第三章

查找与排序技术

3.1 基本的查找技术

查找的基本概念

就是在给定的数据结构中查找某个指定的元素（结点）；若存在这样的结点，查找成功；否则，查找失败。

一、 平均查找长度

- 在查找过程中，要对每个结点记录中的关键字进行反复比较，以确定其位置。因此，与关键字进行比较的平均次数，就称为平均查找长度。是用来评价算法好坏的一个依据。(算法的评价)
- 对含有n个数据元素的查找表，查找成功时的平均查找长度为：

$$ASL = \sum_{i=1}^{n} P_i * C_i$$

其中：P_i 为查找表中第i个数据元素的概率，且

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

C_i为查找第i个数据元素时需比较的次数。

C_i随查找过程及DS的不同而各异。（Ci与方法和结构有关）

二、主要查找算法

(1) 顺序查找

(2) 对分（折半）查找

(3) 分块查找

3.1.1 顺序查找

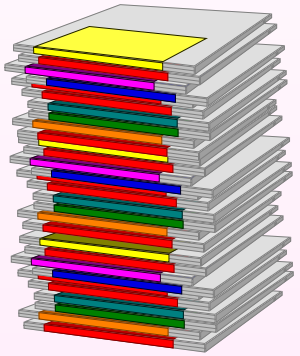
- 算法思想：

  从第1个元素到最后1个元素，逐个进行比较。

- 特点：

  最简单、最普通的查找方法。

- 适用范围（查找表的存储结构）：
  - 既适用于顺序存储结构
  - 也适用于链式存储结构



- 操作步骤：
  - step1 从第1个元素开始查找；
  - step2 用待查关键字值与各结点（记录）的关键字值逐个进行比较；若找到相等的结点，则查找成功；否则，查找失败。

逐个比较

```
int seq_search(item , n , key )
int *item ,n , key ;
{  int i=0 ;
   while ( i < n && item[i] != key )
      i++;
   if (item[i]= = key )
   {   printf("查找成功 !\n"); return (i); }
   else
   { printf("查找失败 !\n"); return (-1);}
}
```

item　待查表
n　元素个数
key 要找的值

While中，有两个判断条件，改进一下，可以减少一半

（改进算法）

```
int seq_search_adv(item , n , key )
  int *item ,n , key ;
  {  int i=0 ;                                  //？哨兵的作用？
     item[n]=key ;                               /* 设置哨兵   */
     while ( item[i]!= key ) i++;  /* 查找key  */
     if ( i< n )                                 /*   如果 i = n，没找到 */
     {   printf("查找成功 !\n"); return (i); }
     else
     {   printf("查找失败 !\n"); return (-1); }
   }
```

注: 顺序查找算法中,执行频率最高的是while语句,改进后,可以节省近一半的时间。?改进后与改进前的不同之处？

主要是取消了原来i<n的判断

算法评价

- 平均查找长度ASL在等概率的情况下

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

- 一般情况下 $ASL \le \frac{n}{2}$
- 优点：
  - 对结点的逻辑次序(不必有序)和存储结构(顺序、链表均可）无要求；
  - 当N值较小时，是较好的算法；
- 缺点：ASL较长
- 讨论：能否减少比较次数，以提高效率。
- 例如：……二分法等，（比较学号查找和姓名查找）

3.1.2 对分查找

- 1. 基本思想：

  将有序序列的中点设置为比较对象，如果要找的元素值小于该中点元素，则将待查序列缩小为左半部分，否则为右半部分。

  即通过一次比较，将查找区间缩小一半。

- 2. 特点：

  二分查找是一种高效的查找方法。它可以明显减少比较次数，提高查找效率。但是，二分查找的先决条件是查找表中的数据元素必须有序。

一、算法描述

- 算法步骤：
  - step1 首先确定整个查找区间的中间位置，

    mid = int((left + right) / 2)
  - step2 用待查关键字值与中间位置的关键字值进行比较；
    - 若相等，则查找成功；
    - 若大于，则在后半区域继续进行二分查找；
    - 若小于，则在前半区域继续进行二分查找。
  - Step3 对确定的缩小区域再按二分公式，重复上述步骤；
  - 最后，得到结果：
    - 要么，查找成功， 要么，查找失败。
- 存储结构
  - 用一维数组存放。

