概率相等。

平均检索长度：$ASL=\frac{n+1}{n}\log_2(n+1)-1$

n较大时,$ASL=\log_2 n$

为换取快速检索所付出的代价是要将线性表排序；适应于一旦建立起来就很少改动而又需要经常检索的线性表。

*5.有序表的其它检索方法

(1)斐波那契(Fibonacci)检索

折半检索是用数据集的中点划分数据集的， Fibonacci检索是用该数列中的数来划分数据集的。——可使用多阶的Fibonacci数列。

(2)插值检索

数据集的划分是依据数据结点大小的期望值来作的——利用线性插值来确定期望地址i：

i=n*(K–Ke)/(Kn–Ke)

其中：n为总结点个数，K待查关键字，Ke,Kn分别为当前检索子集的最小、最大元。

四.分块检索

如果要处理的线性表既希望有较快的速度，又希望其能够动态变化，则可以采用分块检索(block search).

例如，字典正文可以看成若干块。这样字典内容的组织可以是：

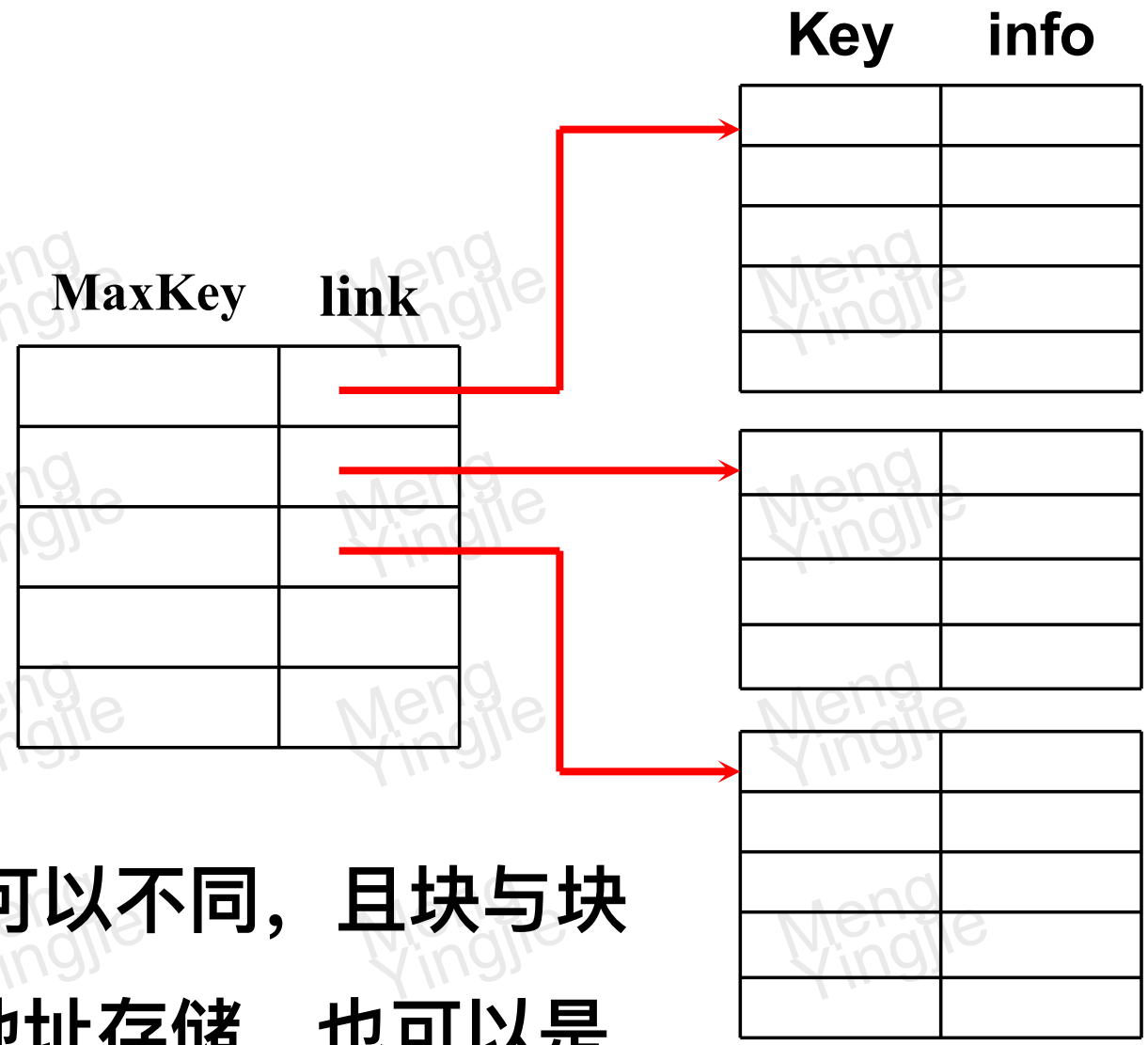
目录+字典正文

1.分块的组织：

分块检索要求把线性表分成若干块，在每一块中记录的存放是任意的，但是块与块之间必须有序，即第一块中的任一记录的关键字都小于第二块中所有记录的关键字，第二块中的任一记录的关键字都小于第三块中所有记录的关键字，……

另外要求建立一个索引表，把每块中最大(或最小)的关键字，按照块的顺序存放在一个辅助数组中，显然这个数组也是按升序排序。

组织示意图:



块与块大大小可以不同，且块与块之间的可以是连续地址存储，也可以是不连续地址，但**以不连续地址其优点体现的更为充分**。

2.检索方法：

检索时首先在索引表中进行，以便确定记录在哪一块，方法可以用顺序也可以用二分法；

当确定块以后，在块内可以进行顺序检索。

优点：既有较高的检索速度，又可以适应数据动态变化(即增/减记录)。

缺点：平繁进行增/减记录可能导致块中记录的分布很不均匀，此时检索速度将会下降。另外需要支出索引表的空间代价。

3.简单分析：

若块进行连续组织，设含有n个记录的文件被分成了b块,每块s个记录,检索等概率,则其平均检索长度最小为多少?

设在索引表中顺序查找，检索由两个阶段构成，则:

$$ASL=ASL_{index}+ASL_{block}$$

$$ASL_{index}=\sum_{i=1}^{b}(i/b)=(b+1)/2 \qquad ASL_{block}=\sum_{i=1}^{s}(i/s)=(s+1)/2$$

