第五章　哈希表与索引表

- 查找的基本概念
- 哈希表
  - 哈希表的基本概念
  - 冲突产生
  - 哈希函数的构造
  - 解决冲突的方法
  - 哈希表的性能分析与应用
- 索引表
  - 索引表的构成
  - 索引表的查找
  - 分块索引
- 各种查找算法的效率分析

2.1 查找的概念(Search by KEY)

- 计算机存储信息的目的是要使用这些信息，必须能对这些信息进行查找，从中找到所需的数据。
- 如何实现对信息的有效查找？针对信息的不同特点和不同的存储方法，有不同的查找方法。

- 查找表：同一类型数据组成的集合，每个数据在表中又称为一个记录。
- 查找是作用在同一个类型的数据元素集合上的，将这样的集合称为查找表。
- 关键字(Key)：是数据元素中某个数据项的值。
  - 主关键字：可以唯一地标识一个记录。
  - 例如：学生记录中的学号

学号	姓名	年龄	籍贯

查找的概念

所谓查找，就是在数据集合中按关键字寻找满足某种条件的数据对象。

1、**顺序查找**---- 查找与值x匹配的元素

待查找数据元素存放在顺序表中，在顺序表中查找一个元素的基本思想是逐个比较，直到找到为止，或者失败。

查找算法效率：O(n)

2、有序表的**折半查找**(**Binary Search也称二分法**)

折半查找表的要求：各个记录按关键字大小顺序排列(有序表)，查找表是可以随机检索的（如数组等）

折半查找的思想：

原理：通过与中间位置上的元素进行比较，每次可以将比较的元素范围缩小一半。

查找算法效率： **$O(\log_2 n)$**

2.2 哈希表查找(Hash Table)

1）哈希表的概念

- 前面的查找算法通过与查找表中的所有记录依次比较来完成。在依次比较之前，没有办法确定查找关键字的位置。key和存储地址之间没有关系。（无法实现类随机查找）
- 在字典中查找一个单词时，可以根据单词的首字母，直接翻到字典的某一部分。只是因为我们事先知道，字典里的单词时按照字母顺序排列的，所以某个字母开始的单词的存放位置可以大致知道。

  - 如果我们在存放记录时，按照一定的规律，将某个记录根据它的关键字的值存放到一定的位置上，那么我们在查找时，就可以根据给定的关键字，知道到哪里去找。

存储地址= f(key)

哈希表查找(Hash Table)

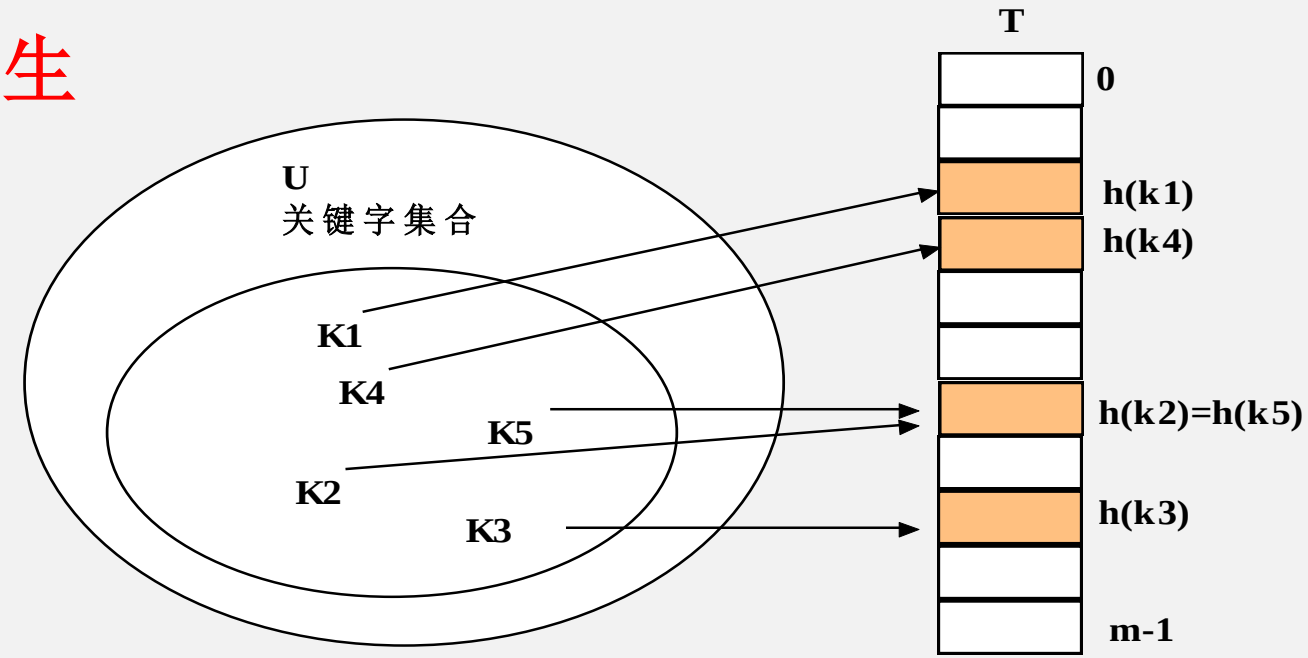
- 哈希方法是在记录的存储位置与其关键字之间建立一个确定的对应函数关系Hash( )，使每个关键字与存储结构中一个唯一存储位置相对应：