对分查找算法举例

- 对给定数列（有序）{3，5，11，17，21，23，28，30，32，50}，共10个数，按折半查找算法，查找关键字值为30的数据元素。找30：

第1次：{3，5，11，17，21，23，28，30，32，50}

left　　mid　　right

mid1= int（1+10）/2 = 5

第2次：{ 23，28，30，32，50 }

left　　mid　　right

mid2 = int（6+10）/2 = 8

二、算法

```
int bin_search ( item , n ,key )
int *item, n, key;       //n为元素个数
{   int left ,right , mid; left=0;   right = n-1;
    while ( left ≤ right )
    {
      mid = ( left + right )/2 ; /*  计算中点位置   */
      if ( key < item[mid])
         right = mid - 1;      /*  待查区间在左部  */
      else if (key > item[mid])
         left = mid + 1;       /*  待查区间在右部  */
      else{ printf ( " Successful search\n");
          return mid ;         /*  查找成功   */
    }
    printf( " Search failure \n");
    return -1;                 /*  查找失败  */
}
```

三、算法评价

- 优点: ASL $\leq \log_2 n$； 即每经过一次比较,查找范围就缩小一半。经$\log_2 n$次比较就可以完成查找过程。
- 缺点：因要求有序，所以对所有数据元素按大小排序是非常费时的操作。另外，顺序存储结构的插入、删除操作不大便利。

3.1.3 分块查找

- 分块查找又称索引顺序查找，这是顺序查找的一种改进方法，用于在分块有序表中进行查找 。
- (1)分块：将n个数据元素“按块有序”划分为m块（m ≤ n）。每一块中的结点不必有序，但块与块之间必须“按块有序”；即第1块中任一元素的关键字都必须小于第2块中任一元素的关键字；而第2块中任一元素又都必须小于第3块中的任一元素，……。每个块中元素不一定是有序的。

一、分块查找算法描述

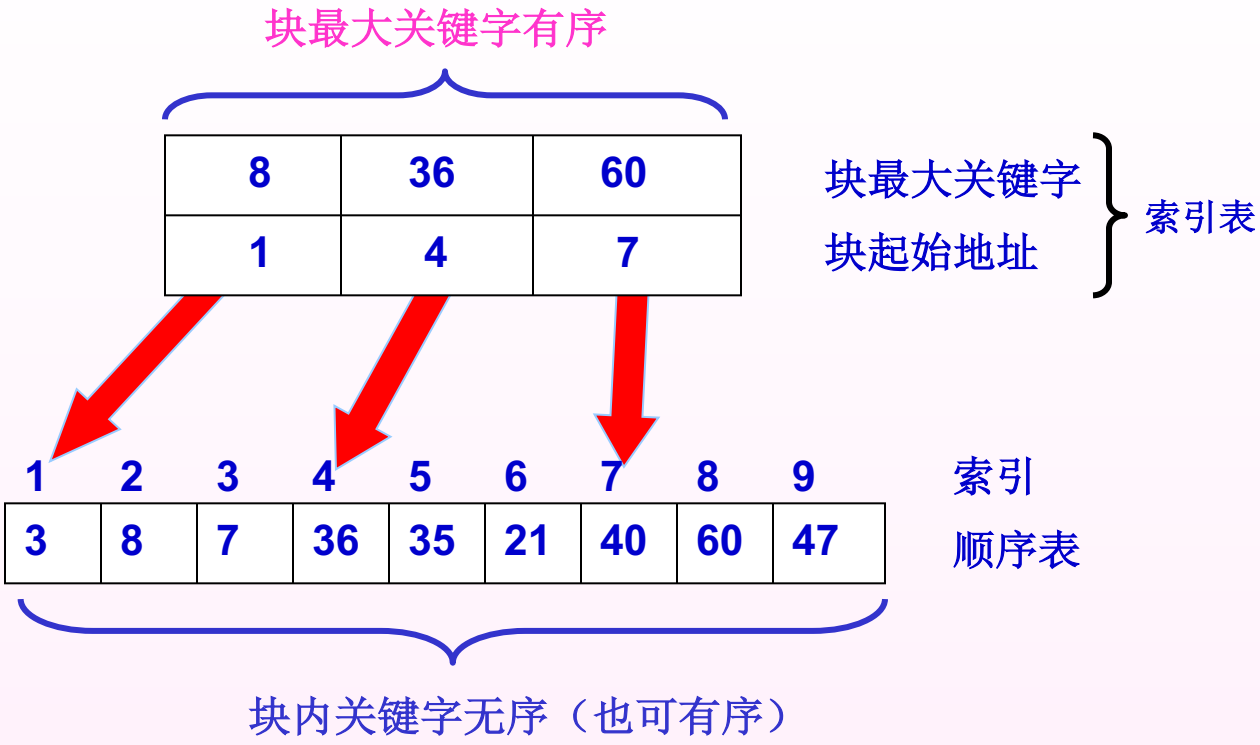
1.算法描述:

