散列存储的**基本思想**是以关键字值为自变量，通过一定的函数关系（称为散列函数或哈希函数），计算出对应的函数值（称为哈希地址），以这个值作为数据元素的地址，并将该数据元素存入到相应地址的存储单元中去。查找时再根据要查找的关键字采用同样的函数计算出哈希地址，然后直接到相应的存储单元中去取要找的数据元素。

经过哈希函数变换后，可能将不同的关键字映射到同一个哈希地址上，这种现象称为**冲突**。实际上，**冲突不可能避免，只能尽可能减少**。在构造散列表时，除了需要选择一个好的哈希函数外，还需要找到一种处理冲突的方法。

根据设定的哈希函数H(key)和所选中的处理冲突的方法，将一组关键字映射到一个有限的、地址连续的地址集(区间)上，并以关键字在地址集中的”象”作为相应数据元素在表中的存储位置，如此构造所得的查找表称之为哈希表。

**构造一个哈希表要解决的问题**

**1.构造好的哈希函数**

**2.制定解决冲突的方法**

**常用的哈希函数**

1.直接定址法

H(key)=a\*key+b

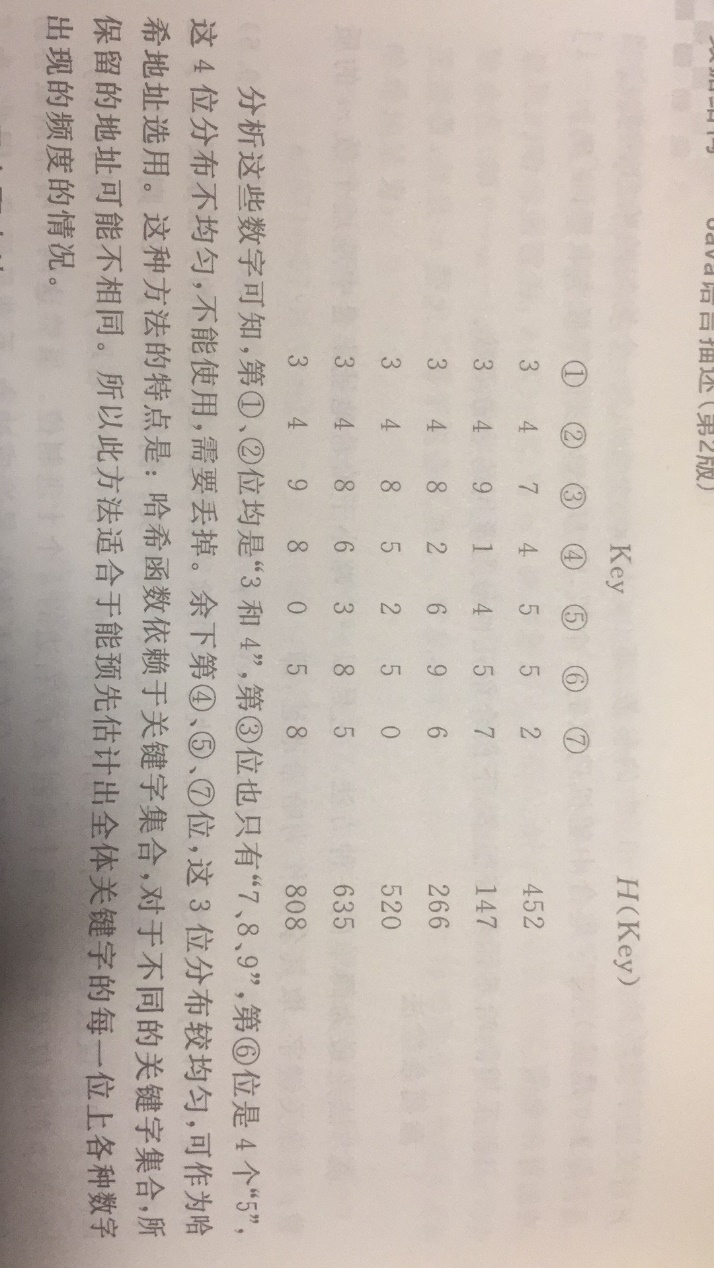
适用于关键字分布基本连续的情况，不连续时空位多，浪费空间

2.除留余数法

H(key)=key%p p为不大于表长m但最接近于或等于m的质数p

3.数字分析法

对于关键字的位数比存储区域的地址码位数多的情况，可以采用对关键字的各位进行分析，丢掉分布不均匀的位，留下分布均匀的位作为哈希地址。



适用于已知关键字的集合

4.平法取中法

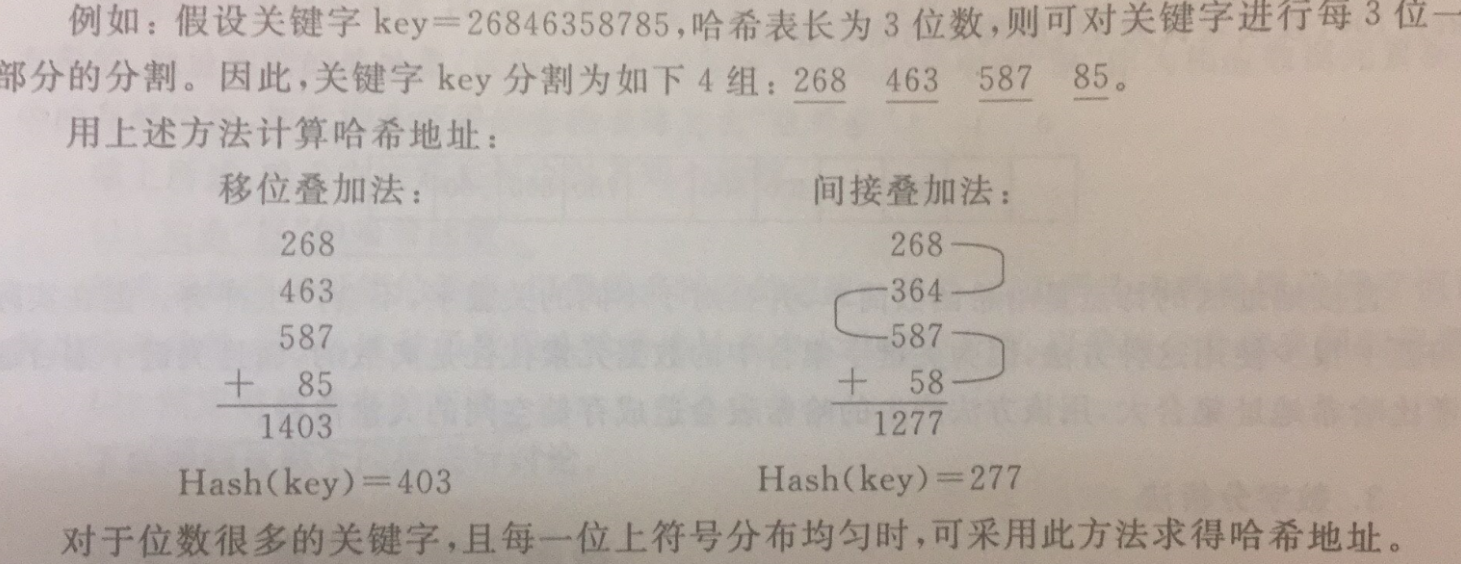
取关键字的平方值的中间几位作为散列地址

散列地址分布比较均匀

适用于关键字的每一位取值都不够均匀或均小于散列地址所需位数的情况

5.折叠法

将关键字分割成位数相同的几部分，取部分的叠加作为散列地址。



6.随机数法

H(key)=random(key)

适用于关键字长度不想等的时候。

**处理冲突的方法**

1.开放定址法

Hi=（H（key）+di）% m m为散列表长

（1）线性探测法

di=1，2，3 … m-1

在冲突发生的时候，顺序查看表中下一个单元，直至找出空闲空间为止。

容易出现堆积，查找效率低。

堆积现象的原因是查找序列过分集中在发生冲突的存储单元的后边，没有在整个散列表空间上分散开来。

（2）平方探测法

di=12 ，-12，22，-22 … k2，-k2 k<=m/2,m必须可以表示成为4k+3的质数

能够避免堆积，但是不能探测到散列表的所有单元，至少一半

（3）双哈希函数探测法

di=H2（key）

这个方法使用两个哈希函数，先用第一个函数计算出地址，若产生冲突，再用第二个函数确定移动的步长因子

2.链地址法（拉链法）

