第9章 查找

9.0 基本概念

9.1 静态查找表

9.2 动态查找表

9.3 哈希表

9.3 B-树和B+树


B-树
B-树
每个结点至多有m棵子树，m-1个关键字
若根结点不是终端结点，则至少有两棵子树
除根之外的所有非终端结点至少有⌈m/2⌉棵子树，至少有⌈m/2⌉-1 个关键字
B-树查找
思想：类似二叉排序树，先找结点，再找关键字
B-树插入
在终端节点插入
关键字个数少于m-1
关键字个数等于m-1
分裂+上溢
B-树删除
Ki在非终端结点中
用Ai指向子孙节点中最小关键字替代
Ki在最下层结点中
关键字个数n≥ ⌈m/2⌉ 直接删除
关键字个数n= ⌈m/2⌉ - 1，兄弟关键字个数n≥ ⌈m/2⌉，借一个
自己及左右兄弟关键字个数n=⌈m/2⌉ - 1，结点合并
B+树
每个结点最多m棵子树，每个结点中最多含有m个关键字
所有的叶子结点中包含了全部关键字
结点仅含其子树中的最大（或最小）关键字
两个指针：一个指向根结点，一个指向关键字最小的终端结点
B+树操作
插入：只在终端结点进行，关键字大于m时分裂
删除
不是终端节点最大关键字，直接删除
结点中最大关键字，用剩下最大关键字替代双亲结点对应关键字
若因删除而使结点中关键字的个数少于⌈m/2⌉时，结点合并

9.4 哈希查找表

一、哈希表的概念

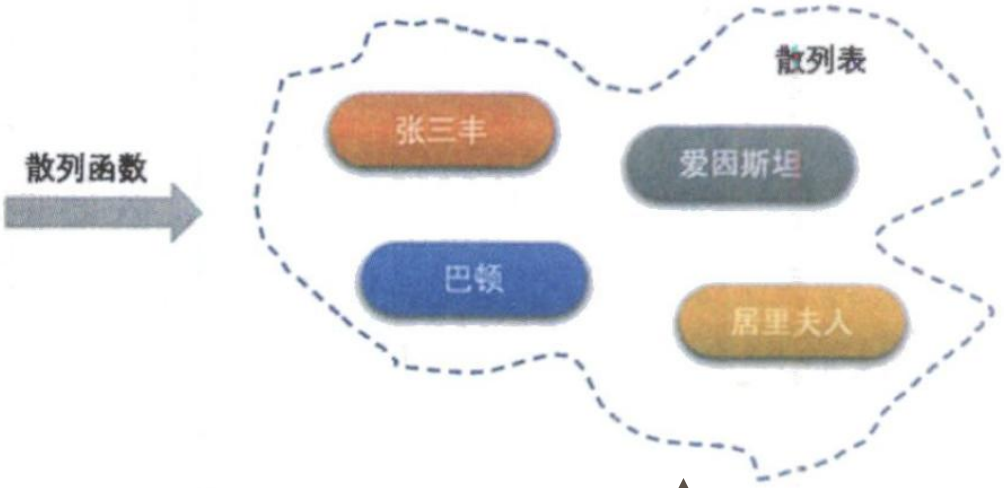
二、哈希函数的构造方法

三、冲突处理方法

四、哈希表的查找及分析

9.4.1 哈希表的概念

编号	姓名	地址
1	张三丰	体育馆
2	爱因斯坦	图书馆
3	居里夫人	化学实验室
4	巴顿	网吧



$$H = f(key)$$

散列函数　　哈希表

优点： 查找速度极快（O(1)）,查找效率与元素个数n无关！

9.4.1 哈希表的概念

例1：若将学生信息按如下方式存入计算机，如：

将2001011810201的所有信息存入V[01]单元；

将2001011810202的所有信息存入V[02]单元；

……

将2001011810231的所有信息存入V[31]单元。

欲查找学号为2001011810216的信息，便可直接访问V[16]！

9.4.1 哈希表的概念

例2：

有数据元素序列(14，23，39，9，25，11)，若规定每个元素k的存储地址H(k)=k，请画出存储结构图。

H(k)称为散列函数

解：根据散列函数H(k)=k，可知元素14应当存入地址为14的单元，元素23应当存入地址为23的单元，……，

对应散列存储表（哈希表）如下：

地址	…	9	…	11	…	14	…	23	24	25	…	39	…
内容		9		11		14		23		25		39	

9.4.1 哈希表的概念

讨论：如何进行散列查找?

根据存储时用到的散列函数H(k)表达式，迅即可查到结果!

例如，查找key=9,则访问H(9)=9号地址，若内容为9则成功;

