# HashMap

1.给你N（1<N<10）个自然数,每个数的范围为（1~100）。现在让你以最快的速度判断某一个数是否在这N个数内，不得使用已经封装好的类，该如何实现。

N:5

10 50 60 1 5

判断7在不在：排序（没必要），遍历，（枚举），数组下标；

a[]=new int[101] => a[10] = 1,a[50]=1,a[7]=-1;查找的时间复杂度O(1) 直接判断a[7]的值

数据太大存不了，浪费了90%的空间

2.给你N（1<N<10）个自然数,每个数的范围为（1~10000000000）。现在让你以最快的速度判断某一个数是否在这N个数内，不得使用已经封装好的类，该如何实现。 A[] = new int[N+1]？

N:5

11 52 63 4 5,999999999；取余：

判断7在不在

散列表英文就是HashTable，也就是我们经常说的哈希表，大家肯定经常听到，其实刚刚上面我们的那个例子就是运用了散列表的思想来解决的

散列表用的是数组支持按照下标随机访问数据的特性，所以散列表其实就是数组的一种扩展，由数组演化而来。可以说，如果没有数组，就没有散列表

实际上，这个例子已经用到了散列的思想。在这个例子里，N自然数，并且与数组的下标形成一一映射，所以利用数组支持根据下标随机访问的

特性。

查找的时间复杂度是O(1)这一特性，就可以实现快速判断元素是否存在序列当中。

什么是散列函数？

拿我们上面的例子来说我们对N取模，其实这就是一个散列函数。也就是大家经常看到的Hash(key)，这个Hash函数就是我们说的散列函数。我们是通过它来计算散列的值的。

现在来看上面例子的另外一种情况：

N:10

11 52 62 63 4 5,999999999；取余：我们对N进行取余.:x%n 散列函数的一种。

11 % 10 = 1 =>a[1] = 11

52 % 10 = 2 => a[2] = 52

62 % 10 = 2？ 就被称之为hash冲突

我要查找52 52%10 => 2 在去找到a[2] = 52?如果等于 则存在，不等于则不存在.