将所有具有相同哈希地址的不同关键字的数据连接到同一个单链表中。若选中的表长为m，则可将哈希表定义为一个由m个头指针组成的指针数组，凡是哈希地址为i的数据元素，均可以以节点的形式插入到T[i]为头指针的单链表中。

适用于经常进行插入和删除的情况。

没有冲突，每次查找和删除的时间复杂度为O(1),若每次都插入到单链表最前边，则插入操作的时间复杂度为O(1)，查找成功的平均比较次数为m/2，查找不成功的比较次数为m，查找和删除的时间复杂度为O(m)。

内部结构：一个链表类型的数组

Private LinkedList[] table;

初始化

建立表长m的数组，记得每一个数组单元都要初始化

**public HashTable(){**

**table=null;**

**}**

**public HashTable(int m){ //初始化**

**table = new LinkedList[m];**

**for(int i=0;i<m;i++) //数组每一项都要进行初始化**

**table[i]= new LinkedList<Integer>();**

**}**

**功能**

**1.产生hash地址（重要）**

**public int hash(int key){ //生成散列地址**

**return key % table.length;**

**}**

**2.插入**

**public void insert(Integer element){ //插入**

**int key = hash(element); //生成散列地址**

**table[key].addFirst(element); //插入到地址链表的头部**

**}**

**3.删除**

**public void remove(Integer element){ //删除**

**int key = hash(element);**

**table[key].remove(element);**

**}**

**4.查找**

**public Object search(Integer element){ //查找**

**int key = hash(element);**

**int index = table[key].indexOf(element);**

**if(index>=0)**

**return table[key].get(index); //查找成功，返回该数据值**

**else**

**return null; //查找失败，返回null**

**}**

**散列法的查找效率**

**取决于散列函数、处理冲突的方法、装填因子。**

**装填因子**α：一个表的装满程度

a= 表中记录数n / 散列表的长度m

**装填因子越大，表越满，冲突可能性越大，但不依赖于m,n**