- step1 先选取各块中的最大关键字构成一个索引表；
- step2 查找分两个部分：

- 先对索引表进行二分查找或顺序查找，以确定待查记录在哪一块中；
- 在已确定的块中用顺序法进行查找。

2.分块查找数据描述

- 将查找表分成3块，每块3个元素



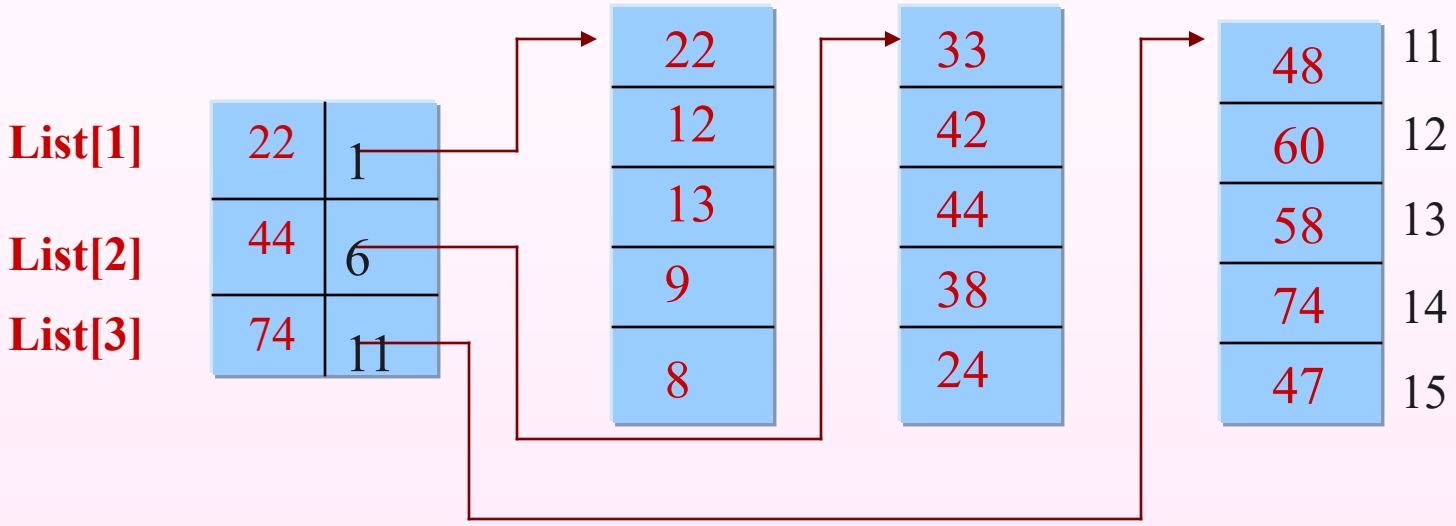
分块查找举例

- 有数列如下：

{ 22,12,13,9,8,33,42,44,38,24,48,60,58,74,47}

按“块有序”分三块:(22,12,13,9,8),(33,42,44,38,24),(48,60,58,74,47)。选取每块中最大的关键字组成索引表[22,44,74],查找关键字值为60的元素。

- 用二分法，确定在索引表中的位置为 mid=2，key值60与a[2]比较，60>a[2],取第3个块;在第3块中用顺序法查找,比较两次,就可以找出60的元素来。