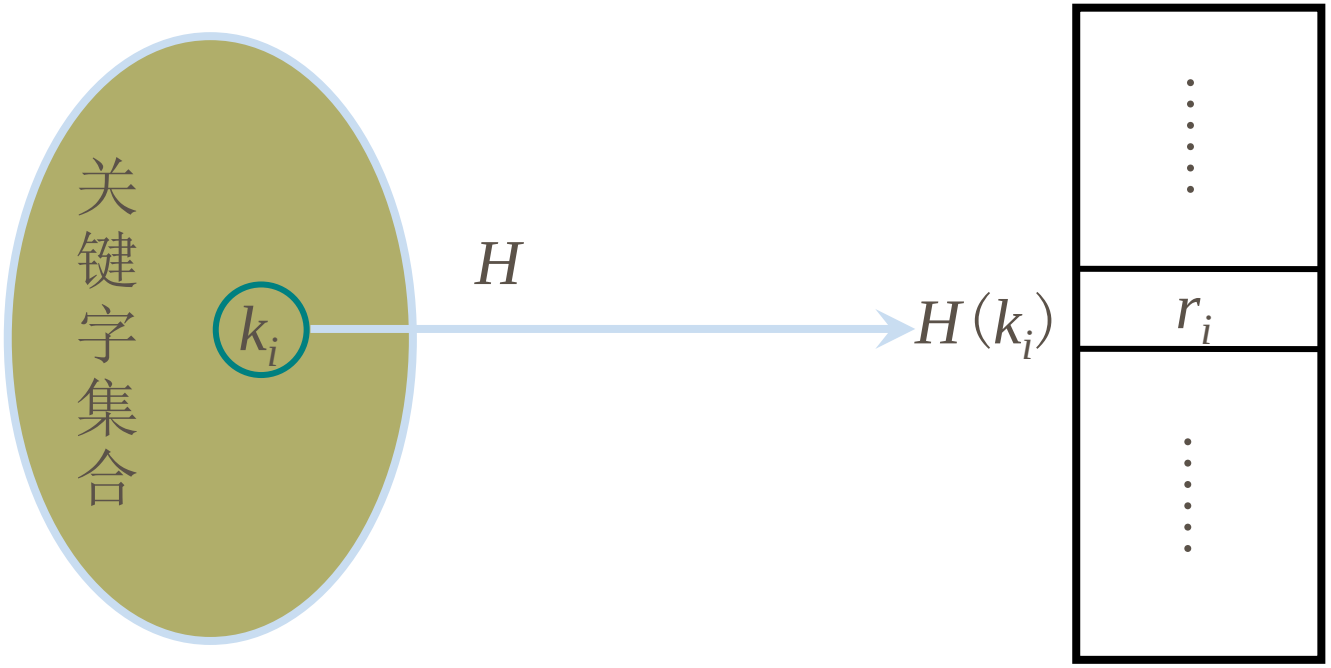
若查不到，应当设法返回一个特殊值，例如空指针或空记录。

明显缺点：空间效率低

如何解决?