  $$Address = Hash(Record.key)$$

- 在存放记录时，依相同函数计算存储位置，并按此位置存放。
- 在搜索时，先对记录的关键字进行相同函数计算，把函数值当做记录的存储位置，在结构中按此位置取记录进行比较。若关键字相等，则搜索成功。
- 哈希表：利用哈希方法建立起来的查找表。
- 哈希函数：哈希方法中用于建立关键字和存储地址之间关系的函数。

哈希表查找(Hash Table)

2) 哈希冲突的产生



特点：

•使用哈希方法进行搜索不必进行多次关键字的比较，搜索速度比较快，可以直接到达或逼近具有此关键字的记录的实际存放地址。

•哈希函数是一个压缩映象函数。关键字集合比哈希表地址集合大得多。因此有可能经过哈希函数的计算，把不同的关键字映射到同一个哈希地址上，这就产生了冲突（collision）。

•我们称这些产生冲突的哈希地址相同的不同关键字为同义词.

哈希表查找(Hash Table)

哈希函数与哈希冲突：

由于关键字集合比地址集合大得多，冲突很难避免。所以对于哈希方法，需要讨论以下两个问题：

- **对于给定的一个关键字集合，选择一个计算简单且地址分布比较均匀的哈希函数，避免或尽量减少冲突；**

  注意：哈希表主表只能采用顺序存储，且表空间一般大于等于关键字集合。

- **拟订解决冲突的方案。**

哈希表查找(Hash Table)

3）构造哈希表

- 给定哈希函数
- 给定冲突处理方法。

- 构造哈希函数
  - 原则
    - 哈希函数应是简单的，能在较短的时间内计算出结果。
    - 哈希函数计算出来的地址应能均匀分布在整个地址空间中 ： 若 *key* 是从关键字集合中随机抽取的一个关键字， 哈希函数应能以同等概率取地址集合 中的每一个值。
    - 哈希函数的定义域必须包括需要存储的全部关键字， 如果哈希表允许有 *m* 个地址时， 其值域必须在 0 到 *m*-1 之间。
- Hash函数构造方法

直接定址法、数字分析法、平方取中法、折叠法、取模法

哈希表查找(Hash Table)

(1) 直接定址法

此类函数取关键字的某个线性函数值作为哈希地址。

Hash (key) ＝ a * key + b 其中： **a, *b*为常数**

这类哈希函数是一对一的映射，一般不会产生冲突。但是，它要求哈希地址空间的大小与关键字集合的大小相同。

示例：有一组关键字如下：{ 942148, 941269, 940527, 941630, 941805, 941558, 942047, 940001 }。哈希函数为

Hash (*key*) = *key* - 940000

Hash (942148) = 2148　　*Hash* (941269) = 1269

Hash (940527) = 527　　*Hash* (941630) = 1630

Hash (941805) = 1805　　*Hash* (941558) = 1558

Hash (942047) = 2047　　*Hash* (940001) = 1

可以按计算出的地址存放记录。

哈希表查找(Hash Table)