二、算法评价

- 设索引表使用二分法，则有：$\mathrm{ASL} \le \log_2\left(\frac{n}{s}+1\right)+\frac{s}{2}$

  其中：n为表长，S为块长（假设各块长度相等）。

- 优点：
  - 插入、删除操作方便；
  - 只要找到对应的块，在块中任意位置操作均可。
- 缺点：

  索引表增加了辅助存储空间。

  注： 索引表在数据库系统中广泛使用。

总结:

- 查找 就是在给定的DS中找出满足某种条件的结点；若存在这样的结点，则查找成功；否则，查找失败。（找）
- 查找表 是一组待查数据元素的集合。（待找）
- 静态查找 是仅仅进行查询和检索操作，不改变查找表中数据元素间的逻辑关系的查找。（不改变元素关系）
- 动态查找 是除了进行查询和检索操作外，还对查找表进行插入、删除操作的查找，动态地改变查找表中数据元素之间的逻辑关系。（改变元素关系）
- 平均查找长度

与关键字进行比较的平均次数。对含有n个数据元素的查找表，查找成功时的平均查找长度为

$$ASL = \sum_{i=1}^{n} Pi * Ci$$

- Pi 为查找第i个数据元素的概率
- Ci为查找第i个数据元素的比较次数。

3.2 哈希（Hash）表技术

- 哈希查找也称为散列查找。它不同于前面介绍的几种查找方法。上述方法都是把查找建立在比较的基础上，而哈希查找则是通过计算存储地址的方法进行查找的。
- 计算是计算机的特点之一，因此，建立在计算基础上的哈希查找是一种快速查找方法。

3.2.1哈希表基本概念

一、直接查找技术

设表的长度为n。如果存在一个函数i=i（k），对于表中的任意一个元素的关键字k，满足以下条件：

（1）1≤i≤n。

（2）对于任意元素的关键字 $k_1 \neq k_2$，恒存在 $i(k_1) \neq i(k_2)$

则称此表为直接查找表，其中函数i＝i（k）称为关键字k的映像函数。

直接查找表的操作主要有以下两种：

- 直接查找表的填入
- 直接查找表的取出

二、Hash表

- 设表的长度为n。如果存在一个函数i=i（k），对于表中的任意一个元素的关键字k满足：　1≤i≤n;

  则称此表为Hash表。其中，函数i=i(k)称为关键字k的Hash码。

- 将关键字序列（09，31，26，19，01，13，02，11，27，16，05，21）依次填入长度为n＝12的表中。设映像函数为 i＝INT（k／3）＋1，其中INT为取整符。

- Hash表技术的关键是要处理好表中元素的冲突问题，它主要包括以下两方面的工作：

（1）构造合适的Hash码，以便尽量减少表中元素冲突的次数。即Hash码的均匀性要比较好。

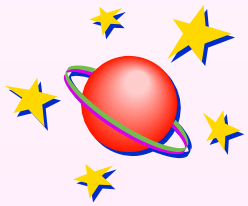
（2）当表中元素发生冲突时，要进行适当的处理。

- 建立哈希函数的原则
  - 均匀性：H（key）的值均匀分布在哈希表中；
  - 简　单：以提高地址计算的速度。

（位置均匀，计算简单）

三、 哈希函数常用的构造方法

- 截段法
- 分段叠加法
- 除法（求模取余法）
- 乘法



冲突及冲突处理

由于哈希函数是一个压缩映像，因此，在一般情况下，很容易产生“冲突”现象

- 在哈希元素（地址）求解过程中，不同关键字值对应到同一个存储地址的现象称为冲突。即关键字K1 ≠ K2， 但哈希函数值 H（K1）= H（K2）。
- 均匀的哈希函数可以减少冲突，但不能避免冲突。发生冲突后，必须解决；也即必须寻找下一个可用地址。
- 处理冲突是建立哈希表过程中不可缺少的一部分。
- 处理冲突主要有两种方法：
  - 开放地址法
  - 链地址法

*处理冲突 —— 开放地址法

- 当发生地址冲突后，求解下一个地址用

  Hi =(H（key）+di） MOD m

  i=1,2,...,k(k ≤ m-1)

