

查找

目 录

Contents

1

静态查找

2

动态查找

3

散列表

查找表(Search Table): 相同类型的数据元素(对象)组成的集合, 每个元素通常由若干数据项构成。

关键字(Key, 码): 数据元素中某个(或几个)数据项的值, 它可以标识一个数据元素。若关键字能**唯一**标识一个数据元素, 则关键字称为**主关键字**; 将能标识若干个数据元素的关键字称为**次关键字**。

查找/检索(Searching): 根据给定的K值, 在查找表中确定一个关键字等于给定值的记录或数据元素。

◆ 查找表中**存在**满足条件的记录: 查找成功; 结果: 所查到的记录信息或记录在查找表中的位置。

◆ 查找表中**不存在**满足条件的记录: 查找失败。

查找

查找有两种基本形式：静态查找和动态查找。

静态查找(Static Search)：在查找时只对数据元素进行查询或检索，查找表称为静态查找表，适合静态查找表的查找方法有：顺序查找、折半查找、散列查找等。

动态查找(Dynamic Search)：在实施查找的同时，**插入**查找表中不存在的记录，或从**查找**表中删除已存在的某个记录，查找表称为动态查找表。适合动态查找的查找方法有二叉排序树的查找。

查找方法评价指标

查找过程中主要操作是关键字的比较，查找过程中关键字的**平均比较次数(平均查找长度ASL: Average Search Length)**作为衡量一个查找算法效率高低的**标准**。ASL定义为：

$$ASL = \sum_{i=1}^n P_i \times C_i \quad \mathbf{n} \text{ 为查找表中记录个数}$$

$$\sum_{i=1}^n P_i = 1$$

其中：

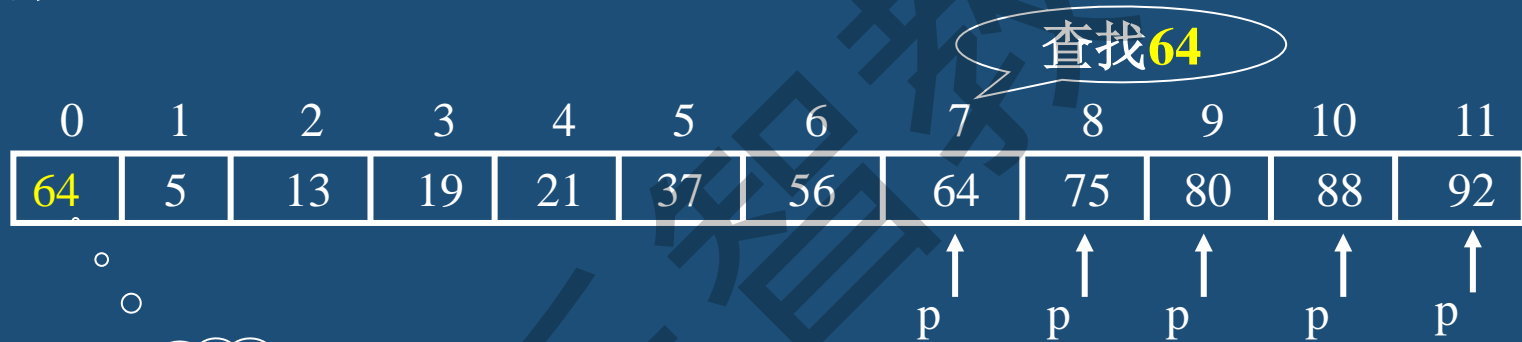
P_i ：查找第*i*个记录的概率；

C_i ：查找第*i*个记录需要进行比较的次数。

顺序查找

1 查找思想

从表的一端开始逐个将记录的关键字和给定K值进行比较，若某个记录的关键字和给定K值相等，查找成功；否则，若扫描完整个表，仍然没有找到相应的记录，则查找失败。



0	1	2	3	4	5	6	7	8	9	10	11
64	5	13	19	21	37	56	64	75	80	88	92

↑ p ↑ p ↑ p ↑ p ↑ p

监视哨

比较次数=5

2 算法分析

不失一般性，设查找每个记录成功的概率相等，即 $P_i=1/n$ ；查找第 i 个元素成功的比较次数 $C_i=n-i+1$ ；

◆ 查找成功时的平均查找长度ASL：

$$ASL = \sum_{i=1}^n P_i \times C_i = \frac{1}{n} \sum_{i=1}^n (n-i+1) = \frac{n+1}{2}$$

◆ 包含查找不成功时：查找失败的比较次数为 $n+1$ ，若成功与不成功的概率相等，对每个记录的查找概率为 $P_i=1/(2n)$ ，则平均查找长度ASL：

$$ASL = \sum_{i=1}^n P_i \times C_i = \frac{1}{2n} \sum_{i=1}^n (n-i+1) + \frac{n+1}{2} = 3(n+1)/4$$

折半查找

折半查找又称为二分查找，是一种效率较高的查找方法。

前提条件：查找表中的所有记录是按关键字有序(升序或降序)。

查找过程中，先确定待查找记录在表中的范围，然后逐步缩小范围(**每次将待查记录所在区间缩小一半**)，直到找到或找不到记录为止。

1 查找思想

用Low、High和Mid表示待查找区间的下界、上界和中间位置指针，初值为Low=1，High=n。

(1) 取中间位置Mid: $Mid = \lfloor (Low + High) / 2 \rfloor$;

(2) 比较中间位置记录的关键字与给定的K值:

① 相等: 查找成功;

② 大于: 待查记录在区间的前半段, 修改上界指针: $High = Mid - 1$, 转(1);

