**程序员常说的「哈希表」是个什么鬼?**

2017-10-01 来源：[人人都是产品经理](http://www.woshipm.com/pmd/805326.html) 作者：人人都是产品经理 分类：[新鲜干货](http://www.yixieshi.com/kjqy/it)

　　今天聊聊「哈希表」，「哈希表」主要作用在于高效查找。



　　在编程实现中，常常面临着两个问题：存储和查找，存储和查找的效率往往决定了整个程序的效率。

　　脑补下，你在家里忘记了指甲刀放在哪里，通常要在你家所有抽屉中顺序寻找，直到找到，最差情况下，有N个抽屉，你就要打开N个抽屉。这种存储方式叫数组，查找方法称为「遍历」。

　　脑补下，你是一个整理控，所有物品必须分门别类放入整理箱，再将整理箱编号，比如1号放入针线，2号放入证件，3号放入细软。这种存储和查找方式称为「哈希」，如果[这个](http://www.yixieshi.com/tag/%e8%bf%99%e4%b8%aa)时候要查找护照，你不许要再翻所有抽屉，直接可在2号整理箱中获取，通常只用一次查找即可，[如何](http://www.yixieshi.com/tag/%e5%a6%82%e4%bd%95)编号整理箱，称为哈希算法。

　　同样是查找，差距怎么那么大涅~，假设[我们](http://www.yixieshi.com/tag/%e6%88%91%e4%bb%ac)有100亿条数据记录，那差距就变得明显，遍历需要查找最多100亿次，最少1次，哈希只需1次。

　　让我们正式介绍哈希和哈希算法，哈希也称散列，哈希表是一种与数组、链表等不同的数据结构，与他们需要不断的遍历比较来查找的办法，哈希表设计了一个映射关系f(key)= address，根据key来计算存储地址address，这样可以1次查找，f既是存储数据过程中用来指引数据存储到什么位置的函数，也是将来查找这个位置的算法，叫做哈希算法。

　　让我们举个例子，比如下面这几个人物，按数组存储：



　　这样我要找到大胸姐的电话号码，需要顺序查找对比整个数组，第一个余罪，不是，第二个不是，第三个不是，直到第四个找到大胸姐。

　　如果以hash存储呢?首先让我们来看看如何设计哈希算法，哈希算法可以随意设计，教科书上一般会说以下几种方法：直接定址法，平方取中法，除数取余法，哈希算法的本质上是计算一个[数字](http://www.yixieshi.com/tag/%e6%95%b0%e5%ad%97)，如果用这几种方法讲解会稍显晦涩，我们假设我们的哈希算法是取姓名的首字母。所以f(余罪) = y， f(傅老大) = f，f(沈嘉文) = s，f(大胸姐) = d。

　　构建的hash表如下：



　　我们看到他们分别以姓名首字母的位置插入到这一张表格中，这样我们构建了这样一个Key-Value表格，此表就是哈希表，也称为Hash Table。未来，当我们要查找余罪的时候，通过计算，余罪在y位置，可以通过1次查找，找到这条记录，也即手机号。

　　这个时候有客官问了，那以首字母为哈希函数的话，应该有[很多](http://www.yixieshi.com/tag/%e5%be%88%e5%a4%9a)比如以y的姓名啊，这个时候就不是一次查找了吧，其实有很多条记录都映射到一个位置上，称为哈希冲突。

　　哈希冲突是跟哈希函数的设计正相关的，你的随机性越大，那么产生哈希冲突的可能性越小，在小概率下，如果还有冲突怎么办，这个时候要做些有损的设计，比如如果有两个首字母为y的姓名，那么可以接到余罪的后面，当查找的时候，需要先查找到y，然后再顺序查找，如图所示：



　　还有一些解决哈希冲突的办法叫「哈希再哈希」，也就是针对第一次哈希的结果再进行一次hash来减小冲突的概率。

　　这就是Hash表，首先Ta是一种数据结构，是一种效率极高的查找方式，哈希表的核心在于哈希函数的设计，哈希冲突了不要紧，我们要增加随机性以及对冲突进行适当的有损化的处理。

本文链接: <http://www.yixieshi.com/95421.html> (转载请保留)

[**哈希表的理解**](https://www.cnblogs.com/lchzls/p/6714079.html)

哈希表是种数据结构，它可以提供快速的插入操作和查找操作。第一次接触哈希表时，它的优点多得让人难以置信。不论哈希表中有多少数据，插入和删除（有时包括侧除）只需要接近常量的时间即0(1）的时间级。实际上，这只需要几条机器指令。

　　对哈希表的使用者——人来说，这是一瞬间的事。哈希表运算得非常快，在计算机程序中，如果需要在一秒种内查找上千条记录通常使用哈希表（例如拼写检查器)哈希表的速度明显比树快，树的操作通常需要O(N)的时间级。哈希表不仅速度快，编程实现也相对容易。

哈希表也有一些缺点它是基于数组的，数组创建后难于扩展某些哈希表被基本填满时，性能下降得非常严重，所以程序虽必须要清楚表中将要存储多少数据（或者准备好定期地把数据转移到更大的哈希表中，这是个费时的过程）。

如果不需要有序遍历数据，并且可以提前预测数据量的大小。那么哈希表在速度和易用性方面是无与伦比的。

**哈希表算法-哈希表的概念及作用**

哈希表简单的理解：在记录的存储位置和它的关键字之间建立一个确定的对应关系f，使每个关键字和结构中一个唯一的存储位置相对应。

举例：

哈希表最常见的例子是以学生学号为关键字的成绩表，１号学生的记录位置在第一条，１０号学生的记录位置在第１０条...