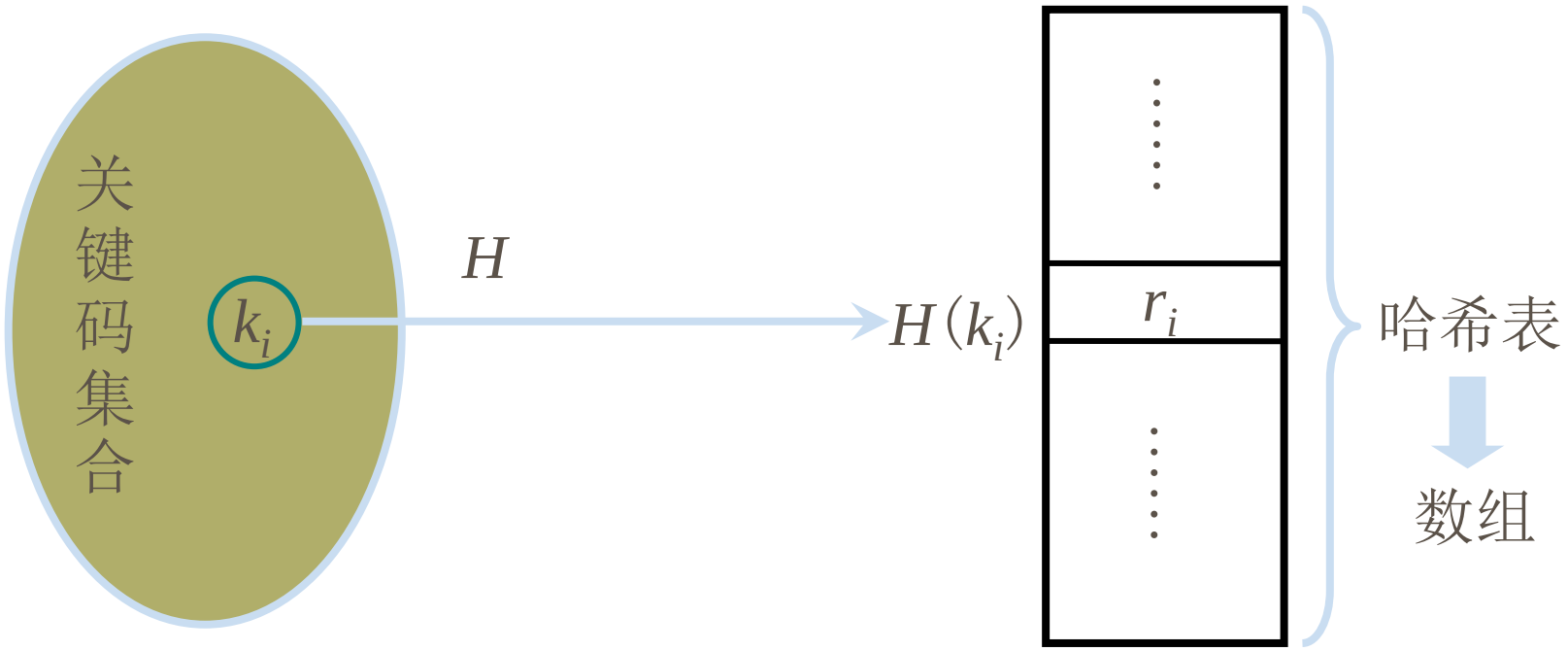
9.4.1 哈希表的概念

哈希的基本思想：在记录的存储地址和记录的关键字之间建立一个确定的对应关系。这样，不经过比较，一次读取就能得到所查的记录。



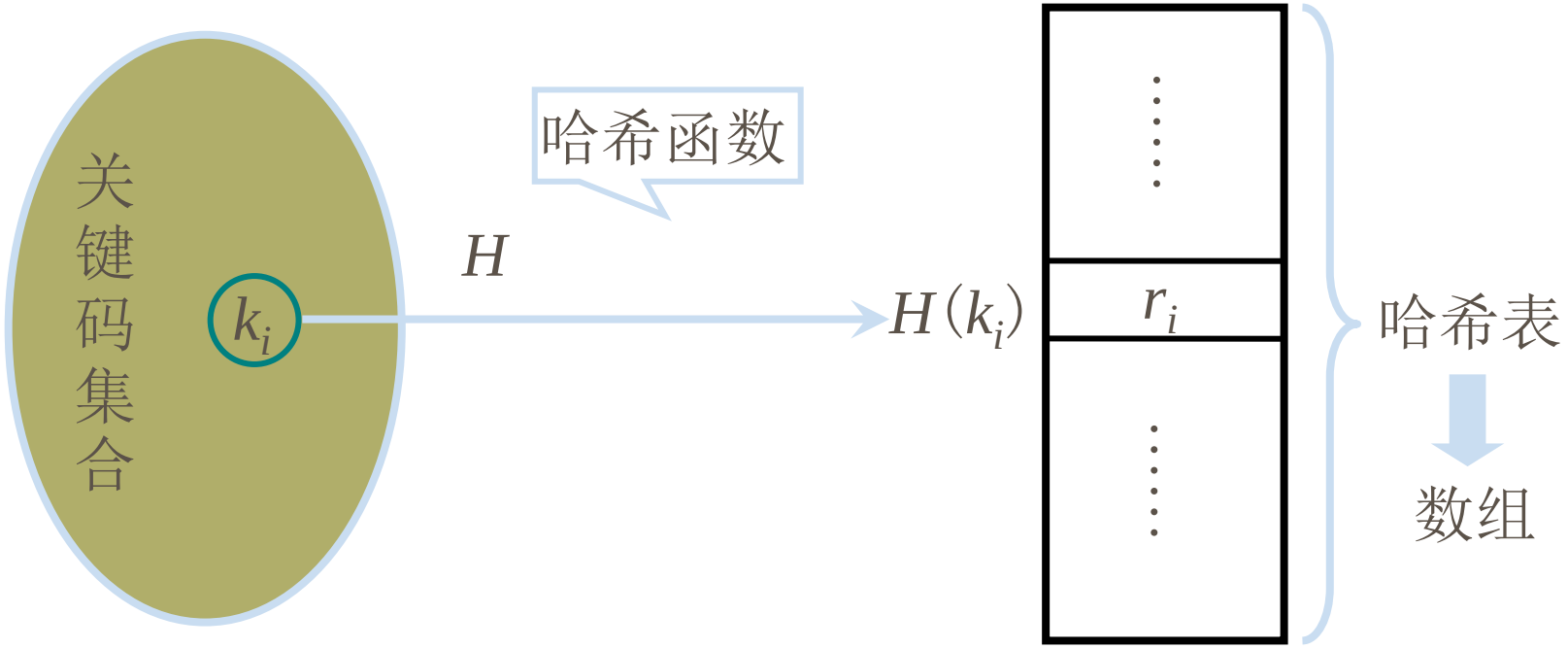
9.4.1 哈希表的概念

哈希表： 采用哈希技术将记录存储在一块连续的存储空间中，这块连续的存储空间称为哈希表。哈希表的容量m≥记录个数n。



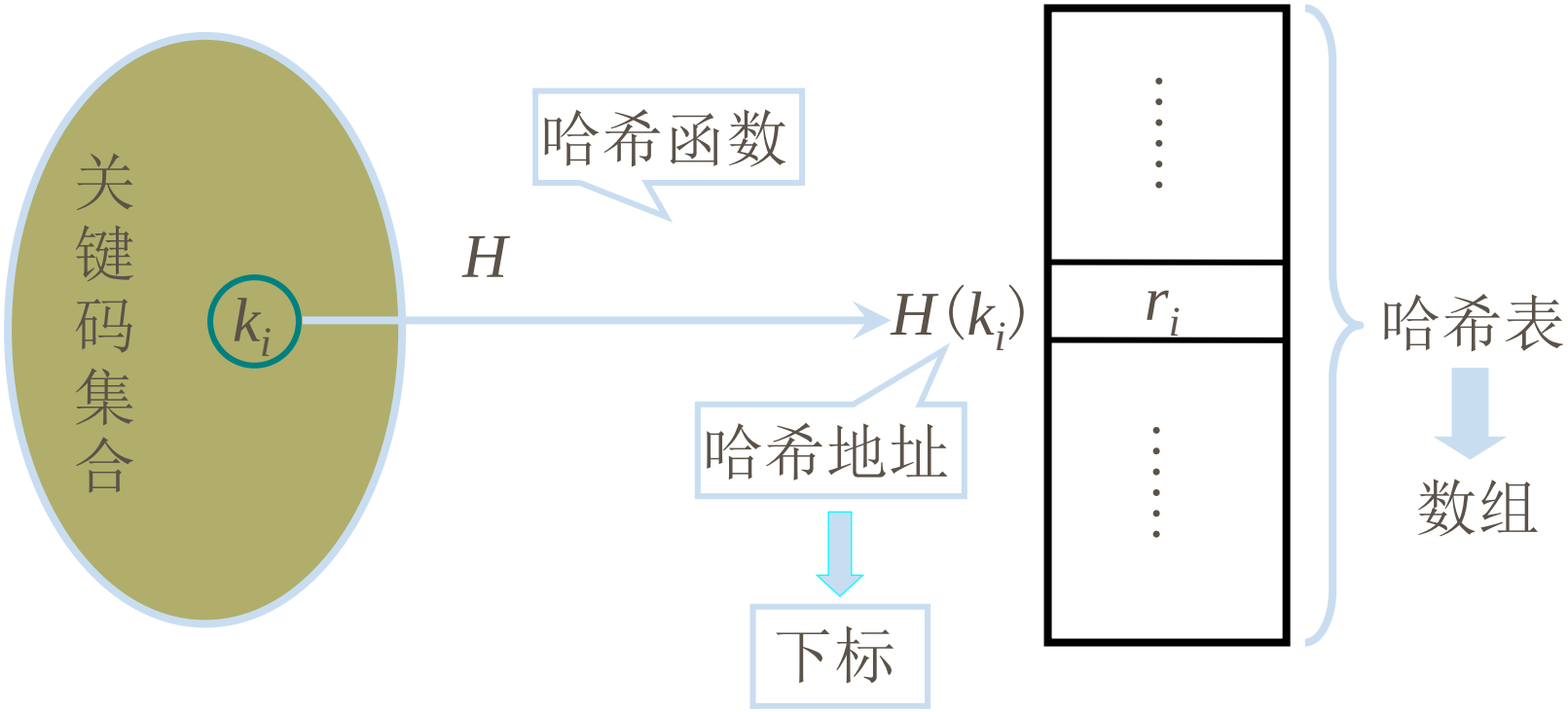
9.4.1 哈希表的概念

哈希函数：将关键字映射为哈希表中适当存储位置的函数，也称为散列函数。



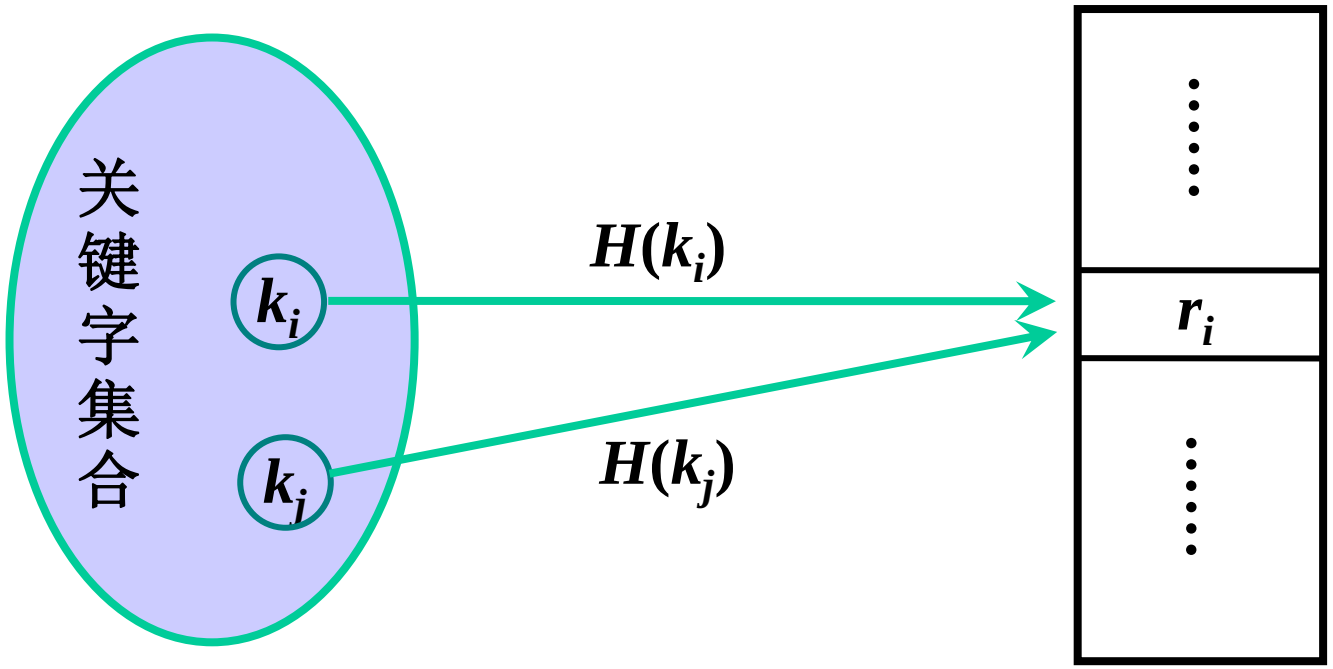
9.4.1 哈希表的概念

哈希地址： 由哈希函数所得的存储位置值 。



9.4.1　哈希表的概念

冲突：通常关键码的集合比哈希地址集合大得多，因而经过哈希函数变换后，可能将不同的关键码映射到同一个哈希地址上，这种现象称为冲突。

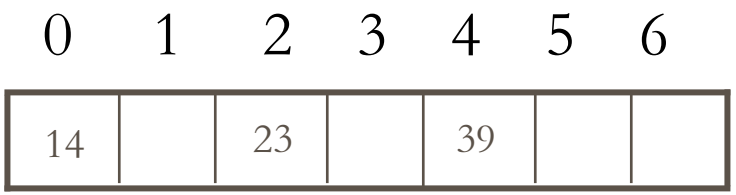


9.4.1 哈希表的概念

冲突现象举例：

有6个元素的关键码分别为（14，23，39，9，25，11）
选取关键码与元素位置间的函数为H(k)=k mod 7

通过哈希函数对6个元素建立哈希表：



6个元素用7个地址应该足够!

H(14)=14%7=0

H(25)=25%7=4
H(11)=11%7=4

有冲突!

9.4.1 哈希表的概念

在哈希查找方法中，冲突是不可能避免的，只能尽可能减少。

所以，哈希方法必须解决以下两个问题：

1）构造好的哈希函数

（a）所选函数尽可能简单，以便提高转换速度；

（b）所选函数对关键码计算出的地址，应在哈希地址内集中并大致均匀分布，以减少空间浪费。

9.4.1 哈希表的概念

2）制定一个好的解决冲突的方案

查找时，如果从哈希函数计算出的地址中查不到关键码，则应当依据解决冲突的规则，有规律地查询其它相关单元。

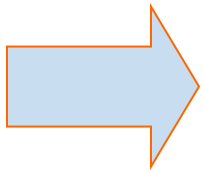