$\therefore$ ASL=(b+1)/2+(s+1)/2=(b+s)/2+1，又n=b×s

ASL=(n/s+s)/2+1

当 $S=\sqrt{n}$ 时, ASL最小ASL=1+ $\sqrt{n}\approx \sqrt{n}$

本节结束

一.概论

树形结构的一个重要应用就是用来组织目录和符号表。

树形结构的检索就是针对树目录和树表所组织的检索。

以树形结构组织的目录我们称为树目录；以树形结构组织的符号表我们称为树表。

以下举例说明树目录和树表。

树目录：

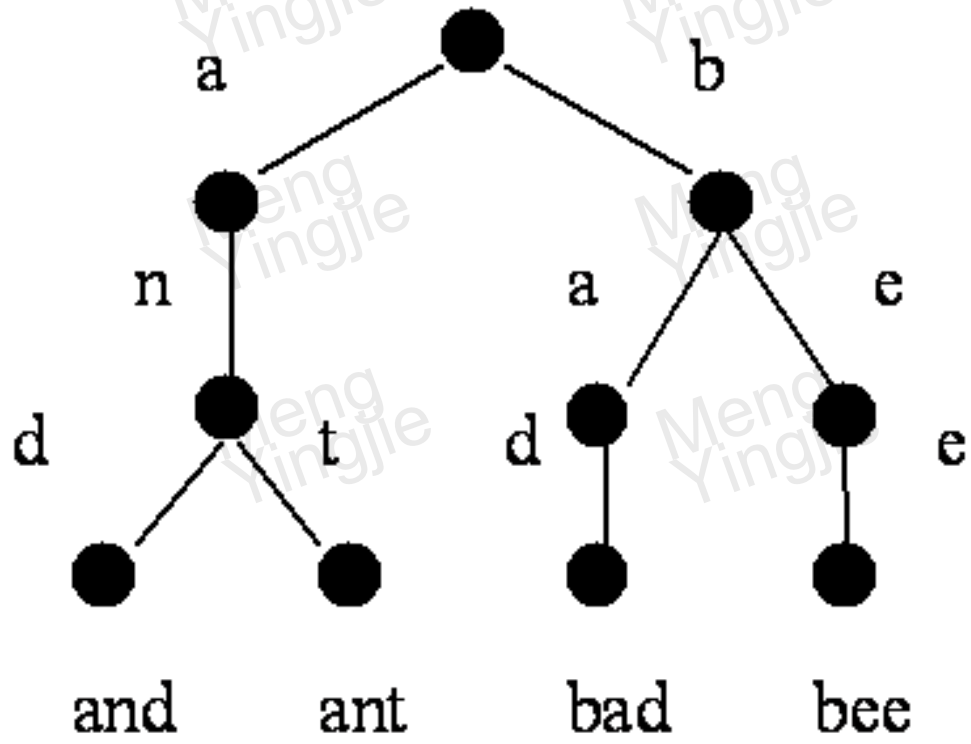
字典、书的组织体系；

计算机中的文件系统目录的组织。

树表：

汇编语言、编译程序等系统软件使用的符号表，是一个名(标识符)——值对的集合，这种表上需要经常进行检索、插入、删除等运算。

存储单词 and、ant、bad、bee

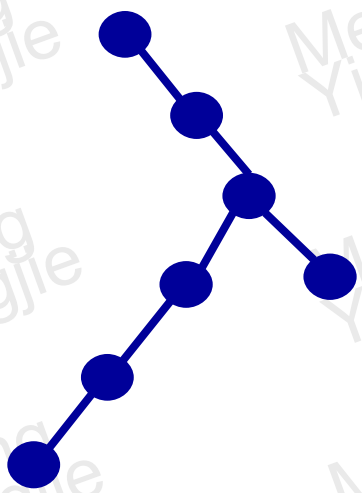


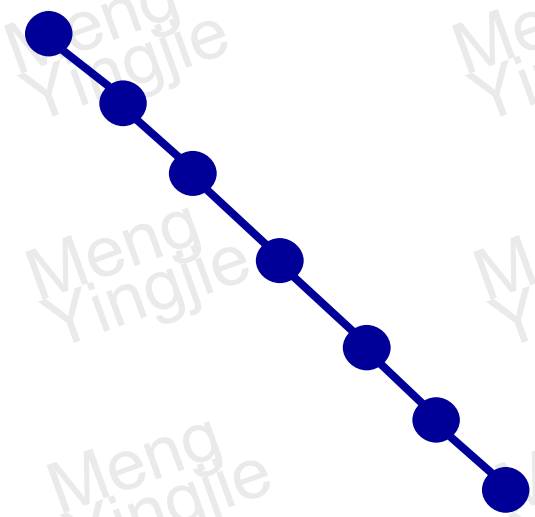
二.二叉排序树(binary sort tree)

各种树形结构的检索方法其本质都是以二叉排序树为根本进行演变的。

关于二叉排序树，详见§6.6，以二叉排序树为结构组织的检索结构也称为二叉检索树或二叉查找树。

二叉检索树的检索效率取决于该二叉树的深度，但该二叉树是动态生成的，由于输入的结点次序的不同可能导致出现不同的二叉树，也可能出现歪树，使其检索性能退化到与顺序检索等同的情况。如下图：





要保证其始终具有优良性能，可以采用AVL树。

三.AVL树(AVL-tree)

1962年艾德尔森.维尔斯基和南迪斯(ADELSON-VELSKII, G.M. and E.M.LANDIS: “An Algorithm for the Organization of Information”, Dokl. Akad. Nauk SSSR, Mathemat, 146(2):263-266, 1962)**引入了一种二叉树结构。**

这种二叉树它的子树的深度是平衡的,称为平衡二叉树(即AVL-树，或称为均高二叉树)。

按照这种平衡结构构造的二叉树称为最佳二叉排序树或最优二叉排序树。

1.概念：

AVL-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。

结点的平衡因子：结点的左子树高度减去右子树的高度的差值。

最优二叉排序树结点的平衡因子取值范围:

{-1, 0 ,1}。

2.AVL-树平衡的保持:

在平衡的二叉树中若进行插入或删除将可能导致不平衡情况的出现，这时就要运用一定的规则进行调整。

平衡规则:

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。以下讨论以插入情况为例:

平衡类型: 设结点A的平衡因子为±2

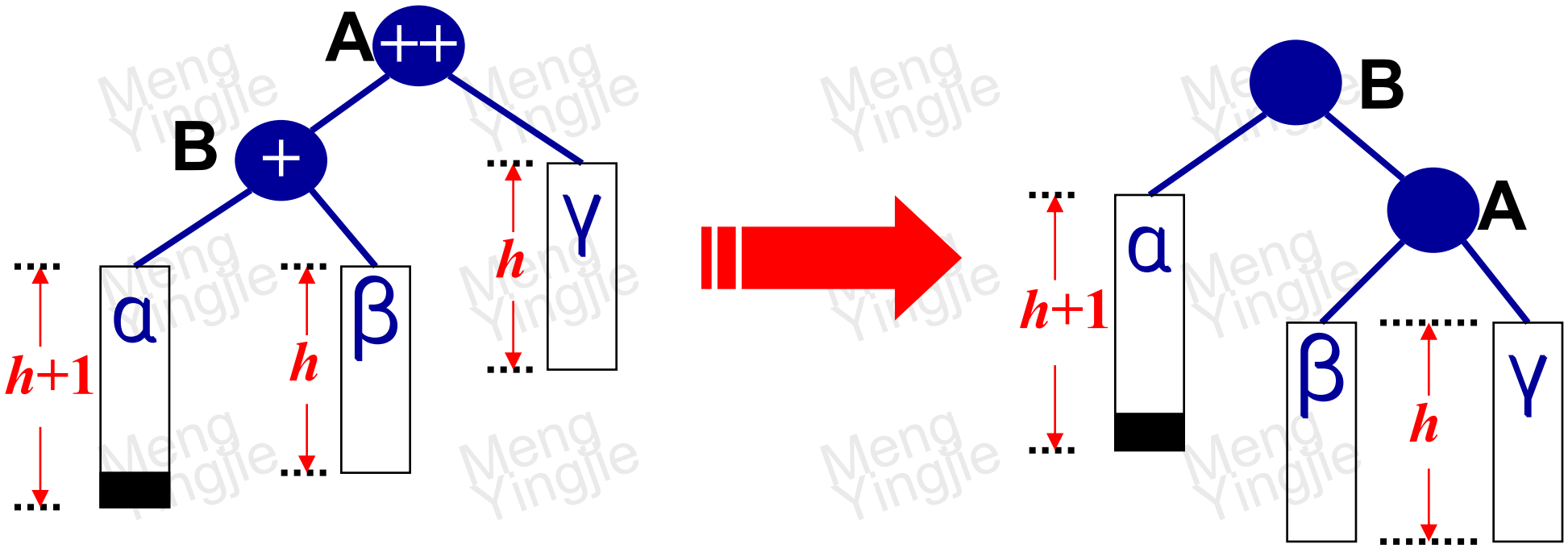
LL型: 新结点插在 A 的 左子树的 左子树中导致不平衡;

RR型: 新结点插在 A 的 右子树的 右子树中导致不平衡;

LR型: 新结点插在 A 的 左子树的 右子树中导致不平衡;

RL型: 新结点插在 A 的 右子树的左 子树中导致不平衡.