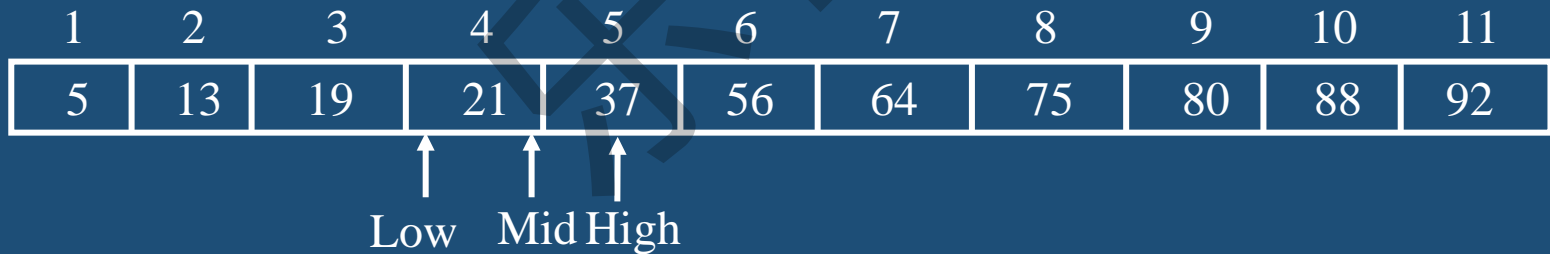
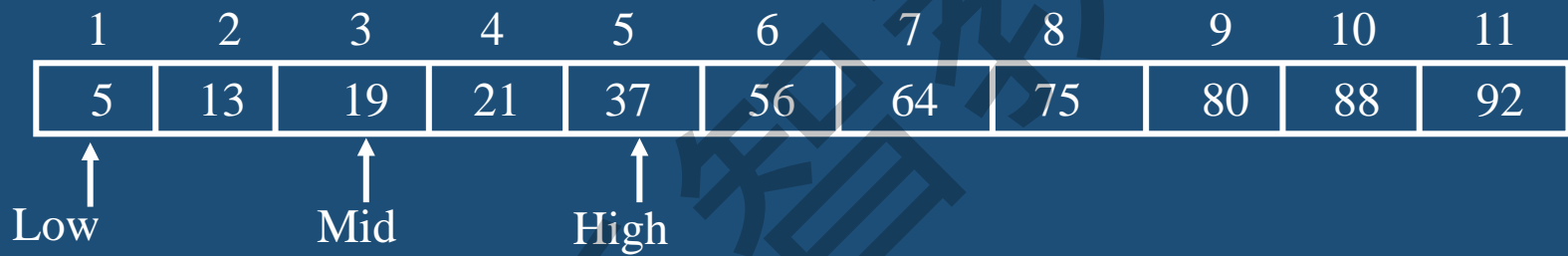
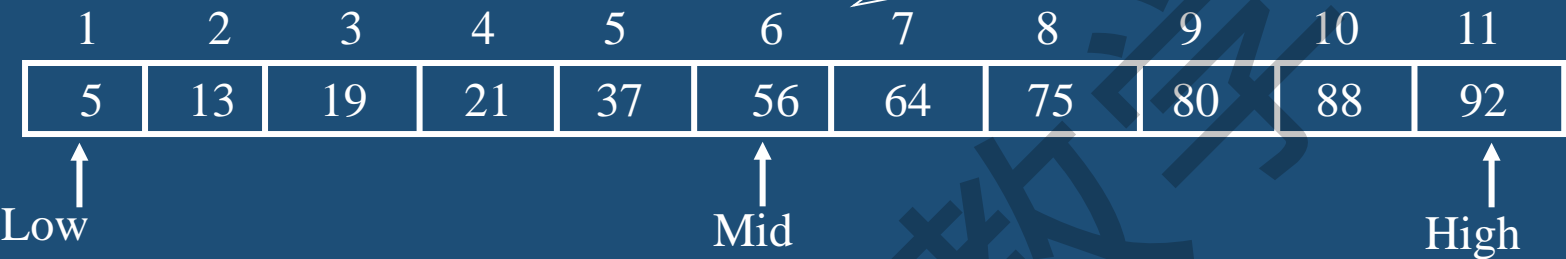
③ 小于: 待查记录在区间的后半段, 修改下界指针: $Low = Mid + 1$, 转(1);

直到越界($Low > High$), 查找失败。

折半查找

查找21

查找成功示例



查找71

Low Mid High

↑ ↑ ↑
Low Mid High

↑ ↑ ↑
Low Mid High

2. 算法分析

① 查找时每经过一次比较，查找范围就缩小一半，该过程可用一棵二叉树表示：

- ◆ 根结点就是第一次进行比较的中间位置的记录；
- ◆ 排在中间位置前面的作为左子树的结点；
- ◆ 排在中间位置后面的作为右子树的结点；

对各子树来说都是相同的。这样所得到的二叉树称为判定树(Decision Tree)。

② 将二叉判定树的第 $\lfloor \log_2 n \rfloor + 1$ 层上的结点补齐就成为一棵满二叉树，深度不变， $h = \lceil \log_2(n+1) \rceil$ 。

折半查找

③ 由满二叉树性质知，第 i 层上的结点数为 2^{i-1} ($i \leq h$)，设表中每个记录的查找概率相等，即 $P_i = 1/n$ ，查找成功时的平均查找长度ASL:

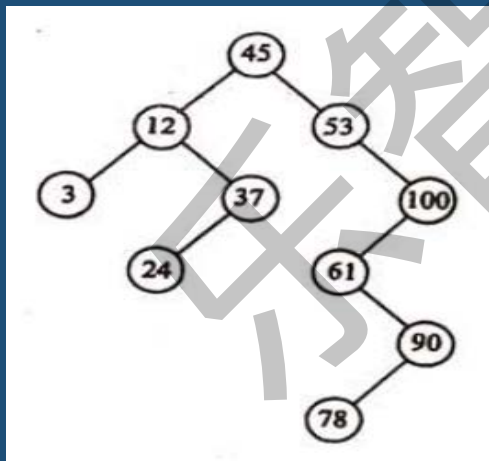
$$ASL = \sum_{i=1}^n P_i \times C_i = \frac{1}{n} \sum_{j=1}^h j \times 2^{j-1} = \frac{n+1}{n} \log_2(n+1) - 1$$

当 n 很大 ($n > 50$) 时， $ASL \approx \log_2(n+1) - 1$ 。

二叉排序树的查找

二叉排序树，又叫二叉查找树，它或者是一棵空树；或者是具有以下性质的二叉树：

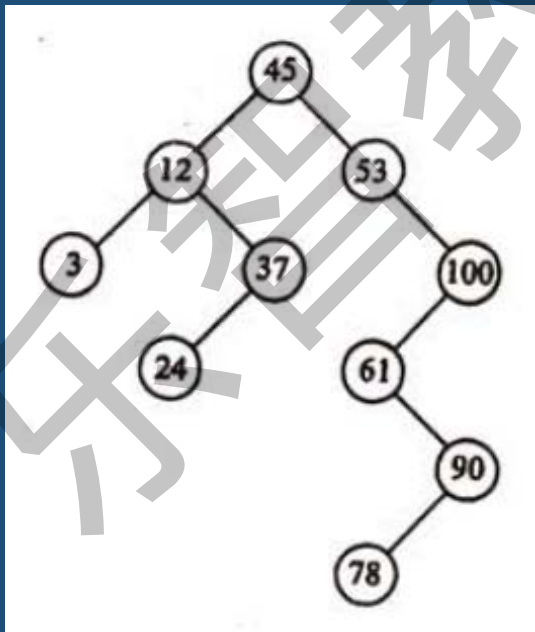
1. 若它的左子树不空，则左子树上所有节点的值均小于它的根节点的值；
2. 若它的右子树不空，则右子树上所有节点的值均大于它的根节点的值；
3. 它的左右子树也分别为二叉排序树。