9.4.2 哈希函数的构造方法

要求一：n个数据原仅占用n个地址，虽然散列查找是以空间换时间，但仍希望散列的地址空间尽量小。

要求二：无论用什么方法存储，目的都是尽量均匀地存放元素，以避免冲突。

9.4.2 哈希函数的构造方法

常用的哈希函数构造方法有：



1. 直接定址法
2. 数字分析法
3. 平方取中法
4. 折叠法
5. 除留余数法
6. 乘余取整法
7. 随机数法

9.4.2 哈希函数的构造方法

1、直接定址法

Hash(key) = a·key + b　(a、b为常数)

优点：以关键码key的某个线性函数值为哈希地址，不会产生冲突.

缺点：要占用连续地址空间，空间效率低。


适用情况？

事先知道关键字，关键字集合不是很大且连续性较好。

9.4.2 哈希函数的构造方法

1、直接定址法

例：关键码集合为{100，300，500，700，800，900}，选取哈希函数为Hash(key)=key/100，

则存储结构（哈希表）如下：

0	1	2	3	4	5	6	7	8	9
	100		300		500		700	800	900

9.4.2 哈希函数的构造方法

2、数字分析法

- 特点：选用关键字的某几位组合成哈希地址。选用原则应当是：各种符号在该位上出现的频率大致相同。
- 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。
- 例如：取18位身份证号码的其中5位做关键字，可定义如下哈希函数：*H(key)=(key[6]-'0') *10^4 + (key[10]-'0')*10^3+ (key[14]-'0')*10^2 + (key[16]-'0')*10+key[17]-'0'*

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
3	**3**	**0**	**1**	**0**	**6**	**1**	**9**	**9**	**0**	**1**	**0**	**0**	**8**	**0**	**4**	**1**	**9**
省		市		区（县）下属辖区编号		（出生）年份				月份		日期		该辖区中的序号			校验

