完整的街道地址，散列函数可以包括街道地址的几个字符，也许还有城市名和邮政编码的几个字符。有些程序设计人员通过只使用奇数位置上的字符来实现他们的散列函数，这里有这么一层想法：用计算散列函数节省下的时间来补偿由此产生的对均匀地分布的函数的轻微干扰。

```
1       /**
2        * A hash routine for String objects.
3        * @param key the String to hash.
4        * @param tableSize the size of the hash table.
5        * @return the hash value.
6        */
7       public static int hash( String key, int tableSize )
8       {
9           int hashVal = 0;
10
11          for( int i = 0; i < key.length( ); i++ )
12              hashVal = 37 * hashVal + key.charAt( i );
13
14          hashVal %= tableSize;
15          if( hashVal < 0 )
16              hashVal += tableSize;
17
18          return hashVal;
19      }
```

图 5-4 一个好的散列函数

剩下的主要编程细节是解决冲突的消除问题。如果当一个元素被插入时与一个已经插入的元素散列到相同的值，那么就产生一个冲突，这个冲突需要消除。解决这种冲突的方法有几种，我们将讨论其中最简单的两种：分离链接法和开放定址法。

5.3 分离链接法

解决冲突的第一种方法通常叫做**分离链接法**(separate chaining)，其做法是将散列到同一个值的所有元素保留到一个表中。我们可以使用标准库表的实现方法。如果空间很紧，则更可取的方法是避免使用它们(因为这些表是双向链接的并且浪费空间)。本节我们假设关键字是前 10 个完全平方数并设散列函数就是 $hash(x) = x \bmod 10$(表的大小不是素数，用在这里是为了简单)。图 5-5 对此做出更清晰的解释。

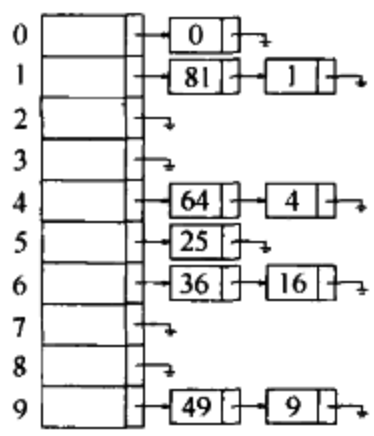

图 5-5 分离链接散列表

为执行一次查找，我们使用散列函数来确定究竟遍历哪个链表。然后我们再在被确定的链表中执行一次查找。为执行 insert，我们检查相应的链表看看该元素是否已经处在适当的位置(如果允许插入重复元，那么通常要留出一个额外的域，这个域当出现匹配事件时增 1)。如果这个元素是个新元素，那么它将被插入到链表的前端，这不仅因为方便，还因为常常发生这样的事实：新近插入的元素最有可能不久又被访问。

实现分离链接法所需要的类架构如图 5-6 所示。散列表存储一个链表数组，它们在构造方法中被指定。