如果我们以学生姓名为关键字，如何建立查找表，使得根据姓名可以直接找到相应记录呢？

上面这张表即哈希表。

如果将来要查李秋梅的[成绩](http://www.hudong.com/wiki/%E6%88%90%E7%BB%A9)，可以用上述方法求出该记录所在位置：

李秋梅:lqm 12+17+13=42 取表中第42条记录即可。

问题：如果两个同学分别叫 刘丽 刘兰 该如何处理这两条记录？

这个问题是哈希表不可避免的，即冲突现象：对不同的关键字可能得到同一哈希地址。

哈希表算法-处理冲突的方法

 如果两个同学分别叫 刘丽 刘兰，当加入刘兰时，地址24发生了冲突，我们可以以某种规律使用其它的存储位置，如果选择的一个其它位置仍有冲突，则再选下一个，直到找到没有冲突的位置。选择其它位置的方法有：

１、开放定址法

Hi=(H(key)+di) MOD m i=1,2,...,k(k<=m-1)

其中m为表长，di为增量序列

如果di值可能为1,2,3,...m-1，称线性探测再散列。

如果di取值可能为1,-1,2,-2,4,-4,9,-9,16,-16,...k\*k,-k\*k(k<=m/2)

称二次探测再[散列](http://www.hudong.com/wiki/%E6%95%A3%E5%88%97" \o "散列)。

如果di取值可能为伪随机数列。称伪随机探测再散列。

比如有一组关键字{12，13，25，23，38，34，6，84，91}，Hash表长为11，Hash函数为address(key)=key%11，当插入12(hash(12)=1)，13(hash(13)=2)，25(hash(25)=3)时可以直接插入，而当插入23时，地址1被占用了，因此沿着地址1依次往下探测(探测步长可以根据情况而定,如(hash(23)+1)%11=2,(hash(23)+2)%11=3,(hash(23)+3)%11=4)，直到探测到地址4，发现为空，则将23插入其中。

２、再哈希法

当发生冲突时，使用第二个、第三个、哈希函数计算地址，直到无冲突时。缺点：计算时间增加。

３、链地址法

将所有关键字为同义词的记录存储在同一线性链表中。

虽然能够采用一些办法去减少冲突，但是冲突是无法完全避免的。因此需要根据实际情况选取解决冲突的办法。

**哈希表算法-哈希表的构造方法**

１、直接定址法

例如：有一个从1到100岁的人口数字统计表，其中，年龄作为关键字，哈希函数取关键字自身。

           但这种方法效率不高,时间复杂度是O(1),空间复杂度是O(n),n是关键字的个数

２、数字分析法

有学生的生日数据如下：

年.月.日

75.10.03  
75.11.23  
76.03.02  
76.07.12  
75.04.21  
76.02.15  
...

经分析,第一位，第二位，第三位重复的可能性大，取这三位造成冲突的机会增加，所以尽量不取前三位，取后三位比较好。

3.平方取中法

　　对关键字进行平方运算，然后取结果的中间几位作为Hash地址。假如有以下关键字序列{421，423，436}，平方之后的结果为{177241，178929，190096}，那么可以取{72，89，00}作为Hash地址。

４、折叠法

将关键字分割成位数相同的几部分（最后一部分的位数可以不同），然后取这几部分的叠加和（舍去进位）作为哈希地址，这方法称为折叠法。

例如：每一种西文图书都有一个国际标准图书编号，它是一个10位的十进制数字，若要以它作关键字建立一个哈希表，当馆藏书种类不到10,000时，可采用此法构造一个四位数的哈希函数。如果一本书的编号为0-442-20586-4,则：

5.除留取余法

　　如果知道Hash表的最大长度为m，可以取不大于m的最大质数p，然后对关键字进行取余运算，address(key)=key%p。

　　在这里p的选取非常关键，p选择的好的话，能够最大程度地减少冲突，p一般取不大于m的最大质数。

６、随机数法

选择一个随机函数，取关键字的随机函数值为它的哈希地址，即

H(key)=random(key) ,其中random为随机函数。通常用于关键字长度不等时采用此法。

**Hash表大小的确定**

　　Hash表大小的确定也非常关键，如果Hash表的空间远远大于最后实际存储的记录个数，则造成了很大的空间浪费，如果选取小了的话，则容易造成冲突。在实际情况中，一般需要根据最终记录存储个数和关键字的分布特点来确定Hash表的大小。还有一种情况时可能事先不知道最终需要存储的记录个数，则需要动态维护Hash表的容量，此时可能需要重新计算Hash地址。