9.4.2 哈希函数的构造方法

2、数字分析法

例：有一组（例如80个）关键码，其样式如下：

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

讨论：

① 第1、2位均是“3和4”，第3位也只有“7、8、9”，因此，这几位不能用，余下四位分布较均匀，可作为哈希地址选用。

② 若哈希地址取两位（因元素仅80个），则可取这四位中的任意两位组合成哈希地址，也可以取其中两位与其它两位叠加求和后，取低两位作哈希地址。

9.4.2 哈希函数的构造方法

3、平方取中法

特点：对关键码平方后，按哈希表大小，取中间的若干位作为哈希地址。

理由：因为中间几位与数据的每一位都相关。

例：2589的平方值为6702921，可以取中间的029为地址。

9.4.2 哈希函数的构造方法

4、折叠法

特点：将关键码自左到右分成位数相等的几部分（最后一部分位数可以短些），然后将这几部分叠加求和，并按哈希表表长，取后几位作为地址。

适用于：每一位上各符号出现概率大致相同的情况。

法1：移位法 —— 将各部分的最后一位对齐相加。

法2：间界叠加法——从一端向另一端沿分割界来回折叠后，最后一位对齐相加。

例：元素42751896, 用法1： 427＋518＋96=1041

用法2： 427 518 96—> 724+518+69 =1311

9.4.2 哈希函数的构造方法

5、除留余数法

$Hash(key)=key \ mod \ p$ （p是一个整数）

特点：以关键码除以p的余数作为哈希地址。

关键：如何选取合适的p?

适用情况？

除留余数法是一种最简单、也是最常用的构造哈希函数的方法，并且不要求事先知道关键字的分布。

9.4.2 哈希函数的构造方法

5、除留余数法

例： $p=21=3\times7$

关键字	14	21	28	35	42	49	56
哈希地址	14	0	7	14	0	7	14

一般情况下，选p为小于或等于表长（最好接近表长）的最小素数或不包含小于20质因子的合数。

9.4.2 哈希函数的构造方法

6、乘余取整法

（A*key mod 1）就是取A*key 的小数部分

$Hash(key)= \lfloor B*(A*key \mod 1) \rfloor$

(A、B均为常数，且0<A<1，B为整数)

特点：以关键码key乘以A，取其小数部分，然后再放大B倍并取整，作为哈希地址。

例：欲以学号最后两位作为地址，则哈希函数应为：$H(k)=100*(0.01*k \% 1)$

其实也可以用法5实现：$H(k)=k \% 100$

9.4.2 哈希函数的构造方法

7、随机数法

Hash(key) = random (key) (random为伪随机函数)

适用于：关键字长度不等的情况。造表和查找都很方便。

9.4.2 哈希函数的构造方法

小结：构造哈希函数的原则：

① 执行速度（即计算哈希函数所需时间）；

② 关键字的长度；

③ 哈希表的大小；

④ 关键字的分布情况；

⑤ 查找频率。

9.4.3、冲突处理方法

常见的冲突处理方法有

1. 开放定址法（开地址法）
2. 链地址法（拉链法）
3. 再哈希法（双哈希函数法）
4. 建立一个公共溢出区

9.4.3、冲突处理方法

1、开放定址法（开地址法）

设计思路：有冲突时就去寻找下一个空的哈希地址，只要哈希表足够大，空的哈希地址总能找到，并将数据元素存入。

如何寻找下一个空的哈希地址?

（1）线性探测法（要求掌握）

（2）二次探测法（要求掌握）

（3）随机探测法

用开放定址法处理冲突得到的哈希表叫闭散列表。

9.4.3、冲突处理方法

（1） 线性探测法

$H_i=(Hash(key)+d_i)\ mod\ m \qquad (1 \leqslant i < m)$

其中：

Hash(key)为哈希函数

m为哈希表长度

d_i 为增量序列1，2，…m-1，且$d_i=i$

含义：一旦冲突，就找附近（下一个）空地址存入。

例： 关键码集为 {47，7，29，11，16，92，22，8，3}，
设：哈希表表长为m=11；
哈希函数为Hash(key)=key mod 11；
拟用线性探测法处理冲突。建哈希表如下：

0	1	2	3	4	5	6	7	8	9	10
11	22		47	92	16	3	7	29	8	

① 47、7是由哈希函数得到的没有冲突的哈希地址；

② Hash(29)=7，哈希地址有冲突，需寻找下一个空的哈希地址：由H_1=(Hash(29)+1) mod 11=8，哈希地址8为空，因此将29存入。

③ 22、8、3同样在哈希地址上有冲突，也是由H_1找到空的哈希地址的。

其中3 还连续移动了两次（二次聚集）

9.4.3、冲突处理方法

（1）线性探测法

优点： 只要哈希表未被填满，保证能找到一个空地址单元存放有冲突的元素；

缺点： 可能使第i个哈希地址的同义词存入第i+1个哈希地址，这样本应存入第i+1个哈希地址的元素变成了第i+2个哈希地址的同义词，…，

因此，可能出现很多元素在相邻的哈希地址上“堆积”起来，大大降低了查找效率。

解决方案：可采用二次探测法或伪随机探测法，以改善“堆积”问题。

9.4.3、冲突处理方法

1、开放定址法（开地址法）

（2）二次探测法

$H_i=(Hash(key)\pm d_i) \bmod m$

其中：Hash(key)为哈希函数

m为哈希表长度;

d_i为增量序列 1^2，-1^2，2^2，-2^2，…，q^2

9.4.3、冲突处理方法

(2) 二次探测法

仍举上例为{47，7，29，11，16，92，22，8，3}，改用二次探测法处理冲突，建表如下：

0	1	2	3	4	5	6	7	8	9	10
11	22	3	47	92	16		7	29	8	
	△	▲						△	△	

只有3这个关键码的冲突处理与上例不同，

Hash(3)=3，哈希地址上冲突，由

$H_1=(Hash(3)+1^2)\ mod\ 11=4$，仍然冲突；

$H_2=(Hash(3)-1^2)\ mod\ 11=2$，找到空的哈希地址，存入。

9.4.3、冲突处理方法

(2) 二次探测法

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。

9.4.3、冲突处理方法

(3) 伪随机探测法

增量序列使用一个伪随机函数来产生一个落在闭区间[1，m-1]的随机序列。

例：表长为11的哈希表中已填有关键字为17，60，29的记录，散列函数为H(key)=key MOD 11。现有第4个记录，其关键字为38，按三种处理冲突的方法，将它填入表中。

