


18 | 散列表（上）： Word文档中的单词拼写检查功能是如何实现的？

2018-10-31 王争





18 | 散列表（上）： Word文档中的单词拼写检查功能是如何实现的？

朗读人：修阳 13'40'' | 6.26M

Word 这种文本编辑器你平时应该经常用吧，那你有没有留意过它的拼写检查功能呢？一旦我们在 Word 里输入一个错误的英文单词，它就会用标红的方式提示“拼写错误”。**Word 的这个单词拼写检查功能，虽然很小但却非常实用。你有没有想过，这个功能是如何实现的呢？**

其实啊，一点儿都不难。只要你学完今天的内容，**散列表**（Hash Table）。你就能像微软 Office 的工程师一样，轻松实现这个功能。

散列思想

散列表的英文叫“Hash Table”，我们平时也叫它“哈希表”或者“Hash 表”，你一定也经常听过它，我在前面的文章里，也不止一次提到过，但是你是不是真的理解这种数据结构呢？

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。

我用一个例子来解释一下。假如我们有 89 名选手参加学校运动会。为了方便记录成绩，每个选手胸前都会贴上自己的参赛号码。这 89 名选手的编号依次是 1 到 89。现在我们希望编程实现这样一个功能，通过编号快速找到对应的选手信息。你会怎么做呢？

我们可以把这 89 名选手的信息放在数组里。编号为 1 的选手，我们放到数组中下标为 1 的位置；编号为 2 的选手，我们放到数组中下标为 2 的位置。以此类推，编号为 k 的选手放到数组中下标为 k 的位置。

因为参赛编号跟数组下标一一对应，当我们需要查询参赛编号为 x 的选手的时候，我们只需要将下标为 x 的数组元素取出来就可以了，时间复杂度就是 O(1)。这样按照编号查找选手信息，效率是不是很高？

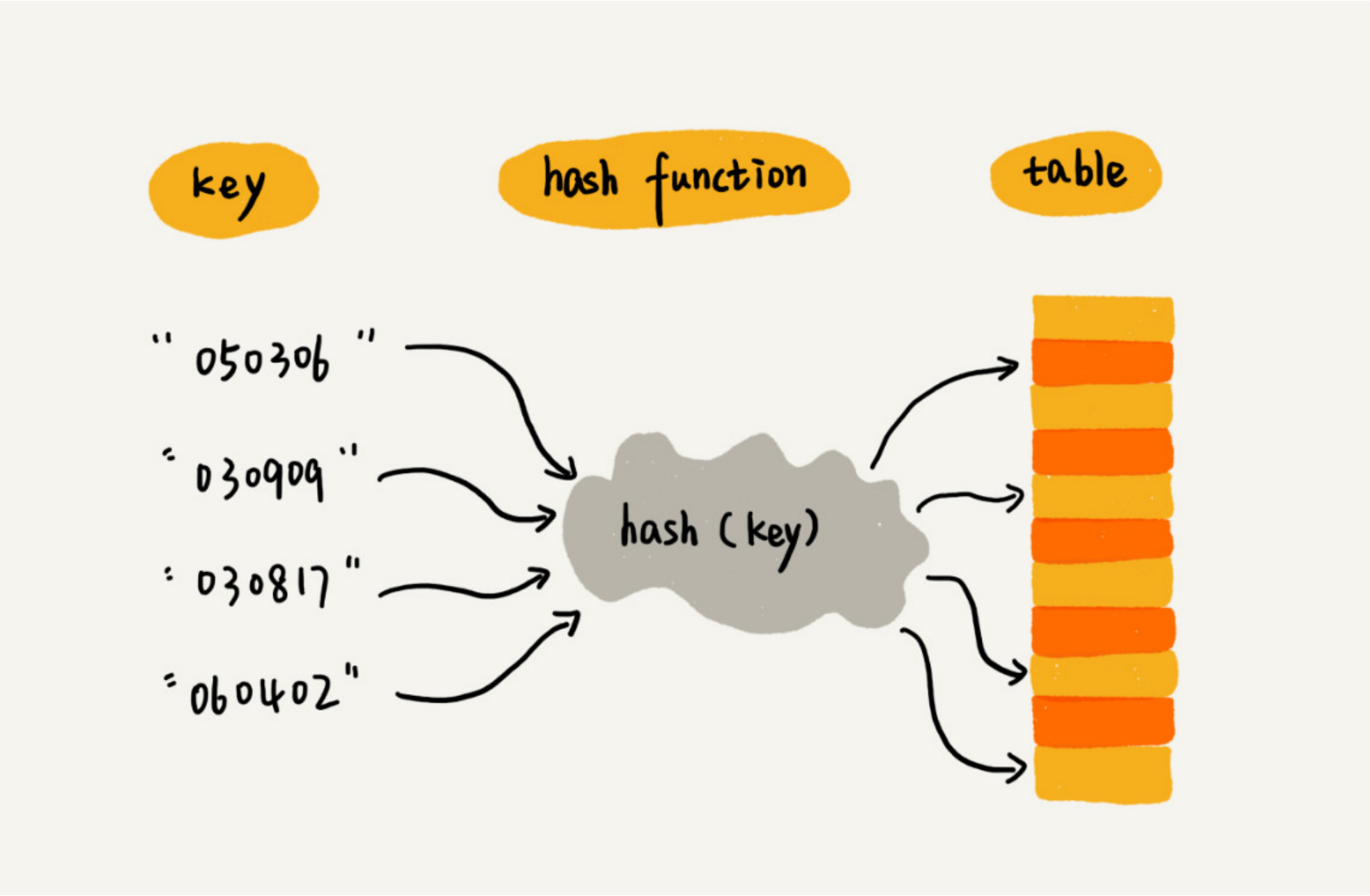
实际上，这个例子已经用到了散列的思想。在这个例子里，参赛编号是自然数，并且与数组的下标形成一一映射，所以利用数组支持根据下标随机访问的时候，时间复杂度是 O(1) 这一特性，就可以实现快速查找编号对应的选手信息。