3.调整过程：LL-型



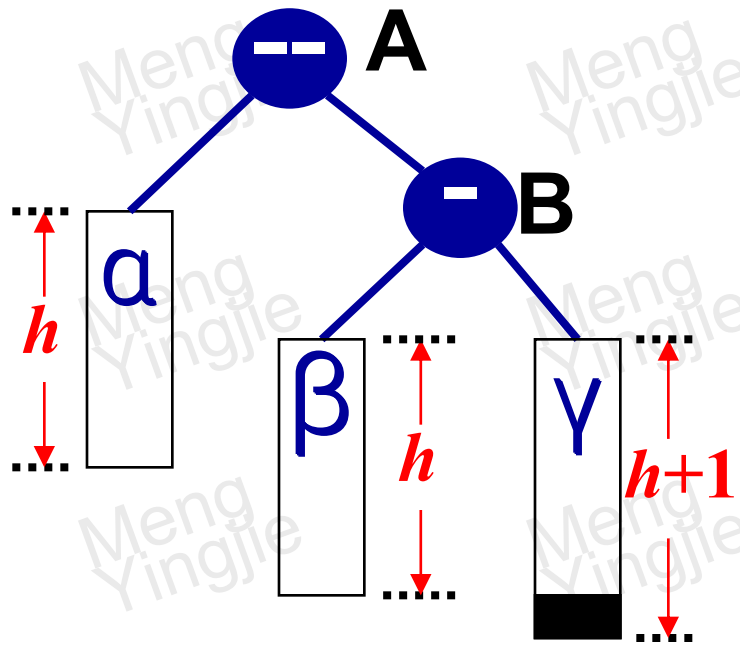
中序逻辑结果：

αBβAγ

中序逻辑结果：

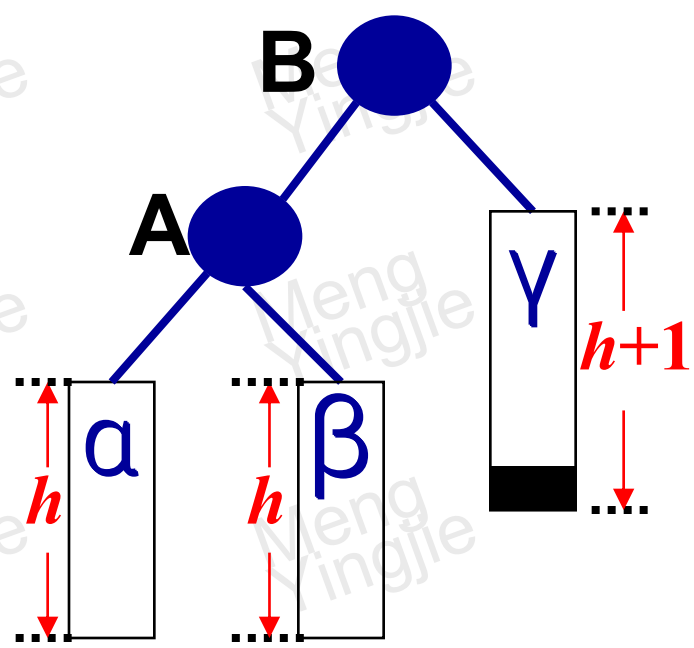
αBβAγ

3.调整过程：RR-型



中序逻辑结果：

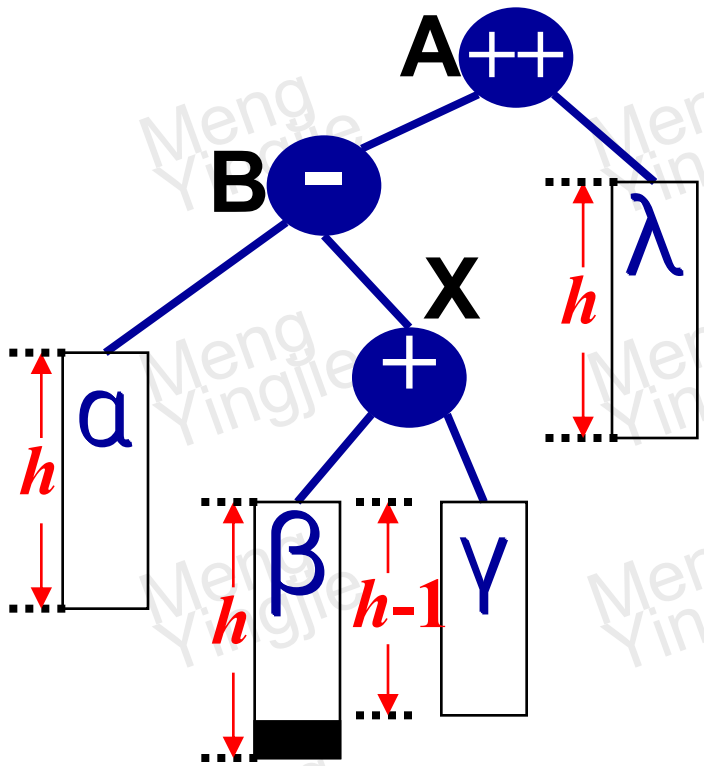
αAβBγ



中序逻辑结果：

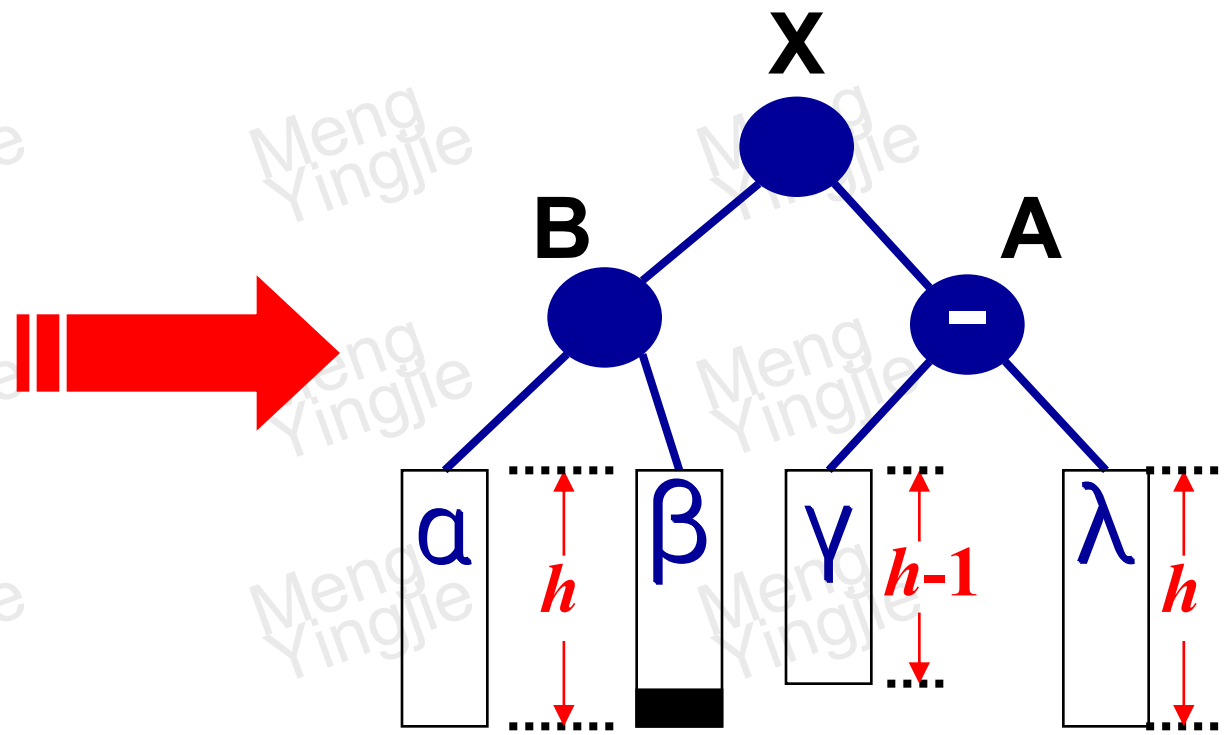
αAβBγ

3.调整过程：LR-型



中序逻辑结果：