(1) H(38)=38 MOD 11=5　冲突

H1=(5+1) MOD 11=6　冲突

H2=(5+2) MOD 11=7　冲突

H3=(5+3) MOD 11=8　不冲突

9.4.3、冲突处理方法

(3) 伪随机探测法

增量序列使用一个伪随机函数来产生一个落在闭区间[1，m-1]的随机序列。

(2) H(38)=38 MOD 11=5　　冲突

H1=(5+1²) MOD 11=6　冲突

H2=(5-1²) MOD 11=4　不冲突

(3) H(38)=38 MOD 11=5　冲突

设伪随机数序列为9，则H1=(5+9) MOD 11=3 不冲突

0	1	2	3	4	5	6	7	8	9	10
			38	38	60	17	29	38		

9.4.3、冲突处理方法

2、再哈希法（双哈希函数法）

$H_i=RH_i(key)$　　$i=1, 2, \ldots, k$

RH_i均是不同的哈希函数，当产生冲突时就计算另一个哈希函数，直到冲突不再发生。

优点：不易产生聚集；

缺点：增加了计算时间。

9.4.3、冲突处理方法

3、链地址法(拉链法)

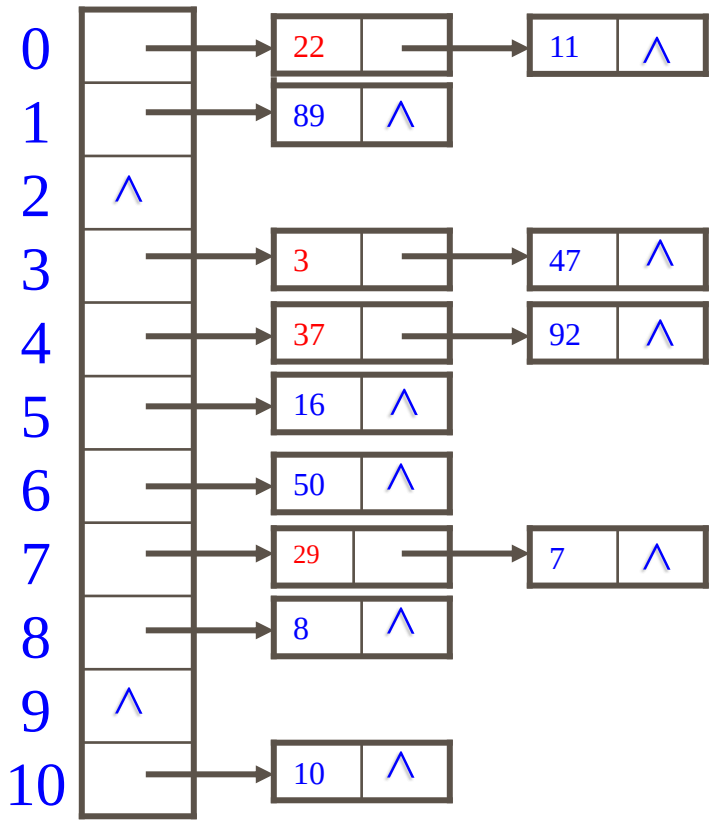
基本思想：将具有相同哈希地址的记录链成一个单链表，m个哈希地址就设m个单链表，然后用一个数组将m个单链表的表头指针存储起来，形成一个动态的结构。

用拉链法处理冲突构造的哈希表叫做开散列表。

9.4.3、冲突处理方法

3、链地址法(拉链法)

例：设{ 47, 7, 29, 11, 16, 92, 22, 8, 3, 50, 37, 89 ,10}的哈希函数为： Hash(key)=key mod 11，用拉链法处理冲突，则建表如右图所示。



有冲突的元素可以插在表尾,也可以插在表头。

9.4.3、冲突处理方法

4. 建立一个公共溢出区

基本思想：哈希表包含基本表和溢出表两部分（通常溢出表和基本表的大小相同），将发生冲突的记录存储在溢出表中。查找时，对给定值通过哈希函数计算哈希地址，先与基本表的相应单元进行比较，若相等，则查找成功；否则，再到溢出表中进行顺序查找。

9.4.3、冲突处理方法

例：关键码集合 {47, 7, 29, 11, 16, 92, 22, 8, 3}，哈希函数为$H(key)=key \bmod 11$，用公共溢出区法处理冲突，构造的哈希表为：

基本表

0	11
1	
2	
3	47
4	92
5	16
6	
7	7
8	8
9	
10	

溢出表

0	29
1	22
2	3
3	
4	
5	
6	
7	
8	
9	
10	

9.4.3、冲突处理方法

- 线性探测只要哈希表有空间，数据一定能够插进去。

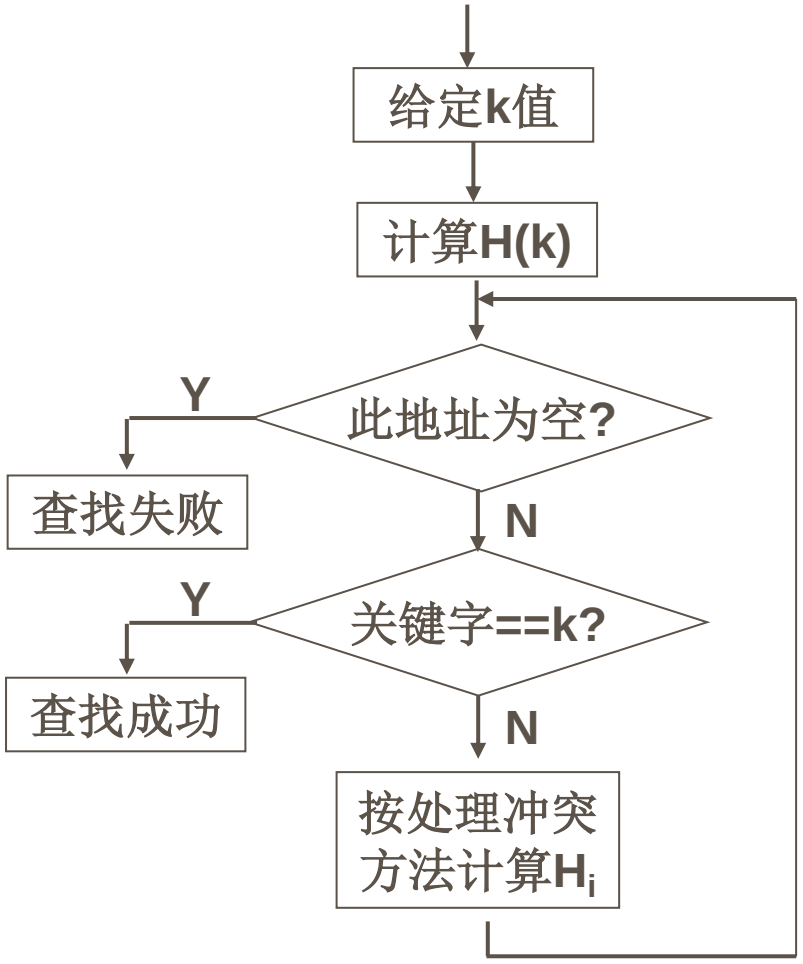
- 二次探测，如果表长m=4×k+3（k=1,2...）不成立，哈希表尽管有空间，数据有可能插不进去。