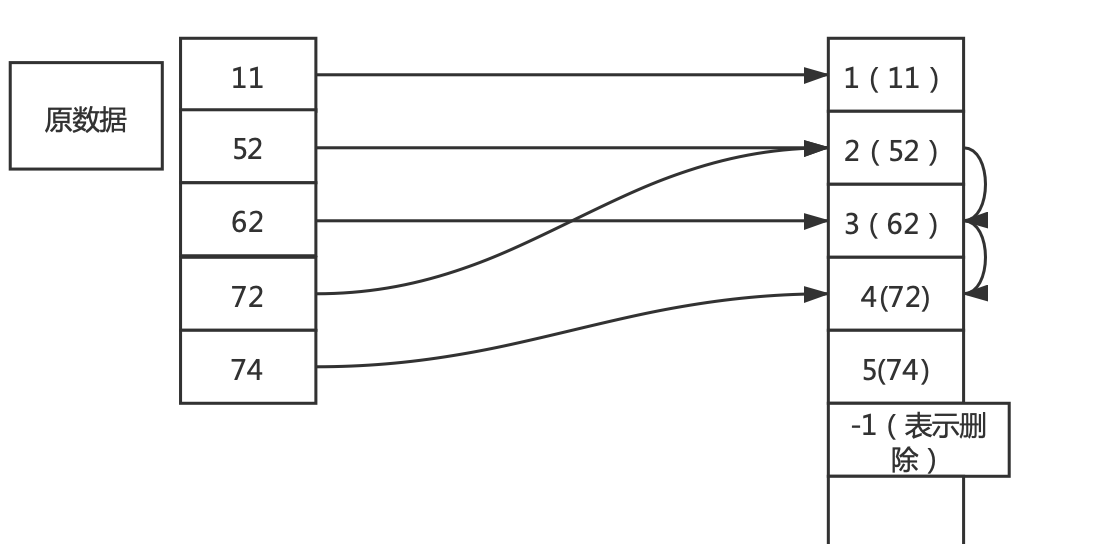
62 % 10=>2 a[2] = 52 不等于62

通过上面另外一种情况我们可以看到Hash冲突了。其实这是一个无法避免的问题。

比如MD5这些加密的Hash算法等等在某些情况下都会出现冲突

我们几乎无法找到一个完美的无冲突的散列函数，即便能找到，付出的时间成本、计算成本也是很大的，所以针对散列冲突问题，我们需要通过其他途径来解决。

开放寻址：开放寻址法的核心思想是，如果出现了散列冲突，我们就重新探测一个空闲位置，将其插入。



当我们往散列表中插入数据时，如果某个数据经过散列函数散列之后，存储位置已经被占用了，我们就从当前位置开始，依次往后查找，看是否有空闲位置，直到找到为止。

那么怎么查找呢？

比如查找20：首先根据Hash函数 得到0，然后往后依次比较数值，如果找到为空

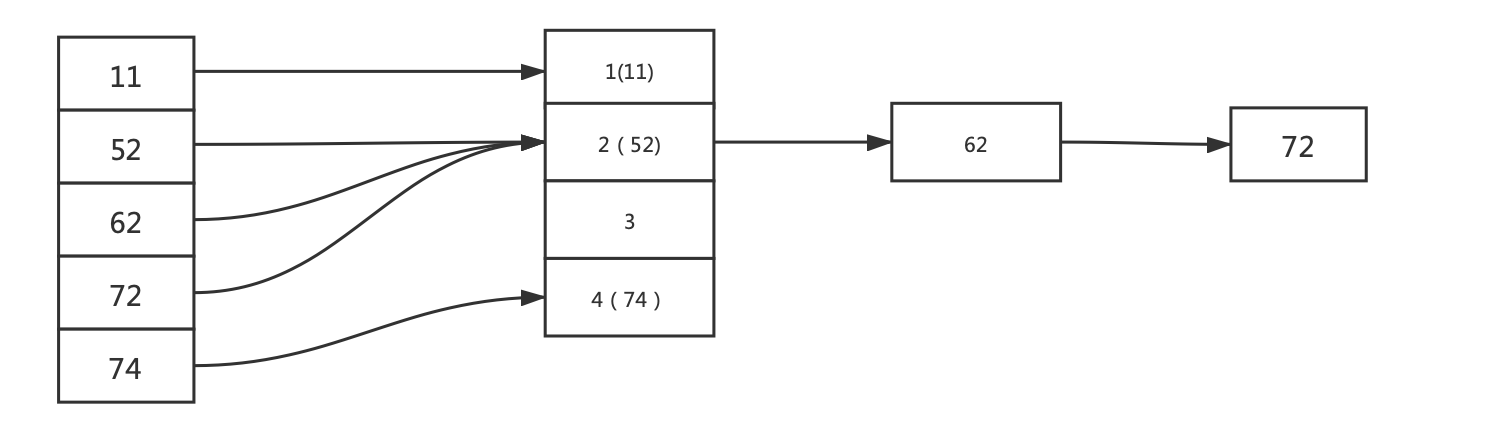
的位置还没找到即不存在

缺点：

1.删除需要特殊处理

2.插入的数据如果过多会导致散列表很多冲突查找可能会退化成遍历

链路地址：其实就是使用链表。链表法是一种更加常用的散列冲突解决办法，相比开放寻址法，它要简单很多。我们来看这个图，在散列表中，每个key会对应一条链表，所有散列值相同的元素我们都放到相同槽位对应的链表中。上面的例子就会变成这样。



由于链表这种结构确实存在一些缺点，所以在我们的JDK中对之进行了优化，引入了更高效的数据结构：

1.初始大小：HashMap默认的初始大小是16，这个默认值是可以设置的，如果事先知道大概的数据量有多大，可以通过修改默认初始大小，减少动态扩容的次数，这样会大大提高HashMap的性能。

2.动态扩容：最大装载因子默认是0.75，当HashMap中元素个数超过0.75\*capacity（capacity表示散列表的容量）的时候，就会启动扩容，每次扩容都会扩容为原来的两倍大小。

3.Hash冲突解决办法：JDK1.7底层采用链表法。

在JDK1.8版本中，为了对HashMap做进一步优化，我们引入了红黑树。而当链表长度太长（默认超过8）时，链表就转换为红黑树。我们可以利用红黑树快速增删改查的特点，提高HashMap的性能。当红黑树结点个数少于8个的时候，又会将红黑树转化为链表。因为在数据量较小的情况下，红黑树要维护平衡，比起链表来，性能上的优势并不明显。

4.Hash函数：JDK的hash函数非常经典，建议大家收藏，以后都可以用。

int hash(Object key) {

int h = key.hashCode()；

return (h ^ (h >>> 16)) & (capitity -1); //capicity表示散列表的大小,最好使用2的整数倍

}

如何设计Hash:

1.必须要高效：即插入，删除 查找必须要快

2.内存：不能占用太多的内存,如果太大，考虑用其他的结构，比如B+Tree，10亿，存硬盘的算法：mysql B+tree

3.Hash函数：这个要根据实际情况考虑

4.扩容:就是预估数据的大小，HashMap 默认的空间是16？ 我知道我要存10000个数，2^n > 10000 or 2^n-1

5.Hash冲突后怎么解决：链表 数组。

Hash的应用

1.加密：MD5 哈希算法。还是存在密码冲突 128位的二进制串，可以表示2^128次，md5(md5(),”1231”),不可逆

2.怎么判断视频是否是重复的？Md5（）；128位

3.相似性检测：论文检测，指纹算法。会把每个论文计算出一个指纹，汉明距离。

4.负载均衡：nginx，2台服务器；可以根据ip计算hash，再做一个取模运算。

5.分布系统：数据分库问题。一个10亿的数据的搜索词，一台机器存不下。要分成10个文件。Hash(key)%10 = > 就可以知道某个key在哪一个文件？扩大成数据库的 分表（10张表） id%10 =（）。

6.分布式存储的时候：问题来了 如果我加了一张表。原来是10张，现在是11张了 怎么办？要重新计算，分配的时候查询怎么办？数据量很大，迁移的不是太多了吗？（一致性哈希 哈希环）

8.查找算法:hashMap查找

一致性Hash：哈希环。

假设我们有k个表，数据的hash值范围：[0,Integer.max].我们把整个数据范围划分成n个区间（n个区间要远远大于我们的k），每一个表就是分配到n/k的区间。当有新的表要来了，我们只需要将某几个小的区间数据迁移就可以了。这是一个环形结构，其根本思想就是分段了。