(2) 数字分析法

- 设有 n 个 d 位数，每一位可能有 r 种不同的符号。这 r 种不同的符号在各位上出现的频率不一定相同。可根据哈希表的大小，选取其中各种符号分布均匀的若干位作为哈希地址。

例如：

9	4	2	1	4	8
9	4	1	2	6	9
9	4	0	5	2	7
9	4	1	6	3	0
9	4	1	8	0	5
9	4	1	5	5	8
9	4	2	0	4	7
9	4	0	0	0	1
①	②	③	④	⑤	⑥

若哈希表地址范围有 3 位数字，取各关键字的 ④⑤⑥ 位做为记录的哈希地址

哈希表查找(Hash Table)

(3) 平方取中法

- 先计算构成关键字的标识符的内码的平方, 然后按照哈希表的大小取中间的若干位作为哈希地址。
- 在平方取中法中, 一般取哈希地址为 2 的某次幂。例如, 若哈希地址总数取为 $m = 2^r$，则对内码的平方数取中间的 r 位。

例如：如果 $r = 3$，所取得的哈希地址参看图的最右一列。

标识符	内码	内码的平方	哈希地址
A	01	01	001
*A*1	0134	20420	042
*A*9	0144	23420	342
B	02	4	004
DMAX	04150130	215264436171 00	443
*DMAX*1	0415013034	526447352215 1420	352
AMAX	01150130	1354236171 00	236
*AMAX*1	0115013034	345424652215 1420	652

平方取中法最接近于“随机化”。

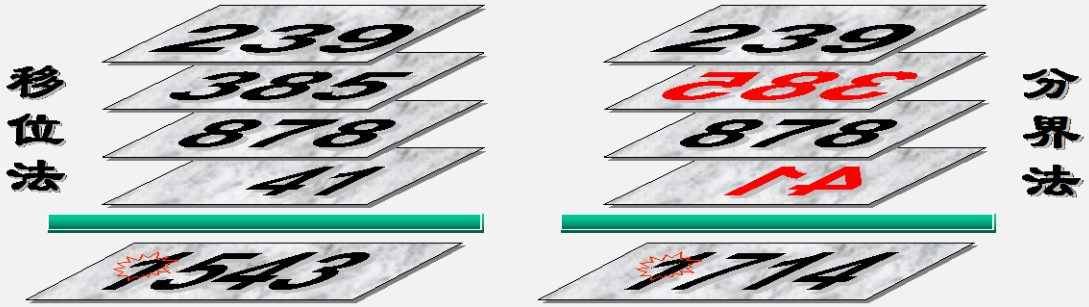
哈希表查找(Hash Table)

(4) 折叠法

- 此方法把关键字自左到右分成位数相等的几部分，每一部分的位数应与哈希表地址位数相同，只有最后一部分的位数可以短一些。
- 把这些部分的数据叠加起来，就可以得到具有该关键字的记录的哈希地址。
- 有两种叠加方法：
  - 移位法 — 把各部分的最后一位对齐相加；
  - 分界法 — 各部分不折断，沿各部分的分界来回折叠，然后对齐相加，将相加的结果当做哈希地址。
- 示例：设给定的关键字为 *key* = 2393858784 1，若存储空间限定 3 位，则划分结果为每段 3 位. 上述关键字可划分为 4段：

  239　385　878　41

- 把超出地址位数的最高位删去，仅保留最低的3位，做为可用的哈希地址。



注：一般当关键字的位数很多，而且关键字每一位上数字的分布大致比较均匀时，可用这种方法得到哈希地址

哈希表查找(Hash Table)

(5) 除留余数法（取模法）

- 设哈希表中允许地址数为 m, 取一个不大于 m, 但最接近于或等于 m 的质数 p 作为除数, 利用以下函数把关键字转换成哈希地址

$$hash(key) = key \% p \quad 其中：p \le m$$

- 其中, “%”是整数除法取余的运算，要求质数 p 不是接近2的幂。

- 示例: 有一个关键字 $key = 962148$, 哈希表大小 $m = 25$, 即 $HT[25]$。取质数 $p = 23$。哈希函数 $hash(key) = key \% p$。则哈希地址为

$$hash(962148) = 962148 \% 23 = 12。$$

- 可以按计算出的地址存放记录。需要注意的是, 使用上面的哈希函数计算出来的地址范围是 0到 22, 因此, 从23到24这几个哈希地址实际上在一开始是不可能用哈希函数计算出来的, 只可能在处理溢出时达到这些地址。