- 拉链法在任何情况下都能将数据插进去。

9.4.4 哈希表的查找及分析

- 基本思想

给定K值，根据造表时设定的哈希函数求得哈希地址，若表中此位置上没有记录，则查找不成功；否则比较关键字，若和给定值相等，则查找成功；否则根据造表时设定的处理冲突的方法找“下一地址”，直至哈希表中某个位置为“空”或者表中所填记录的关键字等于给定值时为止。



散列表的查找过程

9.4.4 哈希表的查找及分析

开放定址哈希表的存储结构

```
int hashsize[N]={997};
typedef struct{
        char *key;
}ElemType;
typedef struct{
        ElemType *elem; //动态分配数组
        int count; //当前数据元素个数
        int sizeindex;  //hashsize[sizeindex]为当前容量
}HashTable;
```

9.4.4 哈希表的查找及分析

开放地址哈希查找

```
int searchHash(HashTable h,KeyType key,int &p,int &c){// 查找
    // c用以计冲突次数，其初值置零,供建表插人时参考
    // p返回的是插入位置
    p = Hash(key); /// 求得哈希地址
    int q = p; //q保存hash值
    while (h.elem[p].key!=NULLKEY &&     // 该位置中填有记录
        !EQ(key,h.elem[p].key)){         /并且关键字不相等
        p = q;
        collision(p,++c);                //求得下一次探查地址p
       // 冲突次数未达到上限，继续；否则退出循环
    }
    if EQ(K，H.elem[p].key) return 1;
       //查找成功，p返回待查数据元素位置
    else return 0; // 查找不成功(H.elem[p].key == NULLKEY)
}
```

9.4.4 哈希表的查找及分析

```
int insertHash(HashTable &h,ElemType e){
// 查找不成功时插人数据元素 e到开放定址哈希表H中,并返回 K;若冲突次数
// 过大,则重建哈希表
    c=0;
    if (searchHash(h,e.key,p,c))
        return -1; //已存在
    else if (c<hashsize[h.sizeindex]/2){ //达到冲突上限
        h.elem[p]=e; //插入e
        ++h.count; return 1;
    }else{
        RecreateHashTable(H); //重建哈希表
        return 0;
    }
}
```

9.4.4 哈希表的查找及分析

- 哈希表的查找过程仍然是给定值和关键字进行比较的过程，仍然用平均查找长度衡量哈希表的查找效率
- 查找过程需和关键字比较的次数取决于三个因素：哈希函数、处理冲突的方法、哈希表的装填因子。
  - ① α=表中填入的记录数/哈希表的长度
  - ② 处理冲突方法相同的哈希表，平均查找长度依赖于哈希表的装填因子
  - ③ 一般来说，α越小，发生冲突的可能性越小。

9.4.4 哈希表的查找及分析

哈希查找的特点：

① 优点：插入查找的速度快；

② 缺点：

a. 通过哈希函数计算哈希地址时，占用一定的计算时间。

b. 占用的存储空间多。为减少冲突的发生，哈希表的长度应大于记录的长度。

c. 在哈希表中只能按关键字进行查找。

9.4.4 哈希表的查找及分析

明确：散列函数没有“万能”通式，要根据元素集合的特性而分别构造。

1：哈希查找的速度是否为真正的O（1）？

答：不是。由于冲突的产生，使得哈希表的查找过程仍然要进行比较，仍然要以平均查找长度ASL来衡量。

9.4.4 哈希表的查找及分析

2：“冲突”是不是特别讨厌?

答：不一定！正因为有冲突，使得文件加密后无法破译！（单向散列函数不可逆，常用于数字签名和间接加密）。

哈希表特点：源文件稍稍改动，会导致哈希表变动很大。

典型应用：md5散列算法

9.4.4 哈希表的查找及分析

3 散列存储的查找效率到底是多少?

答：ASL与装填因子α有关！既不是严格的O(1)，也不是O(n)

$$ASL \approx 1 + \frac{\alpha}{2}$$ (拉链法）

$$ASL \approx \frac{1}{2}(1 + \frac{1}{1-\alpha})$$ (线性探测法)

$$ASL \approx -\frac{1}{\alpha}\ln(1-\alpha)$$ (随机探测法）

应尽量选择一个合适的α，以降低ASL的长度

9.4.5 哈希应用

1、数字签名（数字手印）

HASH函数，是在信息安全领域有广泛和重要应用的密码算法，它有一种类似于指纹的应用。

在网络安全协议中，哈希函数用来处理电子签名，将冗长的签名文件压缩为一段独特的数字信息，像指纹鉴别身份一样保证原来数字签名文件的合法性和安全性。

9.4.5 哈希应用

1、数字签名（数字手印）

SHA-1和MD5都是目前最常用的哈希函数。经过这些算法的处理，原始信息即使只更动一个字母，对应的压缩信息也会变为截然不同的“指纹”，这就保证了经过处理信息的唯一性。为电子商务等提供了数字认证的可能性。