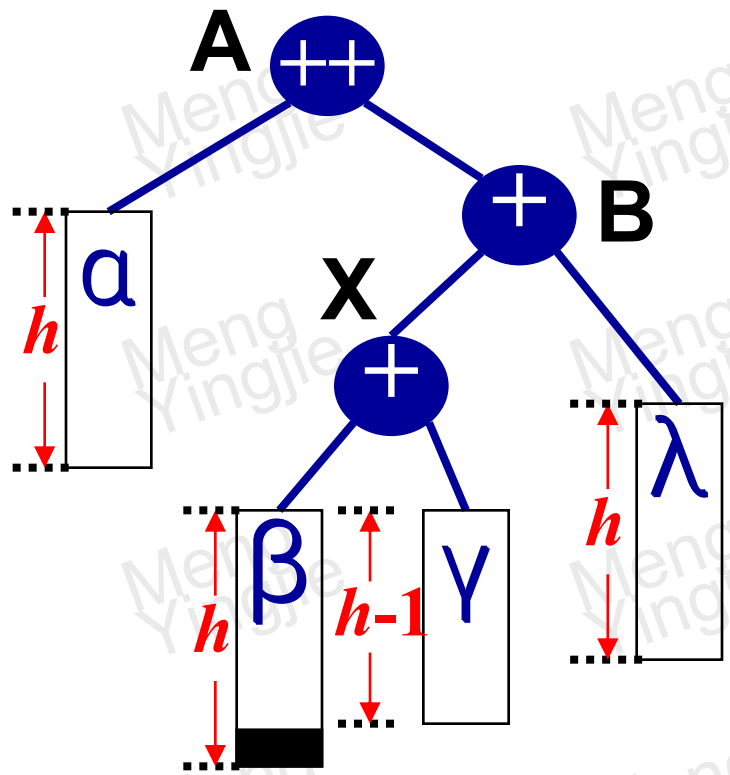
αBβXγAλ



中序逻辑结果：

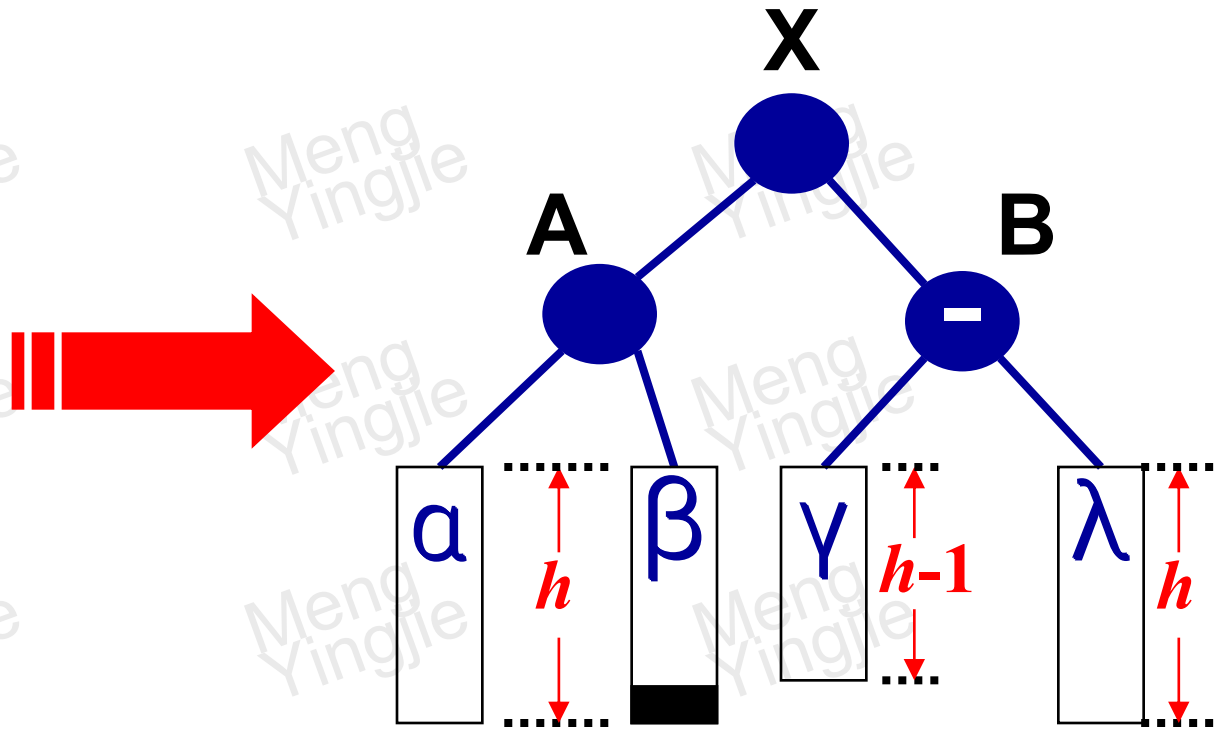
αBβXγAλ

3.调整过程：RL-型



中序逻辑结果：

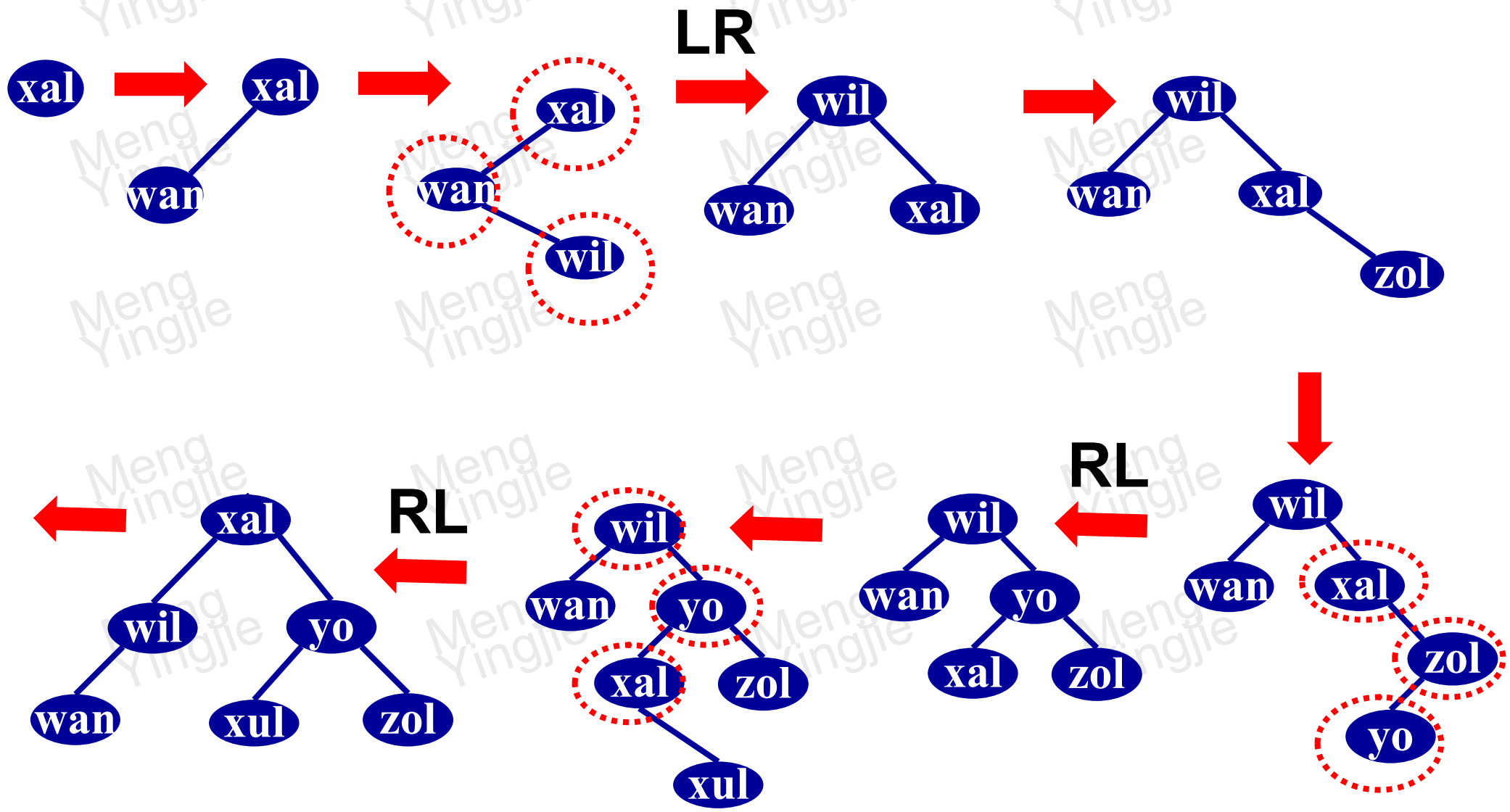
αAβXγBλ



中序逻辑结果：

αAβXγBλ

4.举例：有一个关键字集合K={xal,wan,wil,zol,yo, xul, yum, wen, wim}对其构造一棵最优二叉排序树。



* 四.字符树

1.概念:

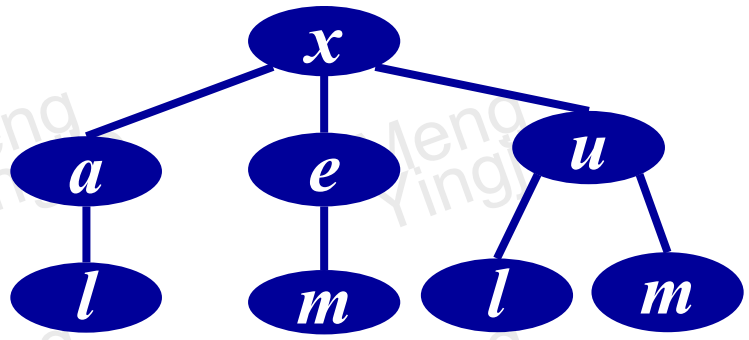
如果树目录中的每个结点对应于一个字符位置,则把这样的树目录称作字符树(也称键树、数字查找树)。

在这里我们可以给予“关键字”更广泛的含义，认为文件目录中每个文件的路径名，字典中的每个单词都是关键字。因此一个字符树就对应于一个关键字集合。

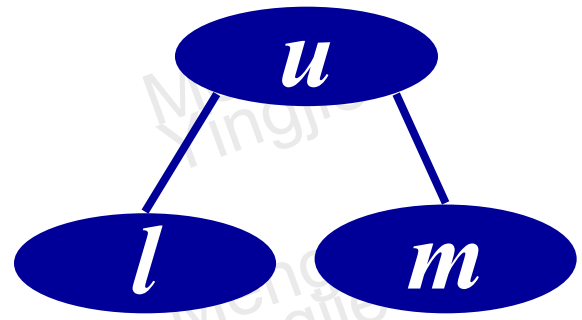
字符树中的每个结点对应于关键字中的一个字符位置。第1层的结点对应于关键字中第一个位置的字符，第2层的结点对应于关键字中第二个位置的字符，……，树叶对应于关键字中最后一个位置的字符。

把从根到一个树叶的路径上所有结点对应的字符连接起来得到的字符串就是一个关键字。树叶代表此关键字，整个树目录中包括的关键字个数等于树叶的树目。

如图所示是由关键字集合{xal,xem,xul,xum}组成的字符树。



字符树中，一棵子树代表具有相同前缀的关键字集合。如图所示子树代表具有相同前缀xu-的关键字集合{xul,xum}。



常见的字符树有双链树与trie结构。

2.字符树的检索过程：

首先用待查关键字的第一个字符与森林的各个根的字符进行比较，然后下一步的检索

在前次比较相等的那棵树上进行。其中，第二步是用待查关键字的第二个字符与选定的这棵树的根的各个子树进行比较，然后检索再沿着前次比较相等的分支进行下去，直到进行到某一层，该层所有结点的字符都与待查关键字相应位置的字符不同，这说明此关键字在树目录里没有出现。

若检索一直进行到树叶，那么就在树目录里找到了给定的关键字。

3.双链树：

把一个关键字集合中出现的所有关键字用字符树的形式表示出来称为**Briandais树**(实际上往往是一个森林，该结构由Rene de la Briandais首先提出的)，把它转换为对应的二叉树并用llink-rlink法进行存储，则通常称作**双链树**。

在双链树中检索给定关键字是沿着指针进行的。当待查关键字中的一个字符与双链树中对应层的第一个结点比较不等时，则沿着rlink依次与同层的其它兄弟比较；当比较相等时则沿着llink进入双链树的下一层，并考查待查关键字中下一个位置的字符。

4.trie结构：

trie来源于retrieval, 该结构是一种检索效率高的结构。

导致双链树检索效率较低的主要原因是，要在每一层中沿着rlink在兄弟集合中顺序查找与待查关键字中对应字符匹配的结点。

trie结构采用了另一种存储方式，使得在每一层中不必须顺序查找，可以直接定位到要找的结点。

trie结构是一种字符树，它的第i层的每个结点对应于关键字的第i个位置的一个字符,但trie结构的结点并不存储此字符,实际上可看成扩充树,整个关键字存储在扩充结点中。

4.trie结构(续)：

trie结构的每个结点由一个指针向量来构成，它的每个分量是一个指针。对于关键字i+1个位置上允许出现的每一个字符在第i层结点中都有一个指针位置与之对应。

对于结点的向量中指针编排次序的组织，以该位置可出现的字符编码顺序进行组织，也就是说每个指针隐含地对应着一个特定的具体字符。

假设每个关键字结尾附加一个字符‘*’；对于一般字符，指针位置非空时，指向下一层的某个结点，这时对应的字符‘*’位置指针应为空；当对应的字符‘*’位置非空时，则该指针指向该关键的数据地址。

trie结构举例： 设关键字集合K={xal,zol,yo,xul,yum,zi,yon,xem,zom}.