哈希表查找(Hash Table)

4) 冲突处理方法

- 因为任一种哈希函数也不能避免产生冲突，因此选择好的解决冲突的方法十分重要。

(1) 开放地址法（Open Addressing)

$H_i = (H(key) + di) \text{ MOD } m$　　其中：　i=1,2,...,k

- 线性探查法 (Linear Probing)
  - di=1,2,3,...,m-1
- 二次探测法 (Quadratic Probing)
  - di=
- 伪随机探查法 (Linear Probing)
  - di=伪随机数序列

哈希表查找(Hash Table)

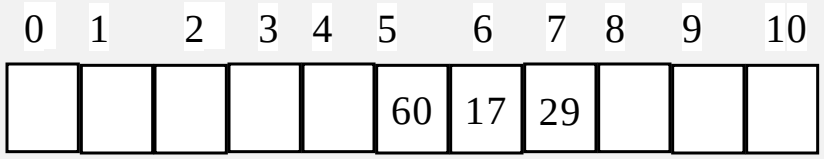
线性再探测方法举例：

假设哈希函数为：

H(key) = key%7

关键字序列: 15, 14, 28, 26, 56, 23

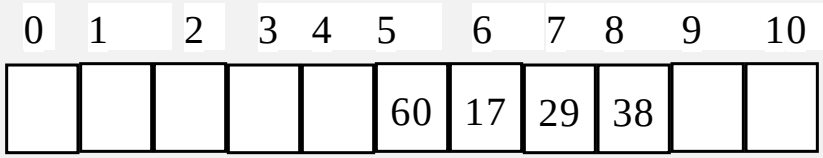
下标	键值
0	14
1	15
2	28
3	56
4	23
5	26
6	



H(key)=key MOD 11

待存储：38，19，16

1、线性探测再散列



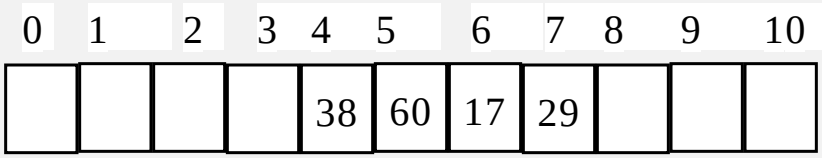
H(38)=5 冲突， di=1: H=5+1=6 冲突

di=2: H=5+2=7 冲突

di=3: H=5+3=8 解决

造成19存放时，非同义词之间冲突的产生。

2、二次探测再散列

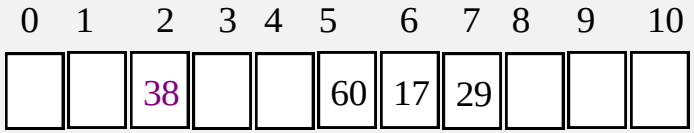


H(38)=5 冲突， di=1: H=5+1=6 冲突

di=-1: H=5-1=4 解决

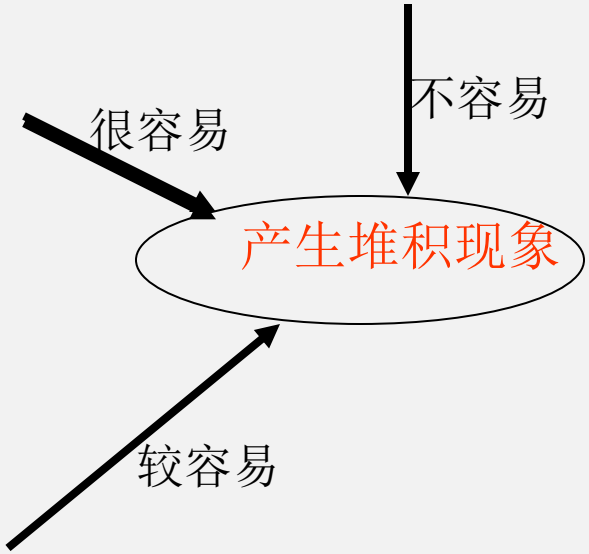
造成16存放时，非同义词之间冲突的产生。

3、随机数探测再散列



H(38)=5 冲突， di=-3: H=5-3=2 解决

在剩余空间中造成非同义词冲突的几率平均化。



哈希表查找(Hash Table)

