

9.2 散列函数的构造方法

❖ 一个“好”的散列函数一般应考虑下列两个因素：

1. 计算简单，以便提高转换速度；
2. 关键词对应的地址空间分布均匀，以尽量减少冲突。

❖ 数字关键词的散列函数构造

1. 直接定址法

取关键词的某个线性函数值为散列地址，即

$$h(\text{key}) = a \times \text{key} + b \quad (a、b \text{ 为常数})$$

地址 $h(\text{key})$	出生年份(key)	人数(attribute)
0	1990	1285万
1	1991	1281万
2	1992	1280万
...
10	2000	1250万
...
21	2011	1180万

$$h(\text{key}) = \text{key} - 1990$$

2. 除留余数法

散列函数为: $h(\text{key}) = \text{key} \bmod p$

例: $h(\text{key}) = \text{key} \% 17$

地址 $h(\text{key})$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
关键词 key	34	18	2	20			23	7	42		27	11		30		15	

□ 这里: $p = \text{Tablesize} = 17$

□ 一般, p 取素数

3. 数字分析法

分析数字关键字在各位上的变化情况，取比较随机的位作为散列地址

□ 比如：取11位手机号码`key`的后4位作为地址：

散列函数为： $h(key) = atoi(key+7)$ (`char *key`)

如果关键词 `key` 是18位的身份证号码：

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	

$$h_1(key) = (key[6] - '0') \times 10^4 + (key[10] - '0') \times 10^3 + (key[14] - '0') \times 10^2 + (key[16] - '0') \times 10 + (key[17] - '0')$$

$$\begin{aligned} h(key) &= h_1(key) \times 10 + 10 && (\text{当 } key[18] = 'x' \text{ 时}) \\ \text{或} &= h_1(key) \times 10 + key[18] - '0' && (\text{当 } key[18] \text{ 为 } '0' \sim '9' \text{ 时}) \end{aligned}$$

4. 折叠法

把关键词分割成位数相同的几个部分，然后叠加

如： 56793542

$$\begin{array}{r} 542 \\ 793 \\ + 056 \\ \hline 1391 \end{array}$$

$$h(56793542) = 391$$

5. 平方取中法

如： 56793542

$$\begin{array}{r} 56793542 \\ \times 56793542 \\ \hline 3225506412905764 \end{array}$$

$$h(56793542) = 641$$

❖ 字符关键词的散列函数构造

1. 一个简单的散列函数——ASCII码加和法

对字符型关键词 key 定义散列函数如下：

$$h(key) = (\sum key[i]) \bmod TableSize$$

冲突严重：a3、
b2、c1；
eat、tea；

2. 简单的改进——前3个字符移位法

$$h(key) = (key[0] \times 27^2 + key[1] \times 27 + key[2]) \bmod TableSize$$

3. 好的散列函数——移位法

涉及关键词所有 n 个字符，并且分布得很好：

$$h(key) = \left(\sum_{i=0}^{n-1} key[n-i-1] \times 32^i \right) \bmod TableSize$$

仍然冲突：string、street、
strong、structure等等；
空间浪费： $3000/26^3 \approx 30\%$

❖ 如何快速计算:

$$h(\text{"abcde"}) = 'a' * 32^4 + 'b' * 32^3 + 'c' * 32^2 + 'd' * 32 + 'e'$$

```
Index Hash ( const char *Key, int TableSize )
{
    unsigned int h = 0;      /* 散列函数值, 初始化为0 */
    while ( *Key != '\0' )   /* 位移映射 */
        h = ( h << 5 ) + *Key++;
    return h % TableSize;
}
```