  其中：H(key)为哈希函数,m为哈希表长度,di为增量序列。增量序列的不同取法，又构成不同的开放地址法。

  ① 线性探测再散列

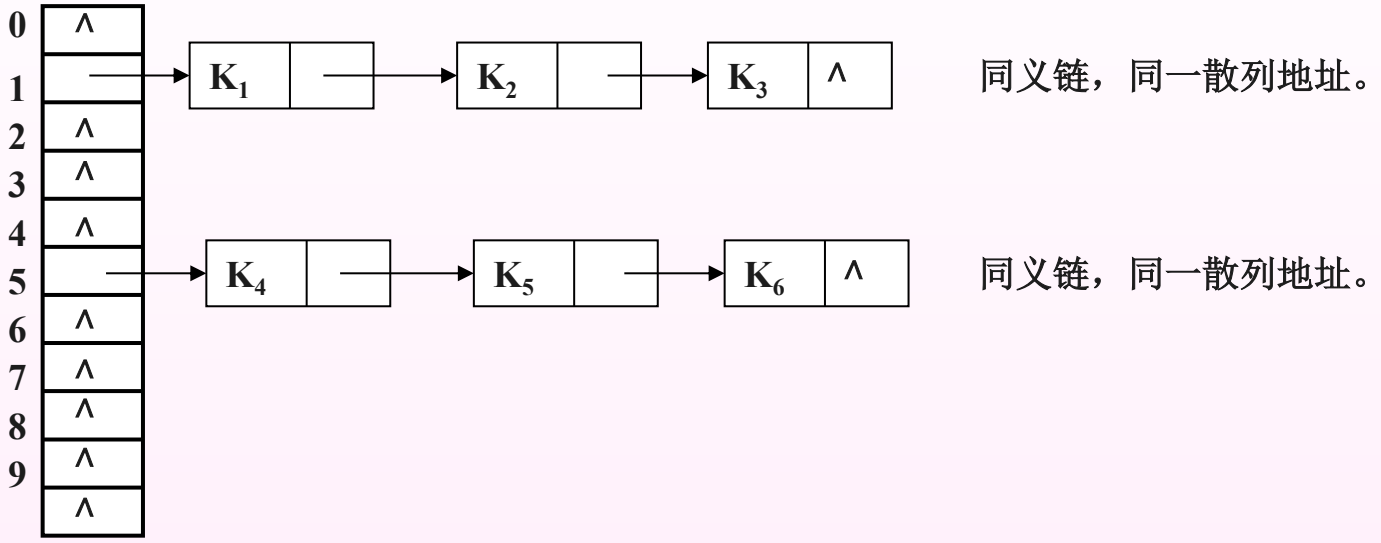
  di=1,2,...,m-1

  ② 随机再散列

  di=RN(i) RN 随机数序列

*处理冲突 —— 链地址法

- 当发生地址冲突后，将所有函数值相同的记录连成一个单链表。



3.2.2 哈希查找操作步骤

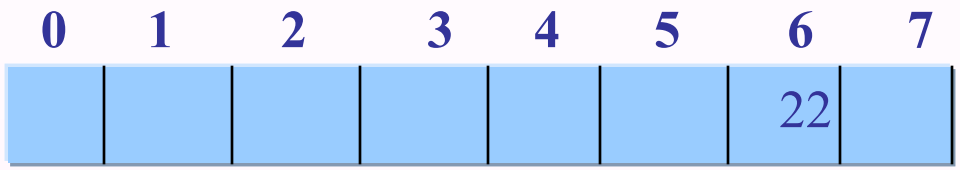
1. 用给定的哈希函数构造哈希表
2. 根据选择的冲突处理方法解决地址冲突
3. 在哈希表的基础上执行哈希查找

一、建立哈希表

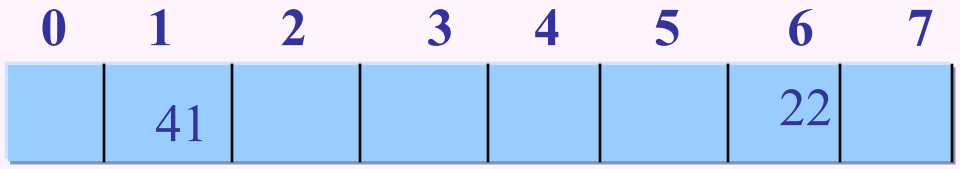
- 建立哈希表操作步骤：
  - step1　取数据元素的关键字key，计算其哈希函数值（地址）。若该地址对应的存储空间还没有被占用，则将该元素存入；否则执行step2解决冲突。
  - step2　根据选择的冲突处理方法，计算关键字key的下一个存储地址。若下一个存储地址仍被占用，则继续执行step2，直到找到能用的存储地址为止。