二叉排序树的查找

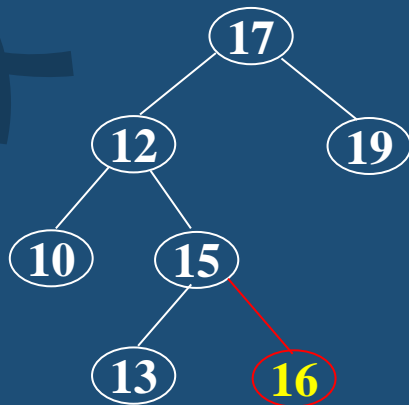
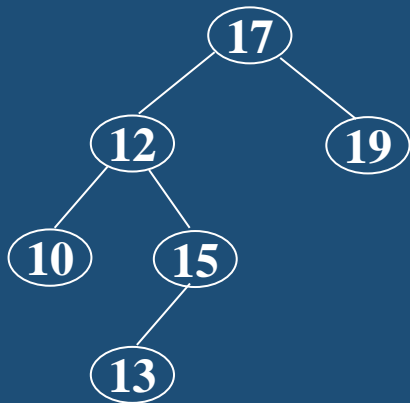
1. 二叉排序树查找

要在二叉树中找出查找最大最小元素是极简单的事情，从根节点一直往左走，直到无路可走就可得到最小值；从根节点一直往右走，直到无路可走，就可以得到最大值。



2. 二叉排序树插入

插入新元素时，可以从根节点开始，遇键值较大者就向左，遇键值较小者就向右，一直到末端，就是插入点。



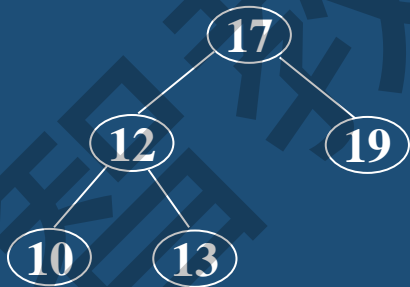
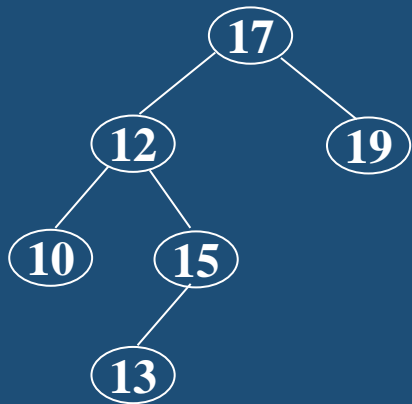
插入新值16。

从根结点开始，遇键值较大的则向左，遇键值教小的则向右，直到尾端，即插入点

3. 二叉排序树删除

对于二叉排序树中的节点A，对它的删除分为两种情况：

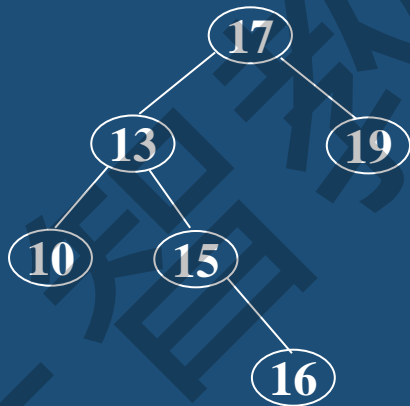
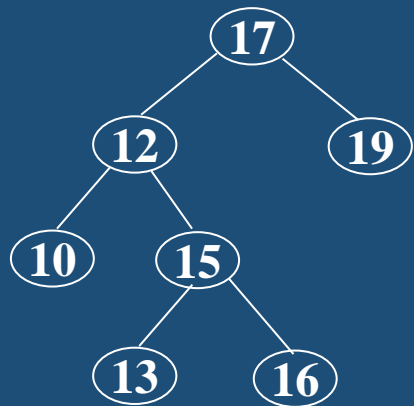
- (1) 如果A只有一个子节点，就直接将A的子节点连至A的父节点上，并将A删除；



删除旧值15（只有一个子结点）。
直接将其子节点连至父节点

3. 二叉排序树删除

如果A有两个子节点，我们就以右子树内的最小节点取代A。



删除旧值12（有两个子结点）

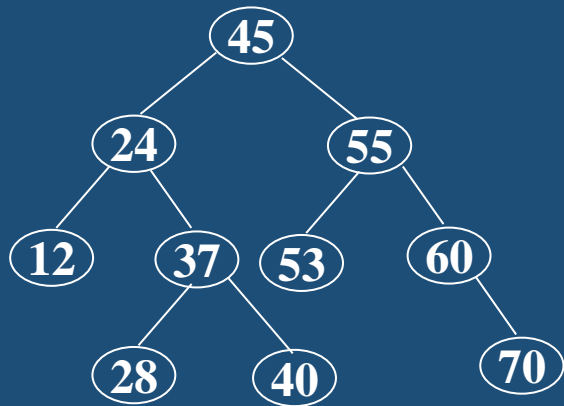
以右子树中的最小值取而代之。

注意，最小值极易获得：一直向左走到底就是

4. 二叉排序树性能分析

对于高度为 H 的二叉排序树，其插入和删除操作的运行时间都为 $O(H)$ 。但在最坏的情况下，即构造二叉排序树的输入序列是有序的，则会形成一个倾斜的单支树，此时二叉排序树的性能显著变坏，树的高度也增加为元素个数 N 。

二叉排序树的查找



如图，在等概率的情况下，（a）的查找成功的平均查找长度为

$$ASL = (1 + 2 \times 2 + 3 \times 4 + 4 \times 3) / 10 = 2.9$$

（b）的查找成功的平均查找长度为

$$ASL = (1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10) / 10 = 5.5$$



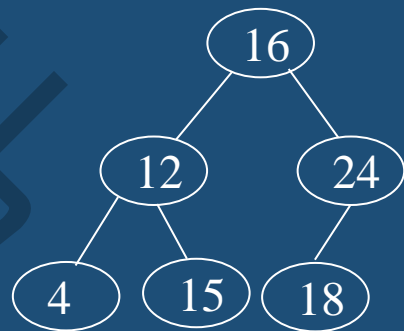
二叉排序树的查找

由上可知，二叉排序树查找算法的平均查找长度，主要取决于树的高度，即与二叉树的形态又关。如果二叉排序树是一个单支树，其平均查找长度与单链表相同，为 $O(n)$ 。如果二叉排序树的左右子树的高度差不超过1，这样的二叉排序树称为平衡二叉树。它的平均查找长度达到 $O(\log_2 n)$ 。

