哈希法又称**散列法、杂凑法以及关键字地址计算法**等，相应的表称为**哈希表。**这种方法的基本思想是：首先在元素的关键字k和元素的存储位置p之间建立一个对应关系f，使得p=f(k)，f称为**哈希函数**。创建哈希表时，把关键字为k的元素直接存入地址为f(k)的单元；以后当查找关键字为k的元素时，再利用哈希函数计算出该元素的存储位置p=f(k)，从而达到按关键字直接存取元素的目的。

   当关键字集合很大时，关键字值不同的元素可能会映象到哈希表的同一地址上，即 k1≠k2 ，但 H（k1）=H（k2），这种现象称为**冲突，**此时称k1和k2为**同义词。**实际中，冲突是不可避免的，只能通过改进哈希函数的性能来减少冲突。

综上所述，哈希法主要包括以下两方面的内容：

 1）如何构造哈希函数

 2）如何处理冲突。

### 8.4.1   哈希函数的构造方法

    构造哈希函数的原则是：①函数本身便于计算；②计算出来的地址分布均匀，即对任一关键字k，f(k) 对应不同地址的概率相等，目的是尽可能减少冲突。

下面介绍构造哈希函数常用的五种方法。

**1． 数字分析法**

如果事先知道关键字集合，并且每个关键字的位数比哈希表的地址码位数多时，可以从关键字中选出分布较均匀的若干位，构成哈希地址。例如，有80个记录，关键字为8位十进制整数d1d2d3…d7d8，如哈希表长取100，则哈希表的地址空间为：00~99。假设经过分析，各关键字中 d4和d7的取值分布较均匀，则哈希函数为：h(key)=h(d1d2d3…d7d8)=d4d7。例如，h(81346532)=43，h(81301367)=06。相反，假设经过分析，各关键字中 d1和d8的取值分布极不均匀， d1都等于5，d8都等于2，此时，如果哈希函数为：h(key)=h(d1d2d3…d7d8)=d1d8，则所有关键字的地址码都是52，显然不可取。

**2． 平方取中法**

当无法确定关键字中哪几位分布较均匀时，可以先求出关键字的平方值，然后按需要取平方值的中间几位作为哈希地址。这是因为：平方后中间几位和关键字中每一位都相关，故不同关键字会以较高的概率产生不同的哈希地址。

例：我们把英文字母在字母表中的位置序号作为该英文字母的内部编码。例如K的内部编码为11，E的内部编码为05，Y的内部编码为25，A的内部编码为01, B的内部编码为02。由此组成关键字“KEYA”的内部代码为11052501，同理我们可以得到关键字“KYAB”、“AKEY”、“BKEY”的内部编码。之后对关键字进行平方运算后，取出第7到第9位作为该关键字哈希地址，如图8.23所示。

|  |  |  |  |
| --- | --- | --- | --- |
| 关键字 | 内部编码 | 内部编码的平方值 | H(k)关键字的哈希地址 |
| KEYA | 11050201 | 122157778355001 | 778 |
| KYAB | 11250102 | 126564795010404 | 795 |
| AKEY | 01110525 | 001233265775625 | 265 |
| BKEY | 02110525 | 004454315775625 | 315 |

图8.23平方取中法求得的哈希地址

**3． 分段叠加法**

      这种方法是按哈希表地址位数将关键字分成位数相等的几部分（最后一部分可以较短），然后将这几部分相加，舍弃最高进位后的结果就是该关键字的哈希地址。具体方法有**折叠法**与**移位法。**移位法是将分割后的每部分低位对齐相加，折叠法是从一端向另一端沿分割界来回折叠（奇数段为正序，偶数段为倒序），然后将各段相加。例如：key=12360324711202065,哈希表长度为1000，则应把关键字分成3位一段，在此舍去最低的两位65，分别进行移位叠加和折叠叠加，求得哈希地址为105和907，如图8.24所示。

1   2   3                    1   2   3

6   0   3                    3   0   6

2   4   7                    2   4   7

1   1   2                    2   1   1

+）   0   2   0               +）  0   2   0

        ————————            —————————

        1   1   0   5                    9   0   7

（a）移位叠加                    (b) 折叠叠加

                      图8.24 由叠加法求哈希地址

**4． 除留余数法**

假设哈希表长为m，p为小于等于m的最大素数，则哈希函数为

h（k）=k  %  p ，其中%为模p取余运算。

例如，已知待散列元素为（18，75，60，43，54，90，46），表长m=10，p=7，则有

    h(18)=18 % 7=4    h(75)=75 % 7=5    h(60)=60 % 7=4

    h(43)=43 % 7=1    h(54)=54 % 7=5    h(90)=90 % 7=6

    h(46)=46 % 7=4

此时冲突较多。为减少冲突，可取较大的m值和p值，如m=p=13，结果如下：

    h(18)=18 % 13=5    h(75)=75 % 13=10    h(60)=60 % 13=8

    h(43)=43 % 13=4    h(54)=54 % 13=2    h(90)=90 % 13=12

    h(46)=46 % 13=7

此时没有冲突，如图8.25所示。

0      1      2     3     4     5      6     7     8     9     10     11    12

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | 54 |  | 43 | 18 |  | 46 | 60 |  | 75 |  | 90 |