你可能要说了，这个例子中蕴含的散列思想还不够明显，那我来改造一下这个例子。

假设校长说，参赛编号不能设置得这么简单，要加上年级、班级这些更详细的信息，所以我们把编号的规则稍微修改了一下，用 6 位数字来表示。比如 051167，其中，前两位 05 表示年级，中间两位 11 表示班级，最后两位还是原来的编号 1 到 89。这个时候我们该如何存储选手信息，才能够支持通过编号来快速查找选手信息呢？

思路还是跟前面类似。尽管我们不能直接把编号作为数组下标，但我们可以截取参赛编号的后两位作为数组下标，来存取选手信息数据。当通过参赛编号查询选手信息的时候，我们用同样的方法，取参赛编号的后两位，作为数组下标，来读取数组中的数据。

这就是典型的散列思想。其中，参赛选手的编号我们叫作**键**（key）或者**关键字**。我们用它来标识一个选手。我们把参赛编号转化为数组下标的映射方法就叫作**散列函数**（或“Hash 函数”“哈希函数”），而散列函数计算得到的值就叫作**散列值**（或“Hash 值”“哈希值”）。



通过这个例子，我们可以总结出这样的规律：散列表用的就是数组支持按照下标随机访问的时候，时间复杂度是 $O(1)$ 的特性。我们通过散列函数把元素的键值映射为下标，然后将数据存储在数组中对应下标的位置。当我们按照键值查询元素时，我们用同样的散列函数，将键值转化数组下标，从对应的数组下标的位置取数据。

散列函数

从上面的例子我们可以看到，散列函数在散列表中起着非常关键的作用。现在我们就来学习下散列函数。

散列函数，顾名思义，它是一个函数。我们可以把它定义成**hash(key)**，其中 key 表示元素的键值，hash(key) 的值表示经过散列函数计算得到的散列值。

那第一个例子中，编号就是数组下标，所以 hash(key) 就等于 key。改造后的例子，写成散列函数稍微有点复杂。我用伪代码将它写成函数就是下面这样：

```
int hash(String key) {  
    // 获取后两位字符  
    string lastTwoChars = key.substr(length-2, length);  
    // 将后两位字符转换为整数  
    int hashValue = convert lastTwoChas to int-type;  
    return hashValue;  
}
```

刚刚举的学校运动会的例子，散列函数比较简单，也比较容易想到。但是，如果参赛选手的编号是随机生成的 6 位数字，又或者用的是 a 到 z 之间的字符串，该如何构造散列函数呢？我总结了三点散列函数设计的基本要求：

- 1. 散列函数计算得到的散列值是一个非负整数；

2. 如果 $\text{key1} = \text{key2}$ ，那 $\text{hash}(\text{key1}) == \text{hash}(\text{key2})$ ；
3. 如果 $\text{key1} \neq \text{key2}$ ，那 $\text{hash}(\text{key1}) \neq \text{hash}(\text{key2})$ 。

我来解释一下这三点。其中，第一点理解起来应该没有任何问题。因为数组下标是从 0 开始的，所以散列函数生成的散列值也要是非负整数。第二点也很好理解。相同的 key，经过散列函数得到的散列值也应该是相同的。