平衡二叉树

平衡二叉树或者是空树，或者是满足下列性质的二叉树。

- (1): 左子树和右子树深度之差的绝对值不大于1;
- (2): 左子树和右子树也都是平衡二叉树。



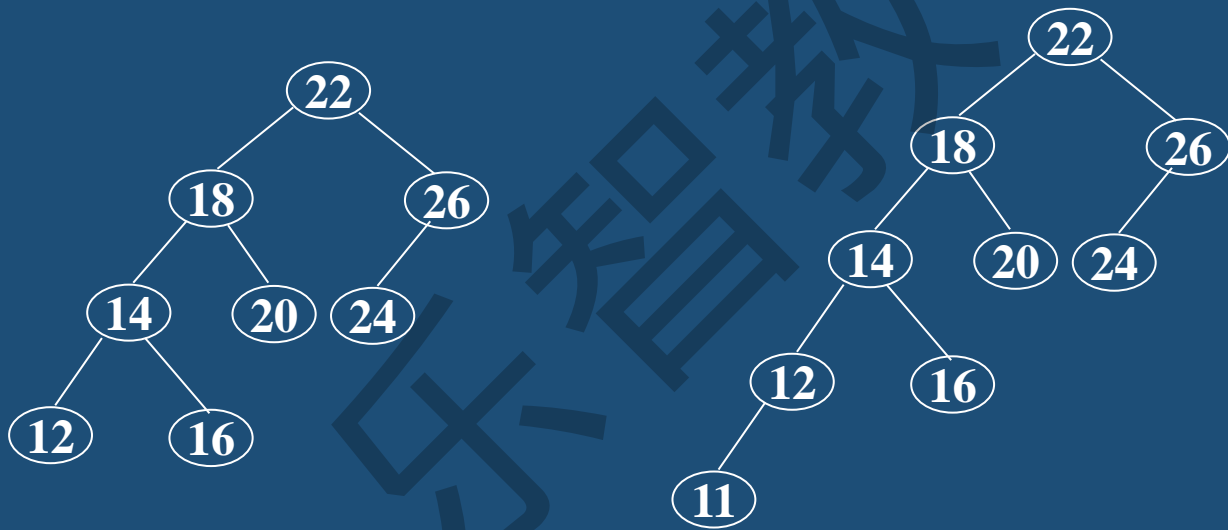
平衡因子(Balance Factor)：二叉树上结点的左子树的深度减去其右子树深度称为该结点的平衡因子。

因此，平衡二叉树上每个结点的平衡因子只可能是-1、0和1，否则，只要有一个结点的平衡因子的绝对值大于1，该二叉树就不是平衡二叉树。

如果一棵二叉树既是二叉排序树又是平衡二叉树，称为**平衡二叉排序树(Balanced Binary Sort Tree)**。

1. 平衡化旋转

一般的二叉排序树是不平衡的，若能通过某种方法使其**既保持有序性，又具有平衡性**，就找到了构造平衡二叉排序树的方法，该方法称为**平衡化旋转**。

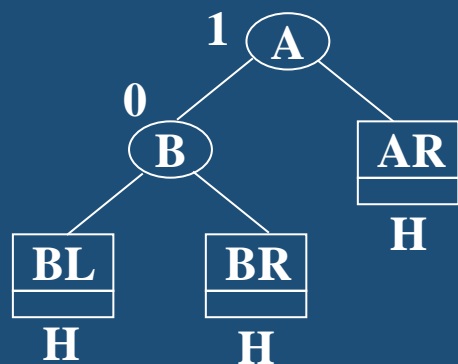


1. 平衡化旋转

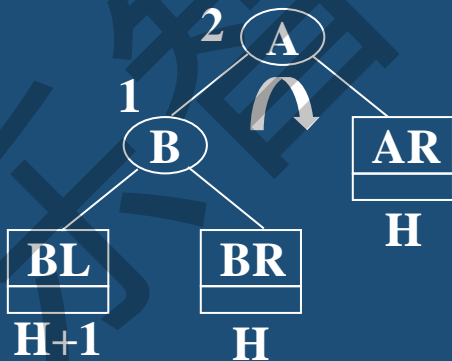
LL型平衡化旋转

LL平衡旋转（右单选择）。由于在结点A的左孩子（L）的左子树（L）上插入了新结点，A的平衡因子由1增至2，导致以A的子树失去平衡，需要一次向右的旋转操作。将A的左孩子B右上旋转代替A成为根结点，将A结点向右下旋转成为B的右子树的根结点，而B的原右子树则作为A结点的左子树。

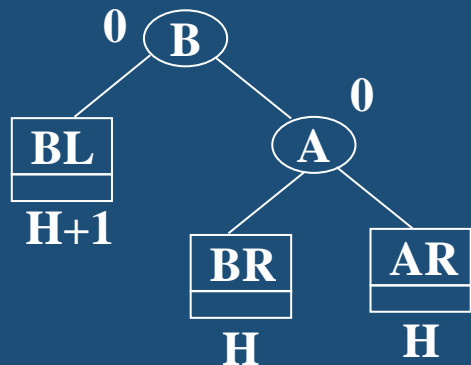
结点旁的数值代表结点的平衡因子，而用方块表示相应结点的子树，下方数值代表该子树的高度。



(1) 插入结点前



(2) 插入结点导致不平衡

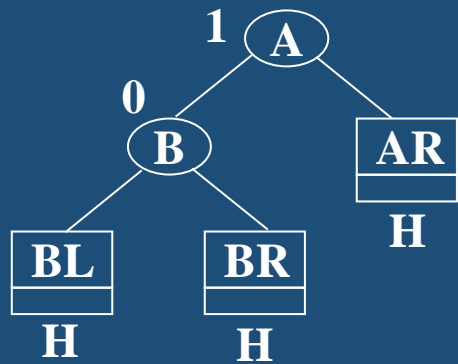


(3) LL旋转

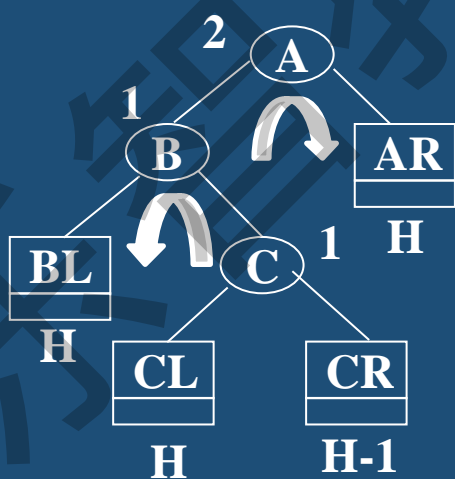
1. 平衡化旋转

LR型平衡化旋转

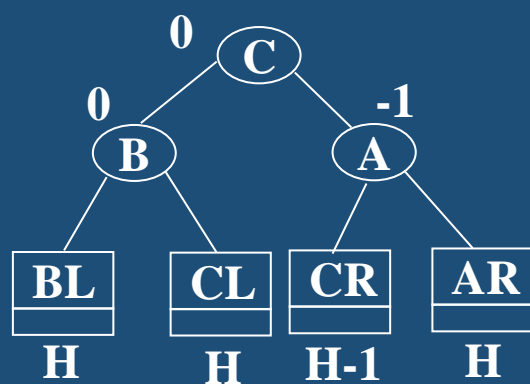
LR平衡旋转（先左后右双旋转）。由于在A的左孩子（L）的右子树（R）上插入了新结点，A的平衡因子由1增至2，导致以A为根失去平衡，需要进行两次旋转操作。先左旋转后右旋转。先将A结点的左孩子B的右子树的根结点C向左上旋转提升到B结点的位置，然后再把该C结点向右上旋转提升到A结点的位置。