举例

- 对给定数列{ 22,41,53,46,30,13,1,67 },建立哈希表。表长取9,即[0～8]。哈希函数设定为: H(key) = key MOD 8 ,用线性探测解决冲突 Hi=(H(key)+di) MOD m ,di=1,2,...,m-1。
- 取22，计算H（22）=22 mod 8 = 6，该地址空，可用；



取41，计算H（41）=41 mod 8 = 1，该地址空，可用；

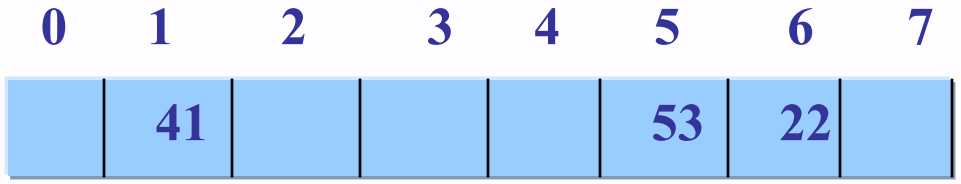


比较次数： 1 1

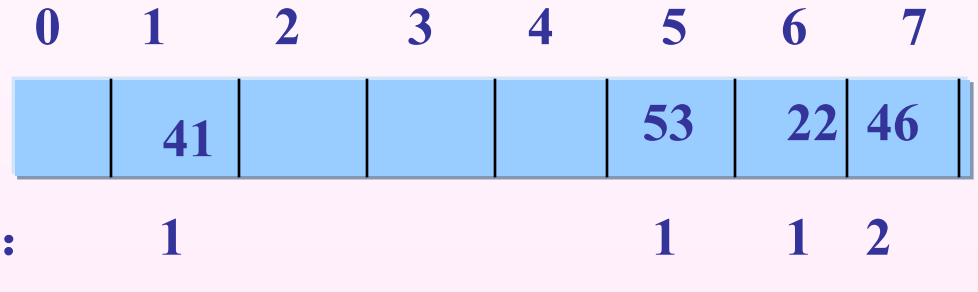
举例（续一）

{ 22,41,53,46,30,13,1,67 }

取53，计算 H（53）= 53 mod 8 = 5，该地址空，可用；



取46，计算 H（46）= 6，该地址冲突，用线性探测法计算下一个可用地址 Hi=（6+1）MOD 8 = 7，　该地址空，可用；



举例（续二）

{ 22,41,53,46,30,13,1,67 }

- 取30，计算 H（30）= 6，该地址冲突，用线性探测法计算下一个可用地址 Hi=（6+1）MOD 8 = 7，　该地址冲突，再用线性探测法计算下一个可用地址；Hi=0，地址空，可用；