关键字第1字符

字符	*	x	y	z
序号	1	2	3	4

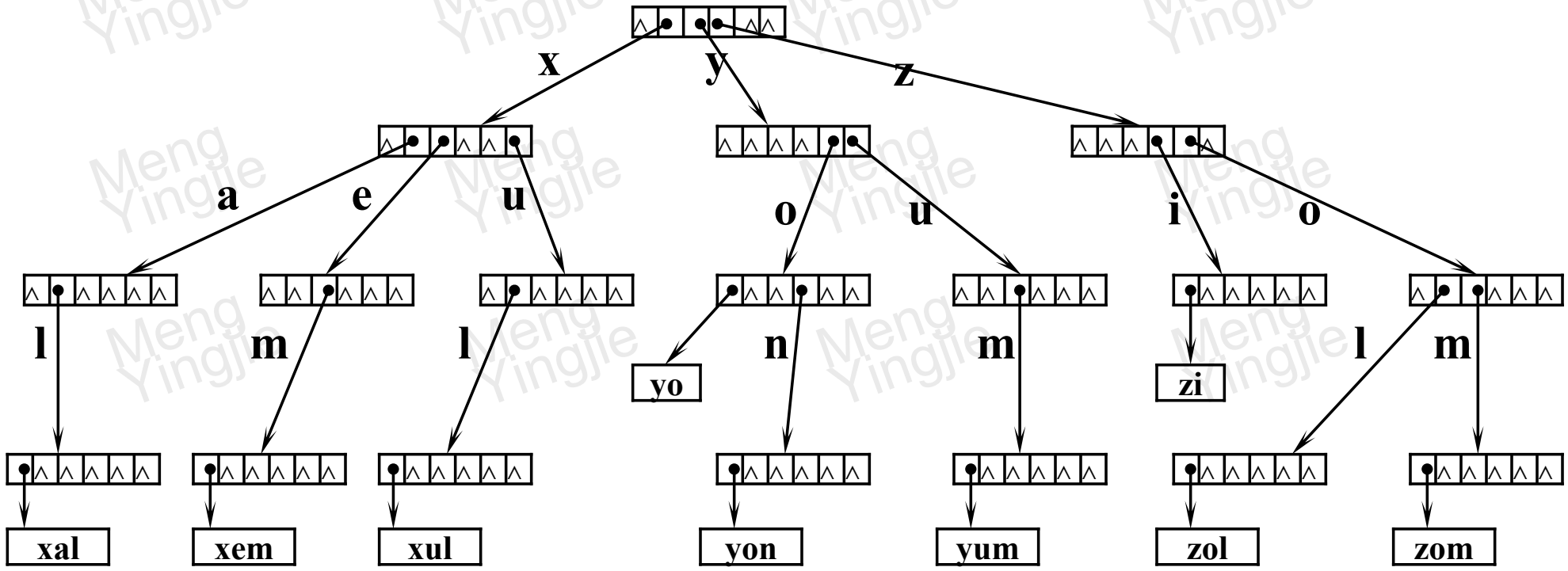
关键字第2字符

字符	*	a	e	i	o	u
序号	1	2	3	4	5	6

关键字第3字符

字符	*	l	m	n
序号	1	2	3	4

设采用等长结点存储,每个结点取6个指针域(各层中允许出现的最大字符个数)。则K的存储示意图如下：



字符树小结：

trie结构的检索效率高是因为它的结点中为下一层的每一个允许字符都保留指针位置，因此可以直接定位进入下一层对应的结点，不必象双链树那样在下一层的兄弟集合中进行顺序比较。

然而，在实际的关键字中下一个字符位置上很可能许多允许的字符并不出现，所以很多指针位置为空。所以trie结构比双链树占用的存储量大。

一个既节省存储又有较高检索效率的方案是把**trie结构和双链树结合起来**。在靠近根的地方，兄弟集合的密度大，可采用trie结构；树的最下面几层兄弟集合的密度小，则采用双链树。

本节结束

一.概论

散列表既是一种存储方法, 也是一种常见的检索方法。

它是按关键字编址的一种技术。

散列法(Hashing, 也称杂凑表、混列表、哈希表)是在1968年以后才逐渐通用起来的一种存储和检索的技术。

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**。

在散列法中把地址映射函数称为**散列函数**(hash function)。

把用散列法组织存储的**线性表**称为**散列表**(hash table)。

散列的数据存储及检索过程：

存储过程：

首先将要处理的关键字看作一个变量K,然后按照一个确定的函数I(K)，映射到表中的某个地址，若此地址还未存储关键字，就将此关键字存储在该位置，若此位置已有其他的关键字G，那么根据另一函数J，映射J(I(K))到表的另一地址，再查看此位置上是否已经存储别的关键字，否则再次映射。如此重复直到找到某位置尚未存储别的关键字为止，就将此关键字存储在该位置上。

查找过程是存储过程的逆过程。

二.散列函数:

为了获得一个好的散列函数，通常应当使得函数与组成关键字k的所有符号有关。

此外如果k是从关键字集合中随即抽取的一个，原则上我们希望H(k)以同等概率取地址空间中的每一个值，如果一个散列函数满足这一性质，则称该散列函数是均匀的。

但寻找一个均匀的散列函数又非常困难。常见的散列函数：

1.直接定址:

取关键字的某个线性函数值。

$H(k)=a\times k+b$， a,b为常数。

也称自身函数。

注意:

对于由字符组成的关键字在计算散列地址时可将关键字中的符号看成其内部表示编号值或编码值，这样就可以参与运算了，以下各函数中类似。

2.除余法:

选择一个适当的正整数p,用p去除关键字，取其余数作为地址，即modulo-P residue ， 可形式化表示为：

$$H(k)=k \text{ MOD } P$$

该方法的关键是选取适当的p， 经理论和实验分析一般认为选取p为小于基本存区长度n的最大素数为益。

n=8,16,32,64,128,256,512,1024,……

p=7,13,31,61,127,251,503,1019,……

3.数字分析法:

对各个关键字内部代码的各位进行分析,抽取分布比较均匀的若干位作为地址,抽取的位数取决于地址码的位数。

例:

关键字	散列地址
00031942**6**	326
00071830**9**	709
00062944**3**	643
00075861**5**	715
00091969**7**	997
00031032**9**	329

常用于关键字的位数远多于地址码的位数情况下。缺点:

4.分划法:

也称折叠法。若关键字很长，且可变的；或者关键字由多个域组成，此时散列技术可将关键字的内部代码分割成多个部份，并将这些部份按某种规律结合起来。

例，不同的分划，及结合：k=582422241,地址码4位。

582422241　　582422241　　582422241

移位折叠相加　　移位相加　　移位相加

85　　58　　582

142　　241　　41

+ 2422　　+ 2422　　+ 4222

11064　　2721　　4845

H(k)=1064　　H(k)=2721　　H(k)=4845

5.平方取中法:

也称中平法。先计算出关键字的平方值，再抽取它的中间若干几位作为散列地址。

有时抽取获得的函数值会超过或达不到地址空间编码的位数，这时可以对函数值再乘以一个比例因子进行变换，使其能够落到基本区地址范围。

6.基数转换法:

关键字的符号组成一般是建立在一定基数制上的，我们可以将该关键字看成是另一个基数制上的表示，再将其还原为原来基数制上的表示后，抽取其若干位作为散列地址。

例： k=236075,为十进制下的关键字。

$(236075)_{13}===>(841547)_{10}$

取2、3、4位作为散列地址,H(k)=415

其目的主要是希望将关键字之间的距离放大，因此一般要求取大于原来的基数制，同时两个数制之间最好互素。

7.随机数法:

随机数在概率算法设计中扮演着十分重要的角色。

在现实计算机上无法产生真正的随机数，因此在概率算法中使用的随机数都是一定程度上随机的，即伪随机数。

产生伪随机数最常用的方法是线性同余法。由线性同余法产生的随机序列$a_1,a_2,\ldots,a_n,\ldots$满足：

$$\begin{cases} a_0 = d \\ a_n = (ba_{n-1}+c) \text{ MOD } m, \quad n=1,2,\ldots \end{cases}$$

其中，$b\geq 0$, $c\geq 0$, $d\geq m$。d称为该随机序列的种子。如何选取b,c,m直接关系到所产生的随机序列的随机性能，这属于随机性理论研究的范畴。

三.碰撞的产生及处理

1.碰撞及其产生:

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)。

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)。

$k_1 \neq k_2$, $H(k_1)=H(k_2)$

导致碰撞的主要原因是相对资源的有限性与相对需求的无限性造成的，往往很难在这两者之间建立完美的对应关系(函数).

例如,人群中的某些人的生日会映射到同一天(引伸说明：生日判定,birthday paradox)；

为了说明碰撞需要引入一个概念，负载因子：

基本区： H(k)值域对应的空间。

$$\alpha = \frac{\text{散列表中结点的数目}}{\text{基本区可容纳的结点数目}}$$

通常的情况是负载因子的大小要选取的适当，目的是既不增加碰撞，又有较快的检索速度，又不浪费存储空间。

一个好的散列函数应当使得计算出的地址尽可能均匀地分布在基本区中，然而一个好的散列通常只能减少碰撞发生的次数，无法保证绝对不产生碰撞。这是因为：

α>1,碰撞不可避免，一般取α<1。α越小碰撞的几率就越小，但这就要求基本区越大，这往往不可取.