(1) 插入结点前



(2) 插入结点导致不平衡

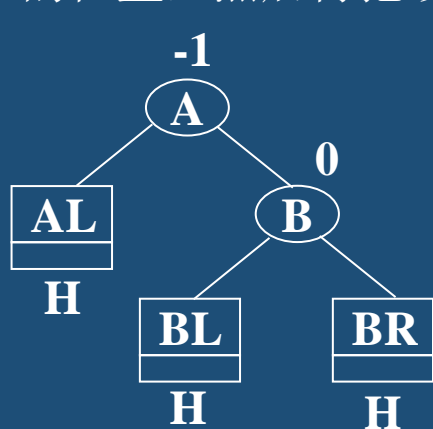


(3) LR旋转

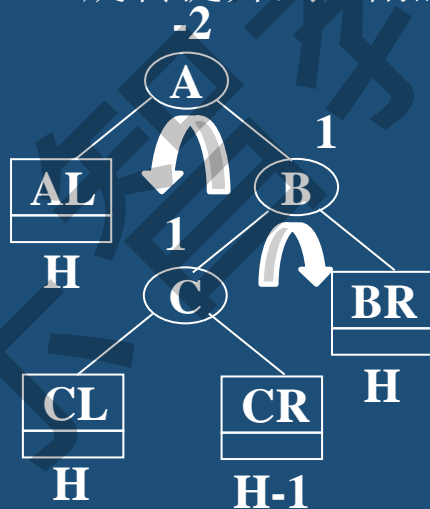
1. 平衡化旋转

RL型平衡化旋转

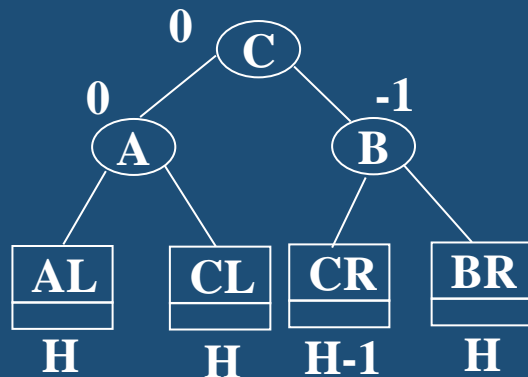
RL平衡旋转（先右后左双旋转）。由于在A的右孩子（R）的左子树（L）上插入了新结点，A的平衡因子由-1减至-2，导致以A为根的子树失去平衡，需要进行两次旋转操作。先右旋转后左旋转。先将A结点的右孩子B的左子树的根结点C向右上旋转提升到B结点的位置，然后再把该C结点向左上旋转提升到A结点的位置。



(1) 插入结点前



(2) 插入结点导致不平衡

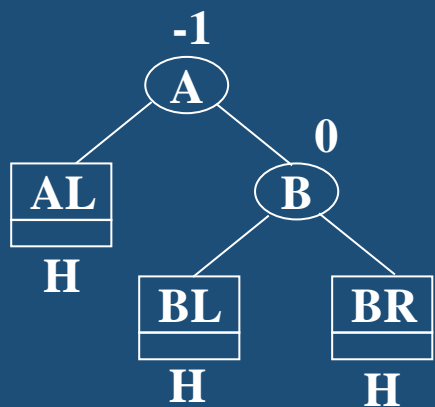


(3) RL旋转

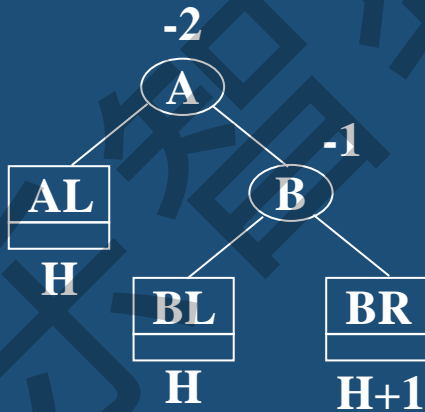
1. 平衡化旋转

RR型平衡化旋转

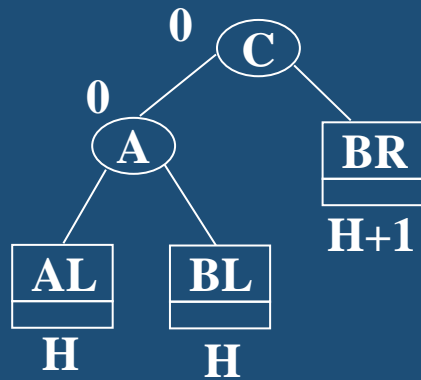
RR平衡旋转（左单旋转）。由于在A的右孩子（R）的右子树（R）上插入了新结点，A的平衡因子由-1减至-2，导致以A为根的子树失去平衡，需要一次向左的旋转操作。将A的右孩子B向左上旋转代替A成为根结点，将A结点向左下旋转成为B的左子树的根结点，而B的原左子树则作为A结点的右子树。