(2) 再哈希法：

- 多次用不同的哈希函数运算。

(3) 公共溢出区

- 设立专门的溢出区，所有产生冲突的记录放在一起。
- 在公共溢出区中进行顺序查找。

(4) 链地址法

- 将所有哈希值相同的记录链在同一个线性列表里。(即数组+单链表)

哈希表查找(Hash Table)

Bucket hashing(公共溢出区法)

假设： H(key) = key%6

键值序列：8、12、6、4、16、23、25、19

Hash table

0	12
1	25
2	8
3	
4	4
5	23

Overflow

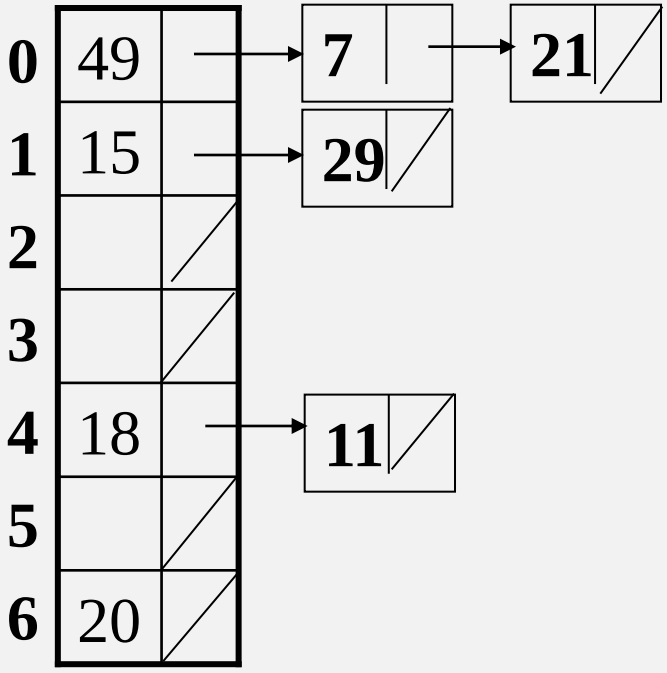
0	6
1	16
2	19
3	
4	
5	

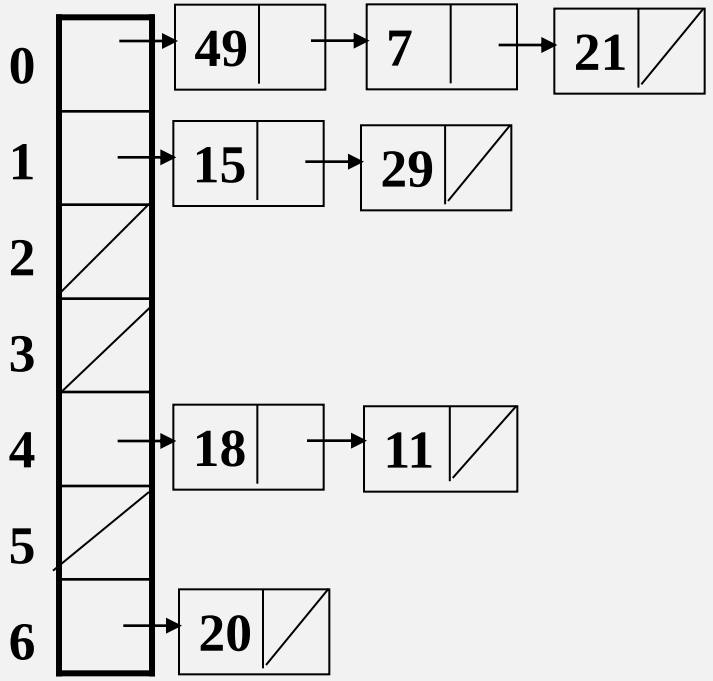
哈希表查找(Hash Table)