Hash表的平均查找出长度

Question1:

将关键字序列（7、8、30、11、18、9、14）散列存储到散列表中。散列表的存储空间是一个下标从0开始的一维数组，散列函数为：      H(key) = (keyx3) MOD 7，处理冲突采用线性探测再散列法，要求装填（载）因子为0.7。

(1) 请画出所构造的散列表。

(2) 分别计算等概率情况下查找成功和查找不成功的平均查找长度。

Ans：

(1).首先明确一个概念装载因子，装载因子是指所有关键子填充哈希表后饱和的程度，它等于 关键字总数/哈希表的长度。 根据题意，我们可以确定哈希表的长度为 L = 7/0.7 = 10；因此此题需要构建的哈希表是下标为0~9的一维数组。根据散列函数可以得到如下散列函数值表。

H(Key) = (keyx3) MOD 7, 例如key=7时， H(7) = (7x3）%7 = 21%7=0，其他关键字同理。

 采用线性探测再散列法处理冲突，所构造的散列表为：

下面对散列表的构造方式加以说明，注意表1中的关键字7和14，30和9， 11和18，这三组关键子的H(Key)值相同，这在构建散列表时就会产生冲突，因为他们的地址相同，所以要通过一定的冲突处理方法来解决这个问题。依题，采用线性探测再散列法处理冲突。下面详细介绍如何构建散列表：

       第一个key 7，它的地址是0，因此放到散列表的数组下表为0的位置，这个位置上没有关键字，因此没有冲突可以直接填入；

       第二个key 8，它的地址是3，因此放到散列表的数组下表为3的位置，这个位置上没有关键字，因此没有冲突可以直接填入；

       第三个key 30，它的地址是6，因此放到散列表的数组下表为6的位置，这个位置上没有关键字，因此没有冲突可以直接填入；

       第四个key 11，它的地址是5，因此放到散列表的数组下表为5的位置，这个位置上没有关键字，因此没有冲突可以直接填入；

       第五个key 18，它的地址是5，因此放到散列表的数组下表为5的位置，但这个位置上已经有关键字11，遇到了冲突，此时我们根据线性探测再散列法来处理这个冲突，探测下一个位置6, 6这个位置上已经存在关键字30则继续增加步长1，因此现在的新地址应为7，位置7上没有关键字，放入即可，到此冲突已经解决；

       第六个key 9，它的地址是6，因此放到散列表的数组下表为6的位置，但这个位置上已经有关键字30，遇到了冲突，探测下一个位置7, 7这个位置上已经存在关键字18则继续增加步长1，因此现在的新地址应为8，位置8上没有关键字，放入即可；

       第七个key 14，它的地址是0，因此放到散列表的数组下表为0的位置，但这个位置上已经有关键字7，遇到了冲突，探测下一个位置1, 位置1上没有关键字，放入即可；

       到这一步所有关键字均已填入，散列表已经构造完成，如表2所示。

（2）等概率情况下查找成功平均查找长度：

        这一问可以根据第一问的构造过程求解：

        key7一次就填入了表中，因此查找次数为1，同理8， 30， 11查找次数均为1； key18 进行了3次放入操作，探测位置分别是5，6，7 ，因此查找次数为3；key9也是3次；key14 进行了两次探测，因此查找次数为2。次数表如表3所示

所以ASLsuccess= （1+1+1+1+3+3+2）/ 7 = 12/7。

        等概率情况下查找不成功的平均查找长度：

        接下来讨论不成功的情况, 看表2，计算查找不成功的次数就直接找关键字到第一个地址上关键字为空的距离即可， 但根据哈希函数地址为MOD7，因此初始只可能在0~6的位置。等概率情况下，查找0~6位置查找失败的查找次数为：

   看地址0，到第一个关键字为空的地址2的距离为3，因此查找不成功的次数为3.

        地址1， 到第一个关键为空的地址2的距离为2，因此查找不成功的次数为2.

        地址2，  到第一个关键为空的地址2的距离为1，因此查找不成功的次数为1.

        地址3，到第一个关键为空的地址4的距离为2，因此查找不成功的次数为2.

        地址4，到第一个关键为空的地址4的距离为1，因此查找不成功的次数为1.

        地址5，到第一个关键为空的地址2(注意不是地址9，因为初始只可能在0~6之间，因此循环回去)的距离为5，因此查找不成功的次数为5.

        地址6，到第一个关键为空的地址2(注意不是地址9，因为初始只可能在0~6之间，因此循环回去)的距离为4，因此查找不成功的次数为4.

        因此查找不成功的次数表如下表所示

分类: [面试](https://www.cnblogs.com/lchzls/category/984616.html)