(1) 插入结点前



(2) 插入结点导致不平衡



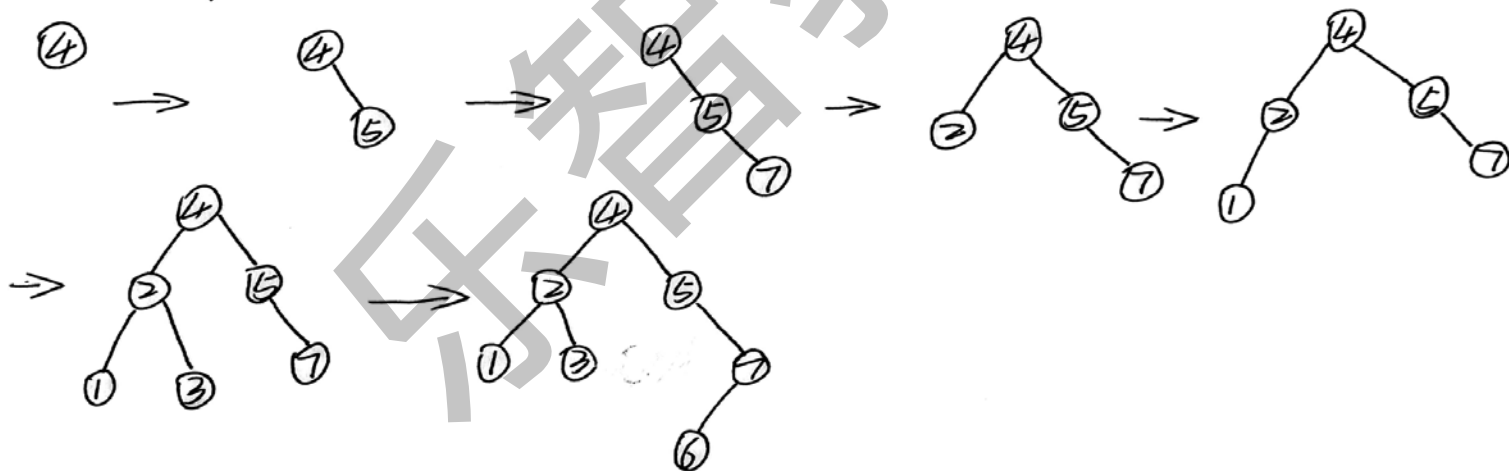
(3) RR旋转

查找（真题检测）

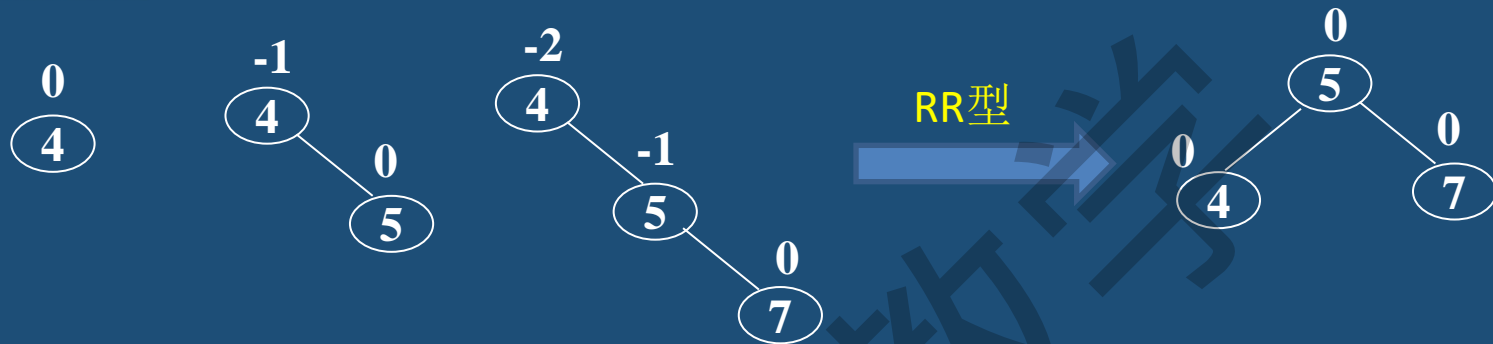
1. 有一组关键字为（4,5,7,2,1,3,6），根据所给的关键字分别进行以下操作：

- ① 按顺序建立一棵二叉排序树，画出该二叉排序树；
- ② 按顺序建立一棵平衡二叉排序树，画出该平衡二叉排序树。

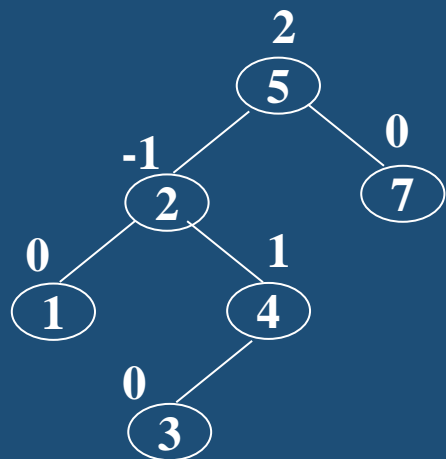
① 二叉排序树



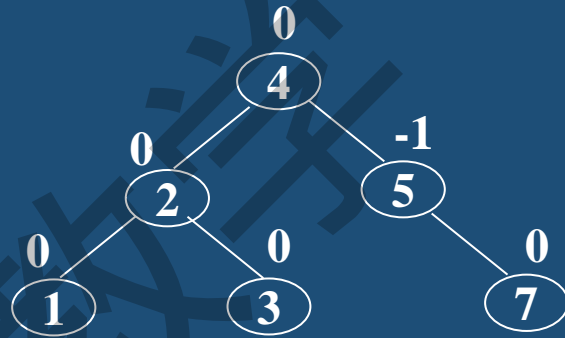
查找 (真题检测)



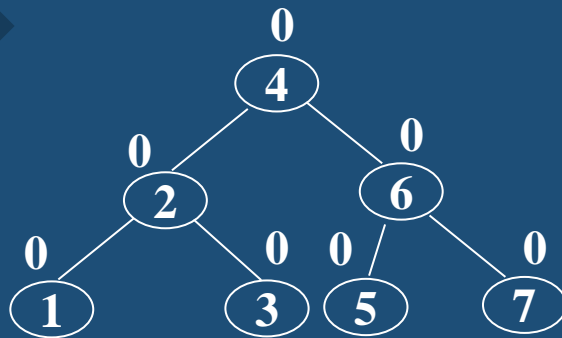
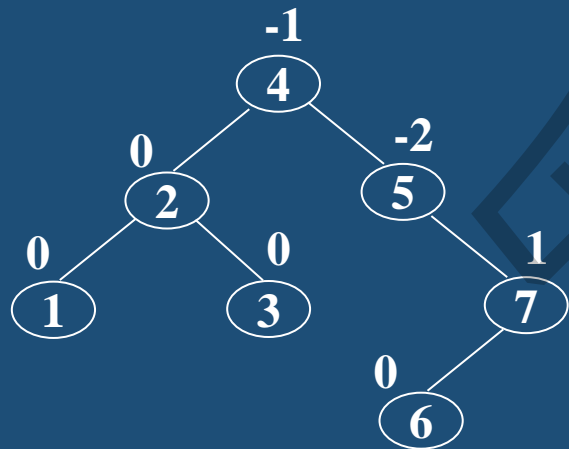
查找 (真题检测)



LR型



RL型



1 基本思想：在记录的存储地址和它的关键字之间建立一个确定的对应关系；这样，不经过比较，一次存取就能得到所查元素的查找方法。

例 30个地区的各民族人口统计表

编号	省、市(区)	总人口	汉族	回族
1	北京				
2	上海				
.....				

以编号作关键字，构造哈希函数：

$H(\text{key}) = \text{key}$

$H(1)=1$, $H(2)=2$

以地区别作关键字，取地区名称第一个拼音字母的序号

作哈希函数： $H(\text{Beijing})=2$ $H(\text{Shanghai})=19$

$H(\text{Shenyang})=19$

哈希函数：在记录的关键字与记录的存储地址之间建立的一种对应关系叫哈希函数。

哈希函数是一种映象，是从关键字空间到存储地址空间的一种映象。可写成： $\text{addr}(\text{ai})=\text{H}(\text{ki})$ ，其中 i 是表中一个元素， $\text{addr}(\text{ai})$ 是 ai 的地址， ki 是 ai 的关键字。

哈希表：应用哈希函数，由记录的关键字确定记录在表中的地址，并将记录放入此地址，这样构成的表叫**哈希表**。

哈希查找(又叫散列查找)：利用哈希函数进行查找的过程叫**哈希查找**。

冲突：对于不同的关键字 ki 、 kj ，若 $\text{ki} \neq \text{kj}$ ，但 $\text{H}(\text{ki})=\text{H}(\text{kj})$ 的现象叫冲突(collision)。

同义词：具有相同函数值的两个不同的关键字，称为该哈希函数的**同义词**。

哈希函数通常是一种压缩映象，所以冲突不可避免，只能尽量减少；当冲突发生时，应该有处理冲突的方法。设计一个散列表应包括：

- ① 散列表的空间范围，即确定散列函数的值域；
- ② 构造合适的散列函数，使得对于所有可能的元素(记录的关键字)，函数值均在散列表的地址空间范围内，且出现冲突的可能尽量小；
- ③ 处理冲突的方法。即当冲突出现时如何解决。