因此散列法除去要选择适当的散列函数以外，还要研究发生碰撞时如何解决，即用什么方法存储同义词。

但也不排除存在没有碰撞的情况。我们把没有碰撞的系统称为完美散列(perfect hashing)。选择一个完美的散列函数需要很高的代价，但在需要很高检索性能时也是值得的。例如CD-ROM中的数据是永远不会变的，但需要很高的检索速度。

2.碰撞的处理:

当碰撞发生后处理碰撞的方法基本上有两类：拉链法和开地址法，也有分为开散列(open hashing,基本区外)和闭散列(closed hashing,基本区内)的。以下逐一讨论。

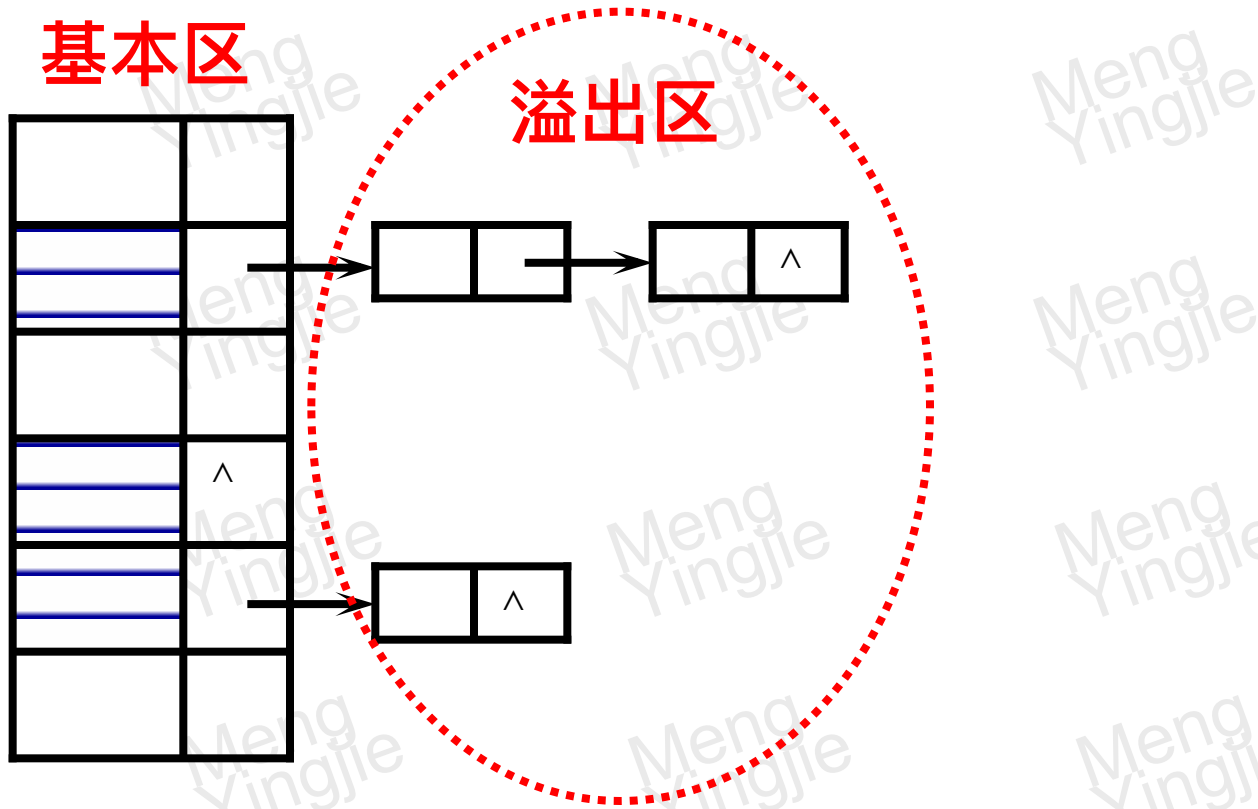
(1).拉链法：

当碰撞发生时,就拉出一条链,建立一个链接方式的同义词子表。依据同义词建立区域的不同有两种方式：

- 分离的同义词子表法；
- 结合的同义词子表法。

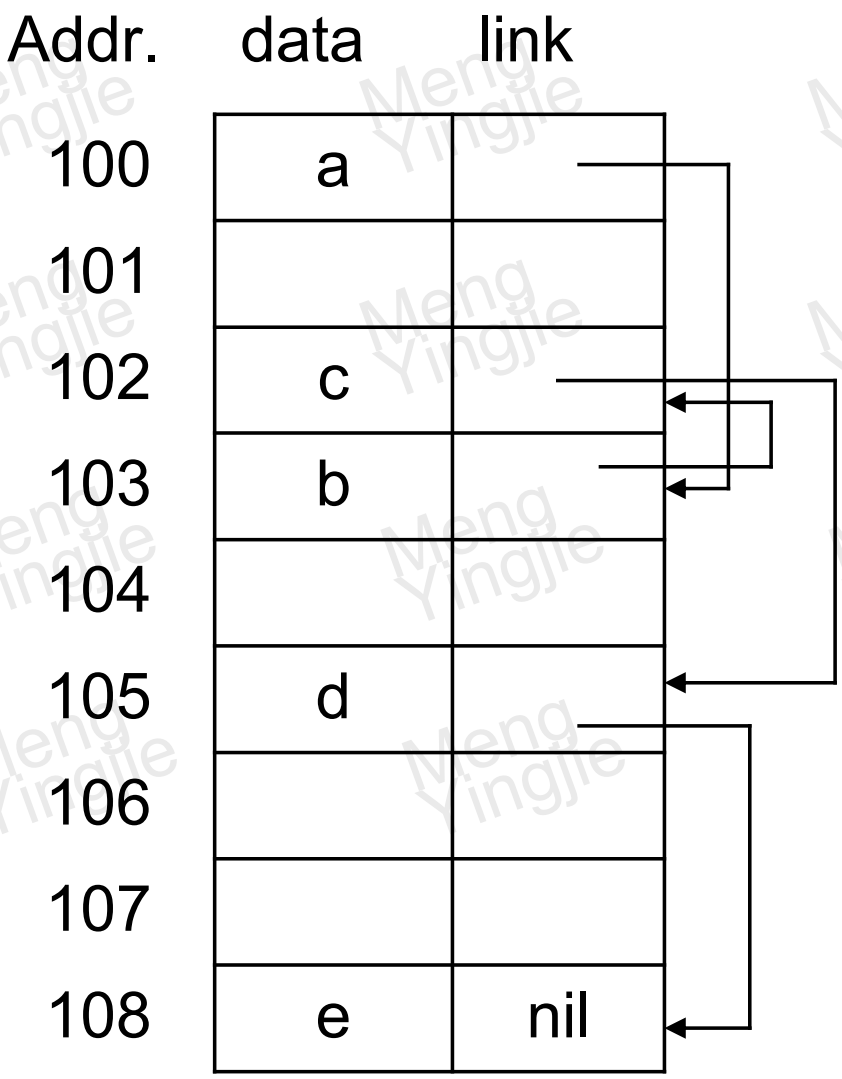
①.分离的同义词子表法：

也称外链法(属于开散列)。 在基本区之外开辟一个溢出区存储同义词。



②.结合的同义词子表法：

也称内链法(属于闭散列)。在基本区内目前还没有被占用的空间开辟溢出区存储同义词。即在基本区内进行拉链。



易出现堆积现象：

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”

(2).开放地址法：

基本思想：

当碰撞发生时,用某种方法形成一个探测的序列, 沿着这个序列一个个单元地查询, 直到找到这个关键字或者找到一个开放的地址(open addressing,即没有进行存储的空单元)。

如果是存储操作，则遇到开放地址就可进行存储；

如果是检索操作，则遇到开放地址就是检索失败。

针对探测序列的产生不同,主要有：

- 线性探测
- 双散列探测。

①.线性探测法(linear probing)：

是最简单的顺序查找方法。当碰撞发生时,到顺序的下一个基本存区单元去进行探测，即：若H(k)=d，此时发生碰撞，则探测序列为：

d+1,d+2,…,m-1,0,1,2,…,d-1

m为基本存区长度。

也可能产生堆积现象。为改善堆积可以采用双散列函数探测法。

②.双散列探测法(double hashing)：

为了改善堆积现象我们可以使用双(重)散列。

该方法使用两个散列函数H_1和H_2，其中H_1与前面的散列函数一样，其值域为[0，m-1]；H_2也以关键字为自变量，函数值必须是一个与M互素的并属于区间[1，m-1]的整数作为散列地址的补偿。即：若H(k)=d，此时发生碰撞，则探测序列可以为：

(d+h2(k))MOD m, (d+2h2(k))MOD m , (d+3h2(k))MOD m ,…

也就是说探测序列是跳跃的。

本节结束

一.概论

在检索的实际应用中，经常要检索文件中满足某个或若干个属性满足一定条件的结点，我们把这类检索称为**基于属性的检索**。

利用前面的检索方法也可以处理这类问题，因要遍查整个数据集，所以代价很大。

因此为了保持较高的检索效率，需要采取一些特殊的结构和方法。这里主要讨论**倒排表**和**多重表**。

二.倒排表(inverted list):