两种---链地址法：

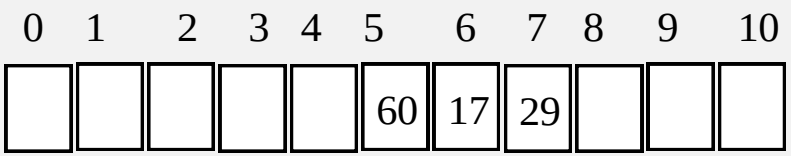
假设哈希函数为 H(key) = key%7

待插入的关键字序列为: 18、49、15、7、20、11、21





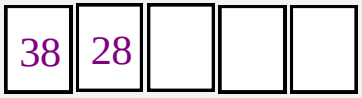
哈希表查找(Hash Table)

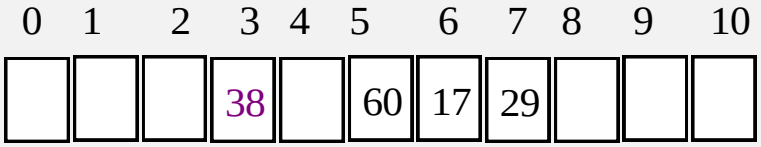


H(key)=key MOD 11

38, 19,16,28

公共溢出区：

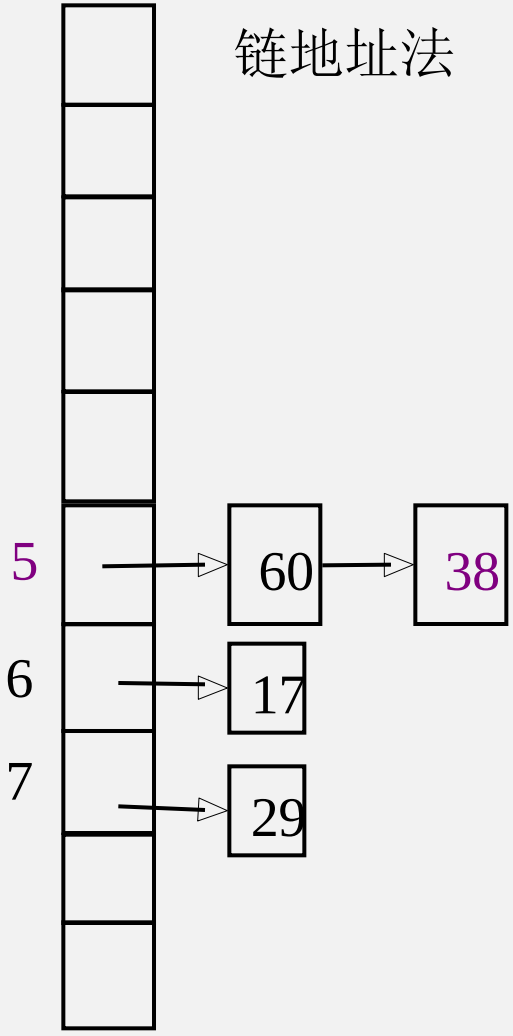




再哈希法： H(key)=key MOD 7

H(38)=3

链地址法



哈希表查找(Hash Table)