2 哈希函数的构造

哈希函数是一种映象，其设定很灵活，只要使任何关键字的哈希函数值都落在表长允许的范围之内即可。哈希函数“好坏”的主要评价因素有：

- ◆ 散列函数的构造简单；
- ◆ 能“均匀”地将散列表中的关键字映射到地址空间。所谓“均匀”(uniform)是指发生冲突的可能性尽可能最少。

(1) 直接定址法

取关键字或关键字的某个线性函数作哈希地址，即 $H(\text{key})=\text{key}$ 或 $H(\text{key})=a \cdot \text{key} + b$ (a, b 为常数)

特点：直接定址法所得地址集合与关键字集合大小相等，不会发生冲突，但实际中很少使用。

(2) 数字分析法

对关键字进行分析，取关键字的若干位或组合作为哈希地址。
适用于关键字位数比哈希地址位数大，且可能出现的关键字事先知道的情况。

(3) 除留余数法

取关键字被某个不大于哈希表表长 m 的数 p 除后所得余数作哈希地址，即

$$H(\text{key}) = \text{key} \bmod p \quad (p \leq m)$$

是一种简单、常用的哈希函数构造方法。

利用这种方法的关键是 p 的选取， p 选的不好，容易产生同义词。 p 的选取的分析：

◆ 选取 $p=2^i (p \leq m)$ ：运算便于用移位来实现，但等于将关键字的高位忽略而仅留下低位二进制数。高位不同而低位相同的关键字是同义词。

◆ 选取 $p=q \times f (q, f \text{ 都是质因数}, p \leq m)$ ：则所有含有 q 或 f 因子的关键字的散列地址均是 q 或 f 的倍数。

◆ 选取 p 为素数或 $p=q \times f (q, f \text{ 是质数且均大于} 20, p \leq m)$ ：常用的选取方法，能减少冲突出现的可能性。

3 冲突处理的方法

冲突处理：当出现冲突时，为冲突元素找到另一个存储位置。

(1) 开放定址法

基本方法：当冲突发生时，形成某个探测序列；按此序列逐个探测散列表中的其他地址，直到找到给定的关键字或一个空地址(开放的地址)为止，将发生冲突的记录放到该地址中。散列地址的计算公式是：

$$H_i(\text{key}) = (H(\text{key}) + d_i) \text{ MOD } m, \quad i=1, 2, \dots, k(k \leq m-1)$$

其中：H(key)：哈希函数；m：散列表长度；

d_i ：第i次探测时的增量序列；

$H_i(\text{key})$ ：经第i次探测后得到的散列地址。

<1> 线性探测法

将散列表 $T[0 \dots m-1]$ 看成循环向量。当发生冲突时，从初次发生冲突的位置依次向后探测其他的地址。

增量序列为： $d_i=1, 2, 3, \dots, m-1$

设初次发生冲突的地址是 h ，则依次探测 $T[h+1]$ ， $T[h+2]$...，直到 $T[m-1]$ 时又循环到表头，再次探测 $T[0]$ ， $T[1]$...，直到 $T[h-1]$ 。探测过程终止的情况是：

- ◆ **探测到的地址为空**：表中没有记录,若是查找则失败；若是插入则将记录写入到该地址；
- ◆ **探测到的地址有给定的关键字**：若是查找则成功；若是插入则失败；
- ◆ 直到 $T[h]$ ：仍未探测到空地址或给定的关键字，**散列表满**。

例1：设散列表长为7，记录关键字组为：15, 14, 28, 26, 56, 23，散列函数：
 $H(\text{key}) = \text{key} \bmod 7$ ，冲突处理采用线性探测法。

解： $H(15) = 15 \bmod 7 = 1$ $H(14) = 14 \bmod 7 = 0$

$H(28) = 28 \bmod 7 = 0$ 冲突 $H_1(28) = 1$ 又冲突 $H_2(28) = 2$

$H(26) = 26 \bmod 7 = 5$

$H(56) = 56 \bmod 7 = 0$ 冲突 $H_1(56) = 1$ 又冲突 $H_2(56) = 2$ 又冲突

$H_3(56) = 3$

$H(23) = 23 \bmod 7 = 2$ 冲突 $H_1(23) = 3$ 又冲突

$H_3(23) = 4$

0	1	2	3	4	5	6
14	15	28	56	23	26	

散列表

线性探测法的特点

- ◆ **优点**：只要散列表未满，总能找到一个不冲突的散列地址；
- ◆ **缺点**：每个产生冲突的记录被散列到离冲突最近的空地址上，从而又增加了更多的冲突机会(这种现象称为冲突的“聚集”)。

散列表

<2> 二次探测法

增量序列为: $d_i = 1^2, -1^2, 2^2, -2^2, 3^2, \dots, \pm k^2$ ($k \leq \lfloor m/2 \rfloor$)

上述例题若采用二次探测法进行冲突处理, 则:

$$H(15) = 15 \text{ MOD } 7 = 1$$

$$H(14) = 14 \text{ MOD } 7 = 0$$

$$H(28) = 28 \text{ MOD } 7 = 0$$

冲突

$$H_1(28) = 1$$

又冲突

$$H_2(28) = 6$$

$$H(26) = 26 \text{ MOD } 7 = 5$$

$$H(56) = 56 \text{ MOD } 7 = 0$$

冲突

$$H_1(56) = 1$$

又冲突

$$H_2(56) = 6$$

又冲突

$$H_3(56) = 4$$

$$H(23) = 23 \text{ MOD } 7 = 2$$

0	1	2	3	4	5	6
14	15	23		56	26	28

二次探测法的特点

- ◆ 优点: 探测序列跳跃式地散列到整个表中, 不易产生冲突的“聚集”现象;
- ◆ 缺点: 不能保证探测到散列表的所有地址。

<3> 再散列法

构造若干个哈希函数，当发生冲突时，利用不同的哈希函数再计算下一个新哈希地址，直到不发生冲突为止。即： $H_i = RH_i(\text{key}) \quad i=1, 2, \dots, k$