0	1	2	3	4	5	6	7
30	41				53	22	46
比较次数：3	1				1	1	2

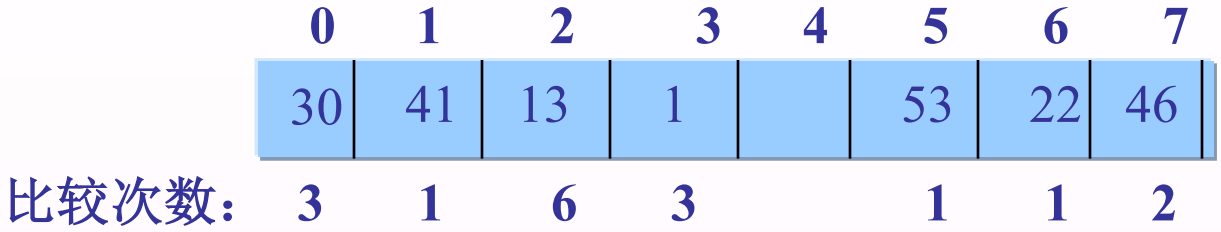
- 取13，计算 H（13）= 5，依法解决冲突，得出：

0	1	2	3	4	5	6	7
30	41	13			53	22	46
比较次数：3	1	6			1	1	2

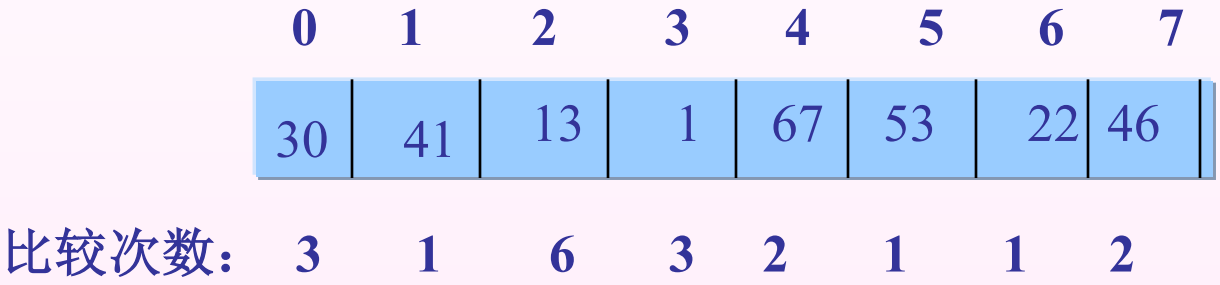
举例（续三）

{ 22,41,53,46,30,13,1,67 }

- 取1，计算 H（1）= 1，该地址冲突，解决冲突可得；



- 取67，计算 H（67）= 3，冲突，解决冲突，得出：



二、哈希查找

哈希查找的过程和建立哈希表的过程是一致的。

设哈希表为HST[0～M-1]，哈希函数取H（key），解决冲突的方法为R（x），则哈希查找步骤为：

- Step1 对给定k值，计算哈希地址 Di=H（k）；若HST[i]为空，则查找失败；若HST[i]=k，则查找成功；否则，执行step2（处理冲突）。
- Step2 重复计算处理冲突的下一个存储地址

  Dk=R（Dk-1），直到HST[Dk]为空，或HST[Dk}=k为止。

  若HST[Dk]=K，则查找成功，否则查找失败。

查找举例

以上述哈希表为例。哈希函数为H(key)=key MOD 8,

设有数列{22, 41, 53, 46, 30, 13, 1, 67}, 用线性探测法解决冲突, 建立的哈希表为:

0	1	2	3	4	5	6	7
30	41	13	1	67	53	22	46
比较次数： 3	1	6	3	2	1	1	2

平均查找长度ASL= $\frac{1}{8}(3+1+6+3+2+1+1+2)=\frac{19}{8}$

① 查找key = 67 比较两次找到, 查找成功;

② 查找key = 21 比较8次找不到, 查找失败。

平均查找次数

- 线性Hash表

  (2-α)/(2-2α)

- 随机Hash表

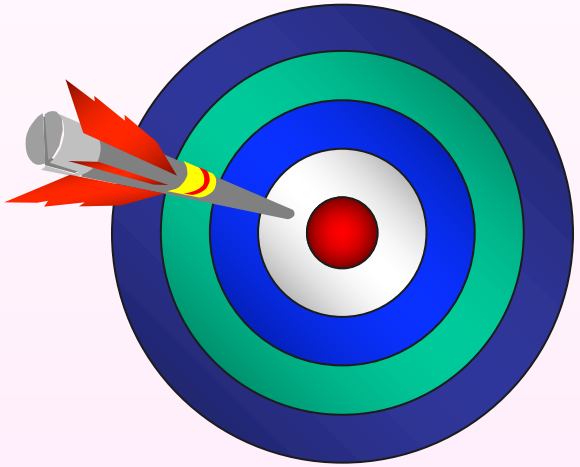
1/α ln(1-α)

α=m/n为Hash表的填满率，n为Hash表的长度，m为表中存在的关键字个数。

总结:

哈希查找　哈希函数常用的构造方法：数字分析法、平方取中法、折叠法、除留余数法（求模取余法）、直接定址法

处理冲突有两种方法：开放地址法（线性探测再散列、随机探测再散列 ）、链地址法



作业

- P232 3.4-(2)