5) 哈希表查找效率分析

平均查找长度

$$ASL(n)= \frac{1}{n}\sum_{i=1}^{n} l_i$$

通常情况下，具有相同的冲突解决方法的哈希表，其平均查找长度和哈希表的装填因子有关。

装填因子为 $\alpha = n / (m)$

它反映了表的装满程度。

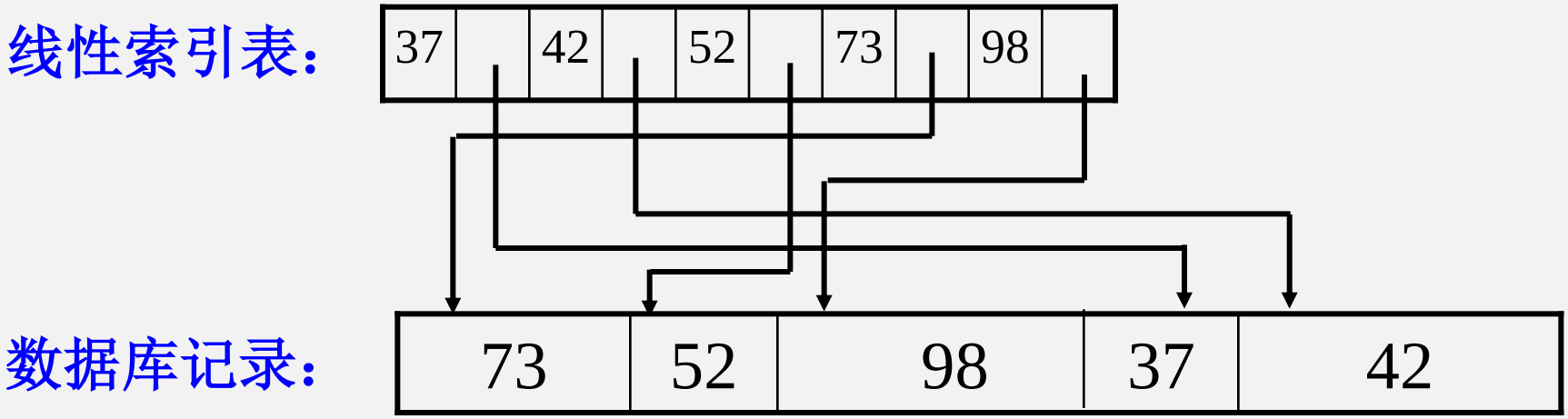
2.3 索引表查找(Index Table)

2.3、索引顺序表

1）线性索引表

线性索引表是指按关键字顺序组织的<关键字，指针>有序序列。其中：关键字是唯一标识记录的字段，指针指向与关键字对应的记录位置。

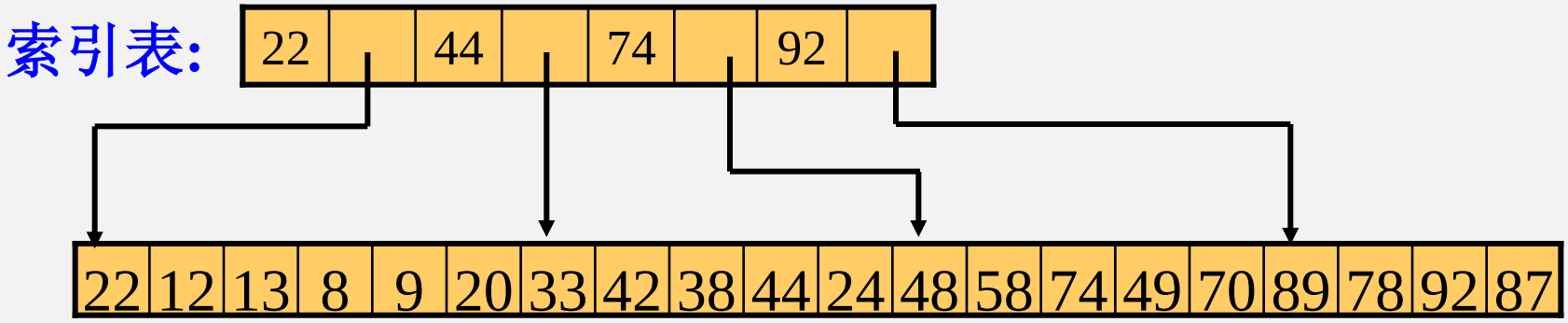
例如：



索引表查找(Index Table)

2)、索引块查找

- 除表本身外，还建立了一个索引。表中数据分块，每一块是一个子表，子表内的数据是无序的，但子表之间的数据是有序的。（即块有序）
- 利用索引，先将数据定位到子表，而后再在子表中，进行查询。减小数据集合。



索引表查找(Index Table)

- **二步查询**：
  - 确定数据所在的分块
  - 在相应的子表中查询
- **算法**
  - 索引表上的查找算法+顺序表的查找算法
- **时间复杂度**：$ASL_{bs} = L_b + L_w$

其中：L_b为确定所在块的平均查找长度。

L_w为在块中查找时的平均查找长度。

a. 若采用顺序查找确定所在块，则：

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

b. 若采用折半查找确定所在块，则：

$$ASL = \log_2(n/s+1) + s/2$$