RH_i ：一组不同的哈希函数。第一次发生冲突时，用 RH_1 计算，第二次发生冲突时，用 RH_2 计算...依此类推知道得到某个 H_i 不再冲突为止。

- ◆ 优点：不易产生冲突的“聚集”现象；
- ◆ 缺点：计算时间增加。

(2) 链地址法

方法：将所有关键字为同义词(散列地址相同)的记录存储在一个单链表中，并用一维数组存放链表的头指针。

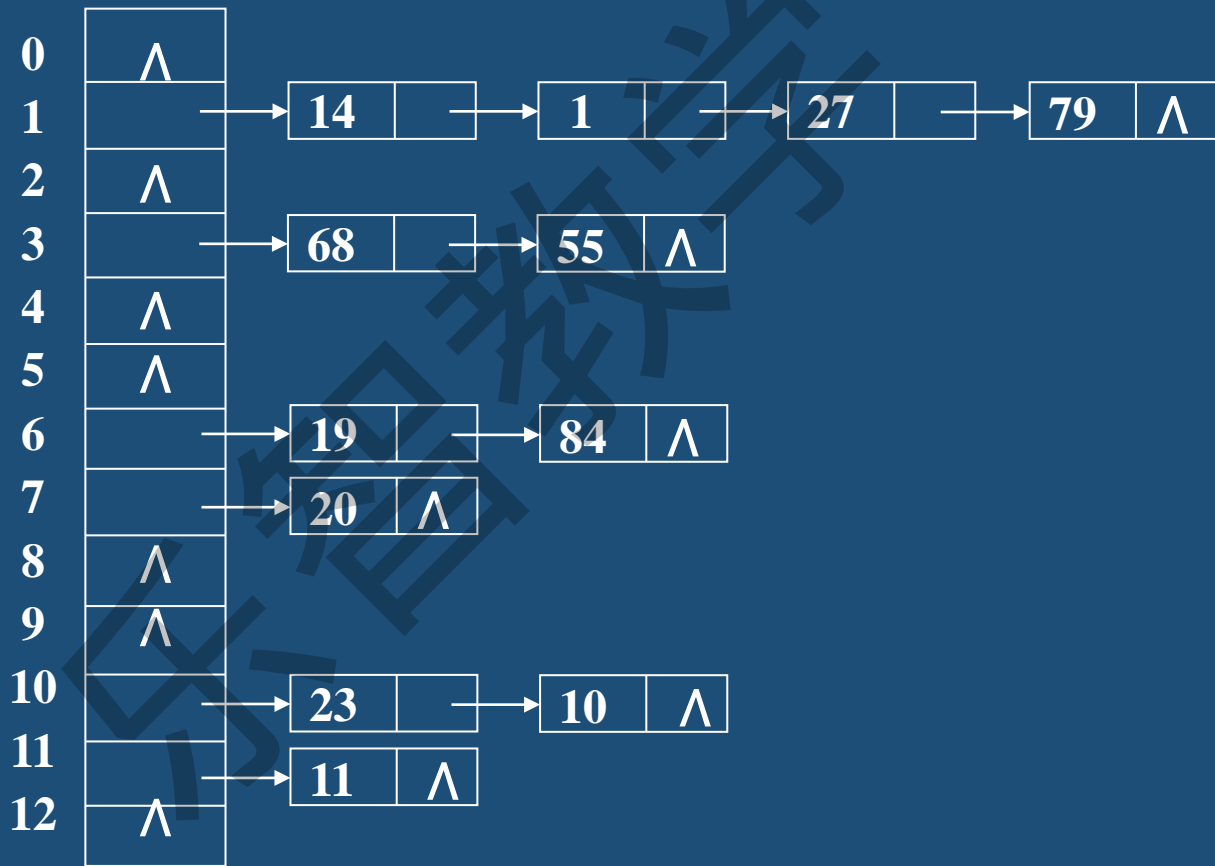
设散列表长为 m ，定义一个一维指针数组：

$\text{RecNode} * \text{linkhash}[m]$ ，其中 RecNode 是结点类型，每个分量的初值为空。凡散列地址为 k 的记录都插入到以 $\text{linkhash}[k]$ 为头指针的链表中，插入位置可以在表头或表尾或按关键字排序插入。

例： 已知一组关键字(19, 14, 23, 1, 68, 20, 84, 27, 55, 11, 10, 79)， 哈希函数为： $H(\text{key}) = \text{key} \bmod 13$ ，用链地址法处理冲突，如右图图9-17所示。

优点：不易产生冲突的“聚集”；删除记录也很简单。

散列表



4 散列查找性能分析

从哈希查找过程可见：尽管散列表在关键字与记录的存储地址之间建立了直接映象，但由于“冲突”，查找过程仍是一个给定值与关键字进行比较的过程，评价哈希查找效率仍要用ASL。

哈希查找时关键字与给定值比较的次数取决于：

- ◆ 哈希函数；
- ◆ 处理冲突的方法；
- ◆ 哈希表的填满因子 α 。填满因子 α 的定义是：

$$\alpha = \frac{\text{表中填入的记录数}}{\text{哈希表长度}}$$

各种散列函数所构造的散列表的ASL如下:

(1) 线性探测法的平均查找长度是:

$$S_{nl成功} \approx \frac{1}{2} \times (1 + \frac{1}{1-\alpha})$$

$$S_{nl失败} \approx \frac{1}{2} \times (1 + \frac{1}{(1-\alpha)^2})$$

(2) 二次探测、再哈希法的平均查找长度是:

$$S_{nl成功} \approx -\frac{1}{\alpha} \times \ln(1-\alpha)$$

$$S_{nl失败} \approx \frac{1}{1-\alpha}$$

(3) 用链地址法解决冲突的平均查找长度是:

$$S_{nl成功} \approx 1 + \frac{\alpha}{2}$$

$$S_{nl失败} \approx \alpha + e^{-\alpha}$$

查找（真题检测）

2. 有一组关键字序列{15,92,124,5,27,28,18,6,36,34,30,26,32,259},将它们用散列函数 $H(\text{key})=\text{key} \bmod 10$ 按顺序散列到哈希表HT(0:9)中,用链地址法解决冲突,画出最终的哈希表,并求在等概率情况下查找成功和不成功时的平均查找长度。

$$H(15)=15 \bmod 10=5$$

$$H(5)=5 \bmod 10=5 \text{ 冲突}$$

$$H(18)=18 \bmod 10=8 \text{ 冲突}$$

$$H(34)=34 \bmod 10=4$$

$$H(32)=32 \bmod 10=2 \text{ 冲突}$$

$$H(92)=92 \bmod 10=2$$

$$H(27)=27 \bmod 10=7$$

$$H(6)=6 \bmod 10=6$$

$$H(30)=30 \bmod 10=0$$

$$H(259)=259 \bmod 10=9$$

$$H(124)=124 \bmod 10=4$$

$$H(28)=28 \bmod 10=8$$

$$H(36)=36 \bmod 10=6 \text{ 冲突}$$

$$H(26)=26 \bmod 10=6 \text{ 冲突}$$

查找 (真题检测)

故链地址处理冲突的散列表如下：

0	1	2	3	4	5	6	7	8	9
30		92		124	15	6	27	28	259

↓

32

↓

34

↓

5

↓

36

 ↓

26

↓

18

由上图可知， 填充因子 α 为： $\alpha = \frac{\text{表中记录数}_n}{\text{散列表长度}_m} = \frac{8}{10} = \frac{4}{5}$

故查找成功的平均查找长度为： $S_{\text{成功}} = 1 + \frac{\alpha}{2} = \frac{7}{5}$

故查找失败的平均查找长度为： $S_{\text{失败}} = \alpha + e^{-\alpha} = \frac{4}{5} + e^{-\frac{4}{5}}$

谢谢观看