在原来数据表的基础上，对于感兴趣的(即可用来作为检索参数的)每个属性的每个值都建立一个线性表，该线性表用作存放与此属性相对应的所有关键字的值。这种辅助线性表称为倒排表。

将原来的数据表可以称作主表。

倒排表可以看成主表的辅助表，本质实际上是一个主表的索引表。

例:对于一个人员信息表,其倒排表可以构造如下：

主关键字

一个人员的信息表

首地址	编号	姓名	年龄	职称	部门	……	籍贯	……
100	0001	于××	35	讲师	文学院		上海	
110	0002	王××	52	教授	信息院		北京	
120	0003	李××	43	副教授	物理院		浙江	
130	0004	刘××	28	助教	信息院		陕西	
140	0005	孙××	32	讲师	文学院		北京	
150	0006	张××	48	副教授	化工院		陕西	
160	0007	周××	27	讲师	物理院		上海	
170	0008	王××	25	助教	文学院		陕西	
180	0009	赵××	42	副教授	美术院		浙江	
190	0010	徐××	36	讲师	信息院		北京	
200	0011	崔××	24	助教	化工院		陕西	
210	0012	徐××	56	教授	美术院		上海	
220	0013	刘××	30	讲师	化工院		陕西	

倒排表：

职称表	主关键字
教授	0002,0012
副教授	0003,0006,0009
讲师	0001,0005,0007,0010,0013
助教	0004,0008,0011

籍贯表	主关键字
北京	0002,0005,0010
浙江	0003,0009
上海	0001,0007,0012
陕西	0004,0006,0008,0011,0013

部门表	主关键字
文学院	0001,0005,0008
美术院	0009,0012
信息院	0002,0004,0010
物理院	0003,0007
化工院	0006,0011,0013

年龄段	主关键字
20~29	0004,0007,0008,0011
30~39	0001,0005,0010,0013
40~49	0003,0006,0009
50~59	0002,0012

在信息检索中，倒排表实际上是由检索词和信息记录的主控键构成的行列矩阵。

职称表	主关键字
教授	0002,0012
副教授	0003,0006,0009
讲师	0001,0005,0007,0010,0013
助教	0004,0008,0011

建立了倒排表后，处理检索效率就比较高了，但是花费了保存倒排表的存储代价。我们只要检索相关的倒排表 (即属性表)即可。

在倒排表上进行插入或删除操作时，不但要改变主表，还要修改行相应的辅助表，这样还会降低更新运算的效率。

但倒排表不能代替原来的主信息表，其必须和主表并存才有实际意义。

三.多重表:

如果在倒排表中对于每个属性不是给出对应于它的所有结点的关键字的值，而是给出主表中这些结点的存储地址，那么对于结点的存取就会更快速一些，因为不必再根据关键字的值检索主表了，而可以直接按给出的地址去存取。

职称倒排表(即索引表)

教授	110,210
副教授	120,150,180
讲师	100,140,160,190,220
助教	130,170,200

主 表

首地址	编号	姓名	年龄	职称	部门	籍贯
100	0001	于××	35	讲师	文学院	上海
110	0002	王××	52	教授	信息院	北京
120	0003	李××	43	副教授	物理院	浙江
130	0004	刘××	28	助教	信息院	陕西
140	0005	孙××	32	讲师	文学院	北京
150	0006	张××	48	副教授	化工院	陕西
160	0007	周××	27	讲师	物理院	上海
170	0008	王××	25	助教	文学院	陕西
180	0009	赵××	42	副教授	美术院	浙江
190	0010	徐××	36	讲师	信息院	北京
200	0011	崔××	24	助教	化工院	陕西
210	0012	徐××	56	教授	美术院	上海
220	0013	刘××	30	讲师	化工院	陕西

三.多重表(续):

基于以上的思想，我们也可以对以上结构进行改造。以职称倒排表为例进行说明。

主表中将具体属性值去掉 ,用链接字段代替 ,把相同属性值的结点链接起来；倒排表中给出属性值和该属性值的首元素指针，这样倒排表就演变成为表头的表——头表。

职称头表

属性值	个数	首指针
教授	2	110
副教授	3	120
讲师	5	100
助教	3	130

主 表

首地址	编号	姓名	年龄	职称	部门	籍贯
100	0001	于××	35	140	文学院	上海
110	0002	王××	52	210	信息院	北京
120	0003	李××	43	150	物理院	浙江
130	0004	刘××	28	170	信息院	陕西
140	0005	孙××	32	160	文学院	北京
150	0006	张××	48	180	化工院	陕西
160	0007	周××	27	190	物理院	上海
170	0008	王××	25	200	文学院	陕西
180	0009	赵××	42	^	美术院	浙江
190	0010	徐××	36	220	信息院	北京
200	0011	崔××	24	^	化工院	陕西
210	0012	徐××	56	^	美术院	上海
220	0013	刘××	30	^	化工院	陕西

三.多重表(续):

经这样处理后的形成的主表和头表统称为多重表,也称多表组织(multilist organization).这个多重表是可以代替原来的信息表的。

在主表中每个属性值构成的链表还可以形成循环链表。

头表

职称	个数	首指针	部门	个数	首指针`	…
教授	2	110	文学院	3	100	
副教授	3	120	美术院	2	180	
讲师	5	100	信息院	3	110	
助教	3	130	物理院	2	120	
			化工院	3	150	

新 主 表

首地址	编号	姓名	年龄	职称	部门	籍贯
100	0001	于××		140	140	
110	0002	王××		210	130	
120	0003	李××		150	160	
130	0004	刘××		170	190	
140	0005	孙××		160	170	
150	0006	张××		180	200	
160	0007	周××		190	^	
170	0008	王××		200	^	
180	0009	赵××		^	210	
190	0010	徐××		220	^	
200	0011	崔××		^	220	
210	0012	徐××		^	^	
220	0013	刘××		^	^	

三.多重表(续):

差异:

多重表

多链表

本节结束

1.设关键字输入顺序为：4、5、7、2、1、3、6请构造AVL树，画出构造过程图示.

2、设散列空间为[0..12]，散列函数为H(k)=k MOD 13,给定的关键字序列为：19，14，23，01，68，20，84，27，55，11，10，79。试画出分别用拉链法和线性探测再散列解决冲突时所构造出的散列表，并求出在等概率情况下，这两种方法的查找成功时的平均查找长度。

本章结束

概率算法(probabilistic algorithm)

1976年雷宾(M.Rabin)提出了一种新的设计算法的方法，就是概率算法(也称随机算法)。这种算法的新颖之处是把随机性注入到算法中，使得算法设计与分析的灵活性及解决问题的能力大为改观。

概率算法是指某些步骤或步骤中的某些动作是随机性(即某些步骤可以随机地选择下一步)的算法。

概率算法的基本特征是对所求解问题的同一实例用同一概率算法求解两次，可能得到完全不同的效果。这两次求解所需的时间甚至所得的结果可能会有相当大的差别。

一般情况下，可将概率算法分为四类：数值概率算法、蒙特卡罗(Monte Carlo)算法、拉斯维加斯(Las Vegas)算法、舍伍德(Sherwood)算法。

生日判定(birthday paradox)

在研究检索技术时，特别是研究散列存储时使用的一个著名概率判定。此判定于1939年发表，其义如下：

若一个房间有23个以上人，则其中有2个人的生日相同的机会是大的。

使用标准数学语言可以这样叙述：

若有一个均匀的映射函数把23个不同的、属于整数集的数映射到区间[1,365]时，则2个数映射到同一位置的概率是0.5073，即大于1/2。

也就是说，当选出一个散列函数后，在仅有23个不同关键字而表长却有365时，发生碰撞的概率比不发生碰撞的概率还要大。利用该判定就可以了解到在使用散列技术时为什么冲突是不可避免的。

附录结束