                      图8.25  除留余数法求哈希地址

**5． 伪随机数法**

    采用一个伪随机函数做哈希函数，即h(key)=random(key)。

在实际应用中，应根据具体情况，灵活采用不同的方法，并用实际数据测试它的性能，以便做出正确判定。通常应考虑以下五个因素 ：

l         计算哈希函数所需时间 （简单）。

l         关键字的长度。

l         哈希表大小。

l         关键字分布情况。

l         记录查找频率

**8.4.2   处理冲突的方法**

   通过构造性能良好的哈希函数，可以减少冲突，但一般不可能完全避免冲突，因此解决冲突是哈希法的另一个关键问题。创建哈希表和查找哈希表都会遇到冲突，两种情况下解决冲突的方法应该一致。下面以创建哈希表为例，说明解决冲突的方法。常用的解决冲突方法有以下四种：

**1.**         **开放定址法**

这种方法也称**再散列法，**其基本思想是：当关键字key的哈希地址p=H（key）出现冲突时，以p为基础，产生另一个哈希地址p1，如果p1仍然冲突，再以p为基础，产生另一个哈希地址p2，…，直到找出一个不冲突的哈希地址pi ，将相应元素存入其中。这种方法有一个通用的再散列函数形式：

          Hi=（H（key）+di）% m   i=1，2，…，n

    其中H（key）为哈希函数，m 为表长，di称为**增量序列。**增量序列的取值方式不同，相应的再散列方式也不同。主要有以下三种：

l         线性探测再散列

    dii=1，2，3，…，m-1

这种方法的特点是：冲突发生时，顺序查看表中下一单元，直到找出一个空单元或查遍全表。

l         二次探测再散列

    di=12，-12，22，-22，…，k2，-k2    ( k<=m/2 )

    这种方法的特点是：冲突发生时，在表的左右进行跳跃式探测，比较灵活。

l         伪随机探测再散列

    di=伪随机数序列。

具体实现时，应建立一个伪随机数发生器，（如i=(i+p) % m），并给定一个随机数做起点。

例如，已知哈希表长度m=11，哈希函数为：H（key）= key  %  11，则H（47）=3，H（26）=4，H（60）=5，假设下一个关键字为69，则H（69）=3，与47冲突。如果用线性探测再散列处理冲突，下一个哈希地址为H1=（3 + 1）% 11 = 4，仍然冲突，再找下一个哈希地址为H2=（3 + 2）% 11 = 5，还是冲突，继续找下一个哈希地址为H3=（3 + 3）% 11 = 6，此时不再冲突，将69填入5号单元，参图8.26 (a)。如果用二次探测再散列处理冲突，下一个哈希地址为H1=（3 + 12）% 11 = 4，仍然冲突，再找下一个哈希地址为H2=（3 - 12）% 11 = 2，此时不再冲突，将69填入2号单元，参图8.26 (b)。如果用伪随机探测再散列处理冲突，且伪随机数序列为：2，5，9，……..，则下一个哈希地址为H1=（3 + 2）% 11 = 5，仍然冲突，再找下一个哈希地址为H2=（3 + 5）% 11 = 8，此时不再冲突，将69填入8号单元，参图8.26 (c)。

0        1       2      3      4      5       6      7      8       9      10

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  | 47 | 26 | 60 | 69 |  |  |  |  |

         （a） 用线性探测再散列处理冲突

0        1       2      3      4      5       6      7      8       9      10

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | 69 | 47 | 26 | 60 |  |  |  |  |  |

         （b） 用二次探测再散列处理冲突

0        1       2      3      4      5       6      7      8       9      10

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  | 47 | 26 | 60 |  |  | 69 |  |  |

         （c） 用伪随机探测再散列处理冲突

                      图8.26开放地址法处理冲突

从上述例子可以看出，线性探测再散列容易产生“二次聚集”，即在处理同义词的冲突时又导致非同义词的冲突。例如，当表中i, i+1 ,i+2三个单元已满时，下一个哈希地址为i, 或i+1 ,或i+2，或i+3的元素，都将填入i+3这同一个单元，而这四个元素并非同义词。线性探测再散列的优点是：只要哈希表不满，就一定能找到一个不冲突的哈希地址，而二次探测再散列和伪随机探测再散列则不一定。

**2.**         **再哈希法**

    这种方法是同时构造多个不同的哈希函数：

    Hi=RH1（key）  i=1，2，…，k

当哈希地址Hi=RH1（key）发生冲突时，再计算Hi=RH2（key）……，直到冲突不再产生。这种方法不易产生聚集，但增加了计算时间。

**3.**         **链地址法**

    这种方法的基本思想是将所有哈希地址为i的元素构成一个称为**同义词链**的单链表，并将单链表的头指针存在哈希表的第i个单元中，因而查找、插入和删除主要在同义词链中进行。链地址法适用于经常进行插入和删除的情况。

例如，已知一组关键字（32，40，36，53，16，46，71，27，42，24，49，64），哈希表长度为13，哈希函数为：H（key）= key % 13，则用链地址法处理冲突的结果如图8.27所示：

|  |
| --- |
|  |
|  | 哈希表及处理冲突的方法 |

图8.27  链地址法处理冲突时的哈希表

本例的平均查找长度 ASL=(1\*7+2\*4+3\*1)=1.5