#hash \_table

**1.1描述一种hash table的实现方法**

1） 除法散列法: p ，令 h(k ) = k mod p ，这里， p 如果选取的是比较大的素数，效果比较好。而且此法非常容易实现，因此是最常用的方法。最直观的一种，上图使用的就是这种散列法，公式： index = value % 16，求模数其实是通过一个除法运算得到的。

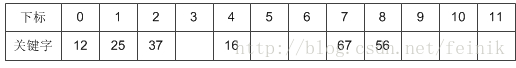
2） 平方散列法 :求index频繁的操作，而乘法的运算要比除法来得省时。公式： index = (value \* value) >> 28 （右移，除以2^28。记法：左移变大，是乘。右移变小，是除）

3） 数字选择法:如果关键字的位数比较多，超过长整型范围而无法直接运算，可以选择其中数字分布比较均匀的若干位，所组成的新的值作为关键字或者直接作为函数值。

4） 斐波那契（Fibonacci）散列法:平方散列法的缺点是显而易见的，通过找到一个理想的乘数index = (value \* 2654435769) >> 28

**1.2冲突处理：**

Hash算法解决冲突的方法一般有以下几种常用的解决方法   
1， 开放定址法：   
所谓的开放定址法就是一旦发生了冲突，就去寻找下一个空的散列地址，只要散列表足够大，空的散列地址总能找到，并将记录存入   
公式为：fi(key) = (f(key)+di) MOD m (di=1,2,3,……,m-1)   
※ 用开放定址法解决冲突的做法是：当冲突发生时，使用某种探测技术在散列表中形成一个探测序列。沿此序列逐个单元地查找，直到找到给定的关键字，或者   
碰到一个开放的地址（即该地址单元为空）为止（若要插入，在探查到开放的地址，则可将待插入的新结点存人该地址单元）。查找时探测到开放的地址则表明表   
中无待查的关键字，即查找失败。   
比如说，我们的关键字集合为{12,67,56,16,25,37,22,29,15,47,48,34},表长为12。 我们用散列函数f(key) = key mod l2   
当计算前S个数{12,67,56,16,25}时，都是没有冲突的散列地址，直接存入：

   
计算key = 37时，发现f(37) = 1，此时就与25所在的位置冲突。   
于是我们应用上面的公式f(37) = (f(37)+1) mod 12 = 2。于是将37存入下标为2的位置：   


2， 再哈希法：   
再哈希法又叫双哈希法，有多个不同的Hash函数，当发生冲突时，使用第二个，第三个，….，等哈希函数  
计算地址，直到无冲突。虽然不易发生聚集，但是增加了计算时间。

3， 链地址法：   
链地址法的基本思想是：每个哈希表节点都有一个next指针，多个哈希表节点可以用next指针构成一个单向链表，被分配到同一个索引上的多个节点可以用这个单向   
链表连接起来，如：   
键值对k2, v2与键值对k1, v1通过计算后的索引值都为2，这时及产生冲突，但是可以通道next指针将k2, k1所在的节点连接起来，这样就解决了哈希的冲突问题   
  
4， 建立公共溢出区：   
这种方法的基本思想是：将哈希表分为基本表和溢出表两部分，凡是和基本表发生冲突的元素，一律填入溢出表

**1.3 素数原理**

质数是独一无二的数字。它们是唯一的，任何其他数的质数的乘积都有可能是唯一的(不像质数本身那样独特)，因为一个质数是用来组成它的。此属性用于散列函数。

给定一个字符串“Samuel”，您可以生成一个唯一的散列，通过将每个组成数字或字母乘以一个素数并将它们加起来。这就是为什么使用质数的原因。

**2.设计一个洗牌的算法，并说出算法的时间复杂度。**

（1）全局洗牌法

a）首先生成一个数组，大小为54，初始化为1~54

b）按照索引1到54，逐步对每一张索引牌进行洗牌，首先生成一个余数 value = rand %54，那么我们的索引牌就和这个余数牌进行交换处理

c）等多索引到54结束后，一副牌就洗好了

（2）局部洗牌法：索引牌从1开始，到54结束。这一次索引牌只和剩下还没有洗的牌进行交换， value = index + rand（） %（54 - index）

算法复杂度是O(n)