# 哈希表实现和哈希冲突

所谓哈希，就是给定一个数据key，对它编码成一个数字，然后再通过这个数字，做到可以索引给定的另一个数据value.这就是哈希的<key,value>模型，key一定是可哈希对象(字符串，浮点数，整数)，value不限。

对key编码的过程需要哈希函数，设它为int hash(key)，哈希函数在后面说，它有多种实现方法，哈希函数是函数，是一对一的关系，一个key只能对应一个value,且每个hash(key)的数值一定不同，否则就会出现哈希冲突，哈希函数设计的的好坏的标准就是产生冲突的几率大小。

储存这些数据的数据结构叫哈希表或者散列表，是用普通数组来储蓄，处理哈希冲突的方法有以下。

## **开放地址法**

所谓开放地址法，就是从发生冲突的那个单元起，按照一定的次序，从哈希表中找到一个空闲的单元，然后把发生冲突的元素存入到该单元的一种方法。这个数组大小也决定了这个哈希表最多能装多少元素，假设数组是d[]长度是len。

对于某个哈希函数hash()，如果发生冲突,也就是hash(data1)和之前的某个数据的hash相同。就是要处理冲突，开放地址法3分以下几种：

### 线性探测

循环去看下一个位置或者上一个位置是否被占用，直到发现没有被占用的位置为止。特别的：如果正着走到了d[len-1]要从d[0]探测，表现为代码是i=(i+1)%len；逆向走到d[0]要从d[len-1]开始探测，表现为代码是i=(i-1+len)%len。

显然这样效率不高，出现数据”堆积”现象时，退化成o(len)的时间处理冲突。

### 补偿线性探测法

将线性探测的步长从 1 改为 q ，表现为代码是i＝(i＋1) % len 改为：i＝(i＋q) % len，而且要求 len 与 q是互质的，以便能探测到哈希表中的所以位置

### 随机探测法

将线性探测的步长从 1 改为 随机数randq ，表现为代码是i＝(i＋1) % len 改为：i＝(i＋randq) % len， randq可以每次随机生成，也可以事先生成随机序列，每次冲突就从这里面取。

### 平方探测法

将线性探测的步长从 1 改为 某个平方数平q2=1\*1 ,2\*2,3\*,4\*4………，表现为代码是i＝(i＋1) % len 改为：i＝(i＋q2) % len这种方法不一定能探测到哈希表所以位置，但是可以探测到大部分位置，这就足够了。

## 拉链法(连地址法)

这种方法里，不再用普通数组当做哈希表，而是用链表数组当做哈希表。这个数组大小也决定了这个哈希表最多能装多少元素，假设数组是d[]长度是len。

这种方法解决哈希中途比较容易，对于某个哈希函数hash()，如果发生冲突,也就是hash(data1)和之前的某个数据的hash相同。直接在对应hash(data1) 位置的链表里添加一个位置，放进data1即可。

这样解决哈希冲突复杂度是o(1)，但缺点是在查找这个元素时，要用顺序查找逐个遍历，假如哈希函数垃圾冲突太多，还是慢，但如果哈希函数好的话，这个方法很快，是java哈希表内部处理冲突的方法。

## 多哈希法

多哈希法是建立很多哈希函数，一个哈希函数冲突了用另一个计算，这种方法计算耗时，但可以保证几乎不会冲突。

## 建立公共溢出区

将哈希表分为公共表和溢出表，当溢出发生时，将所有溢出数据统一放到溢出区

# 字符串的哈希

一个很朴素的想法就是令哈希值是每一位阿斯克码之和，这样会导致”ab”和”ba”的哈希值相同，这叫做哈希冲突，要处理冲突，这样就浪费了时间。一个好的算法会尽可能减少哈希冲突，显然朴素的想法哈希冲突很多。

下面介绍几种经典的较好的哈希算法：

这里仅仅介绍哈希算法，算出来的数值太大，还不能作为数组下标来映射，顶多完成P3370这道题(给你n个字符串问你有多少个不同的)，筛去重复数据。

此外这里没有对哈希冲突(有的地方叫哈希碰撞)做处理，一旦有冲突，会直接导致答案错误，这个在map实现里说

## BKDR Hash：

它的主要思路是选取恰当的进制，可以把字符串中的字符看成一个大数字中的每一位数字，然后取一个进制Q，和一个模数P,利用同余定理算出一个小于P的数值，选取好的Q和P可以让冲突变少，Q的选取要求比字符集字符数多，p则是一个大数，p最少也要10^9级别，P越小冲突概率越高,P取素数也可以降低冲突概率，和生日悖论有关。此外对于每个字符对应的数字，a-z千万不能让a对应0，这样不能区分ab和b，所以应该a-z对应1-27.或者干脆直接用阿斯克码当中代表的数(阿斯科马的0代表空，不会对应有效字符)。P的选择也有个巧妙方法是直接选择2^64-1这样利用unsigned long long类型，可以自动取模。

## Md5 Hash：

计算机的存储单位为字节，一个字节对应8个二进制位，共可以表示2^8也就是256种状态。若表示数的话，最多只能表示256个数。

如一个字节可以表示非负整数的0~255，而表示更大的数，则需要占用多个字节，如表示256至少需要两个字节。

256的二进制形式为 00000001 00000000。这样在计算机存储上就存在一个问题：是先存储00000001这个字节，还是先存储00000000这个字节呢？实际上，采用这两种存储方式的都有，取决于CPU架构和编译器。这就引出了字节序的概念。

来看” hello”这个字符串，它是阿斯克字符构成，1个字符1位，那么长度是5，意味着由5个01字符串编码而成。每个字符是阿斯克码，展开2进制是这样：

**01101000** 01100101 **01101100** 01101100 **01101111**

**第一步:处理原文**

首先，我们计算出原文的比特长度，

对512求余的结果，如果不等于448，就需要填充原文使得原文对512求余的结果等于448。填充的方法是第一位填充1，其余位填充0。填充完后，信息的长度就是512\*N+448。

之后，用剩余的位置（512-448=64位）记录原文的真正长度，把长度的二进制值补在最后。这样处理后的信息长度就是512\*(N+1)。