# 常见问题解决思路

## 大整数数据相加

可以使用数组

## 大整数，删去k个数字后得到最小值

从首位开始比较，如果第一位比第二位数字大，则删除第一位，依次类推。

# LRU算法的基本思路

使用链表数据结构，将最新查询的数据放到链表的最右端，假设缓存容量已经达到上限，则把最左端的数据删除。java的linkedHashMap已经对哈希链表做了很好的实现。

# Hash算法

Hash就是把任意长度的输入，通过散列算法，变换成固定长度的输出，该输出就是散列值。

两个不同的输入值，根据同一散列函数计算出的散列值相同的现象叫做碰撞。

常见的Hash函数有以下几个：

直接定址法：直接以关键字k或者k加上某个常数（k+c）作为哈希地址。

数字分析法：提取关键字中取值比较均匀的数字作为哈希地址。

除留余数法：用关键字k除以某个不大于哈希表长度m的数p，将所得余数作为哈希表地址。

分段叠加法：按照哈希表地址位数将关键字分成位数相等的几部分，其中最后一部分可以比较短。然后将这几部分相加，舍弃最高进位后的结果就是该关键字的哈希地址。

平方取中法：如果关键字各个部分分布都不均匀的话，可以先求出它的平方值，然后按照需求取中间的几位作为哈希地址。

伪随机数法：采用一个伪随机数当作哈希函数。

衡量一个哈希函数的好坏的重要指标就是发生碰撞的概率以及发生碰撞的解决方案。任何哈希函数基本都无法彻底避免碰撞，常见的解决碰撞的方法有以下几种：

开放定址法：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。

链地址法：将哈希表的每个单元作为链表的头结点，所有哈希地址为i的元素构成一个同义词链表。即发生冲突时就把该关键字链在以该单元为头结点的链表的尾部。

再哈希法：当哈希地址发生冲突用其他的函数计算另一个哈希函数地址，直到冲突不再产生为止。

建立公共溢出区：将哈希表分为基本表和溢出表两部分，发生冲突的元素都放入溢出表中。

在Java中，保存数据有两种比较简单的数据结构：数组和链表。

数组的特点是：寻址容易，插入和删除困难；而链表的特点是：寻址困难，插入和删除容易。

在同一个版本的Jdk中，HashMap、HashTable以及ConcurrentHashMap里面的hash方法的实现是不同的。再不同的版本的JDK中（Java7 和 Java8）中也是有区别的。

X % 2^n = X & (2^n - 1)：Java之所有使用位运算(&)来代替取模运算(%)，最主要的考虑就是效率