9.4.5 哈希应用

md5散列算法——信息-摘要算法（1991年）

message-digest algorithm）——用于加解密和数字签名

md5的典型应用是对一段信息（message）产生一个128位的信息摘要（message-digest），以防止被篡改。

md5以512位分组来处理输入的信息，且每一分组又被划分为16个32位子分组，经过了一系列的处理后，算法的输出由四个32位(链接变量参数)分组组成，将这四个32位分组级联后将生成一个128位散列值。

9.4.5 哈希应用

例1： md5用于BBS登录时的身份认证

在BBS服务器上，用户的密码都是以md5（或其它类似的算法）经加密后存储在文件系统中。

当用户登录的时候，系统把用户输入的密码计算成md5值，然后再去和预存在服务器中的md5值进行比较，进而确定输入的密码是否正确。

9.4.5 哈希应用

例1： md5用于BBS登录时的身份认证

优点：系统在不知用户密码明码的情况下就可以确定用户登录系统的合法性。这不但可以避免用户的密码被具有系统管理员权限的用户知道，而且还在一定程度上增加了密码被破解的难度。

9.4.5 哈希应用

例2：单纯的数据校验

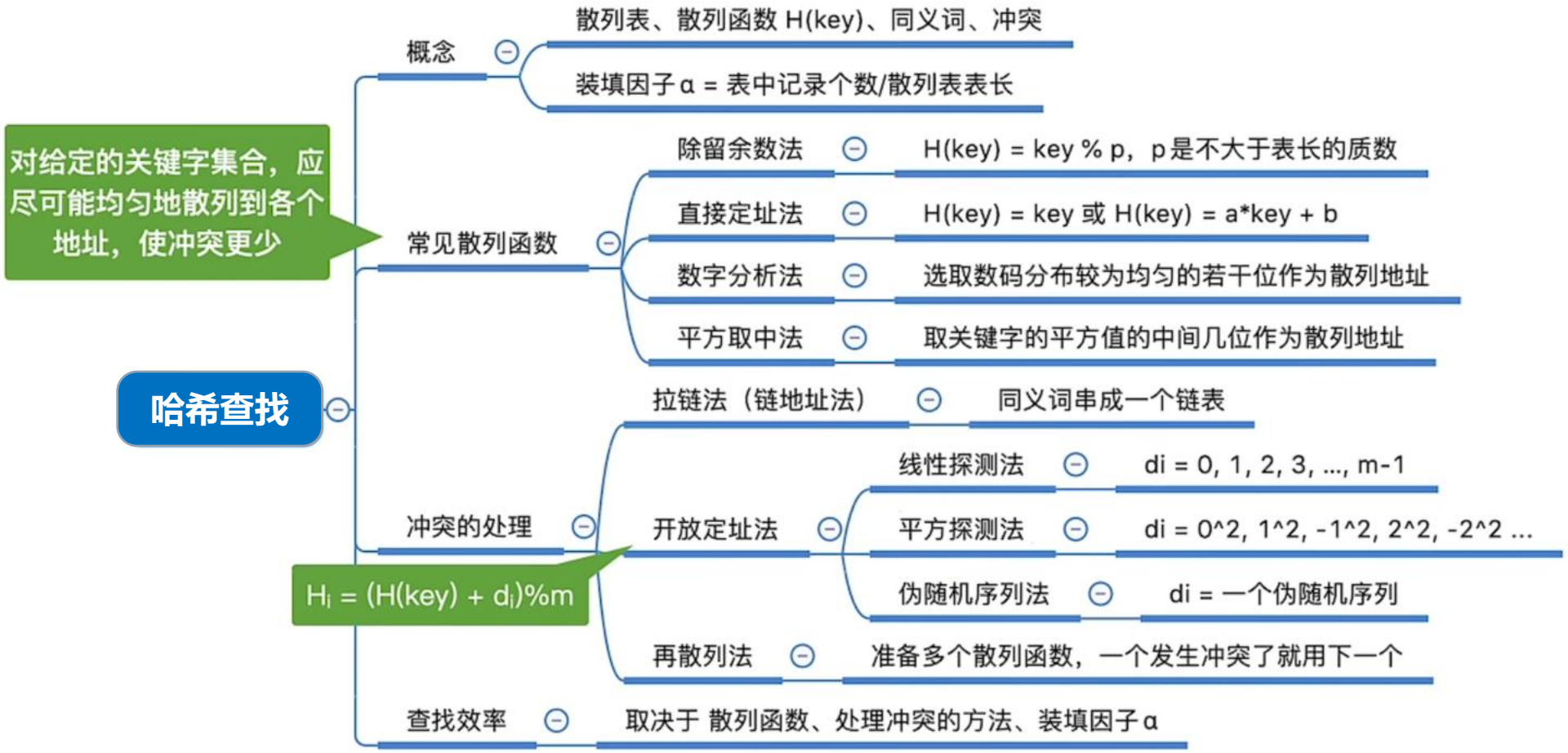
下载光盘镜像文件时 一般会有一个MD5文件记录校验和，以防止下载600MB的文件出现错误导致软件无法安装，这在Linux/FreeBSD等通过网络发布的光盘安装系统中常用。

9.4.5 哈希应用

现在使用的重要计算机安全协议，如SSL，PGP都用哈希函数来进行签名。

但是，如果有人一旦找到两个文件可以产生相同的压缩值，就可以伪造签名，给网络安全领域带来巨大隐患。

总结


哈希查找
对给定的关键字集合，应尽可能均匀地散列到各个地址，使冲突更少
概念
散列表、散列函数 H(key)、同义词、冲突
装填因子 α = 表中记录个数/散列表表长
常见散列函数
除留余数法
H(key) = key % p，p 是不大于表长的质数
直接定址法
H(key) = key 或 H(key) = a*key + b
数字分析法
选取数码分布较为均匀的若干位作为散列地址
平方取中法
取关键字的平方值的中间几位作为散列地址
冲突的处理
$H_i = (H(key) + d_i)\%m$
拉链法（链地址法）
同义词串成一个链表
开放定址法
线性探测法
di = 0, 1, 2, 3, ..., m-1
平方探测法
di = 0^2, 1^2, -1^2, 2^2, -2^2 ...
伪随机序列法
di = 一个伪随机序列
再散列法
准备多个散列函数，一个发生冲突了就用下一个
查找效率
取决于 散列函数、处理冲突的方法、装填因子 α