第三点理解起来可能会有问题，我着重说一下。这个要求看起来合情合理，但是在真实的情况下，要想找到一个不同的 key 对应的散列值都不一样的散列函数，几乎是不可能的。即便像业界著名的MD5、SHA、CRC等哈希算法，也无法完全避免这种**散列冲突**。而且，因为数组的存储空间有限，也会加大散列冲突的概率。

所以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。

散列冲突

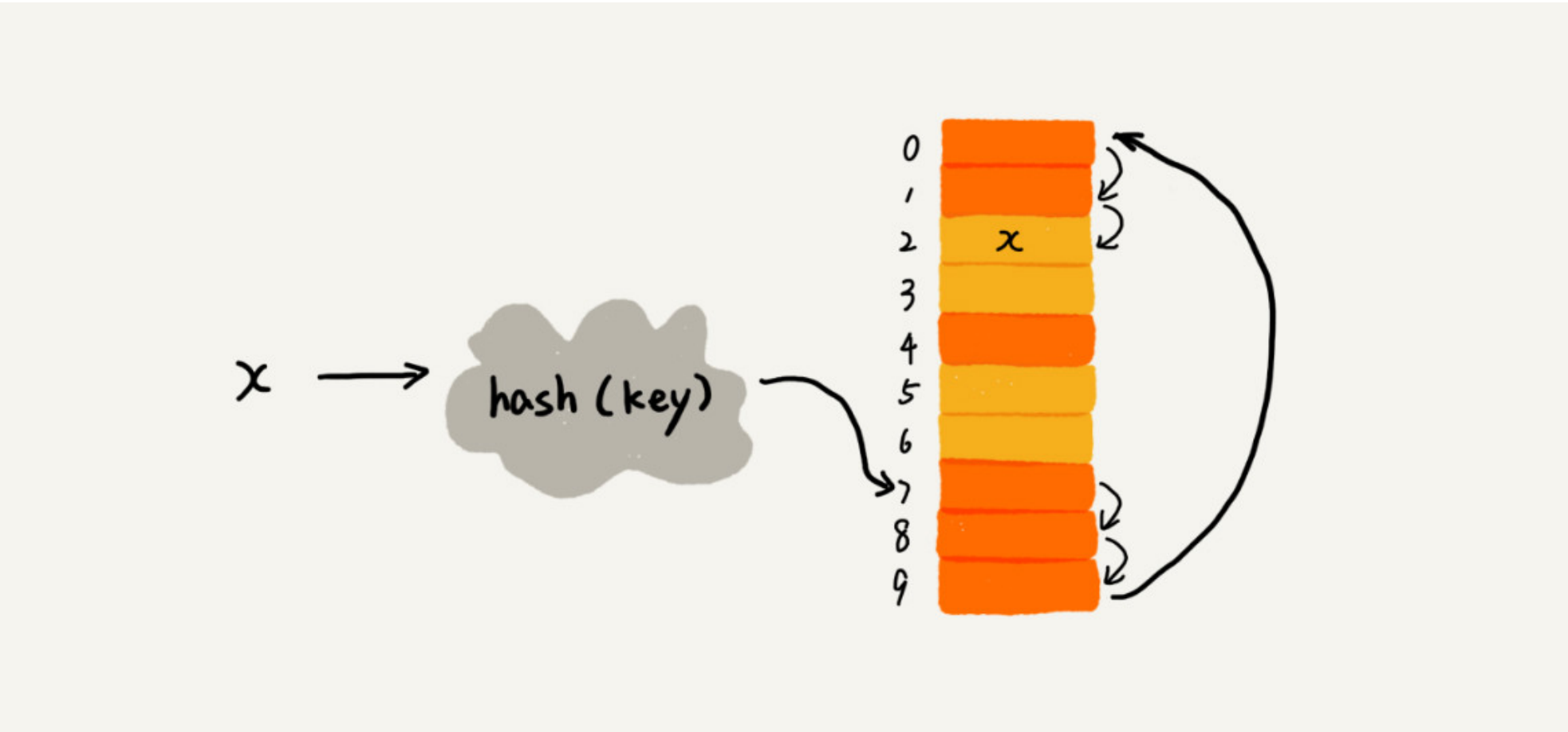
再好的散列函数也无法避免散列冲突。那究竟该如何解决散列冲突问题呢？我们常用的散列冲突解决方法有两类，开放寻址法（open addressing）和链表法（chaining）。

1. 开放寻址法

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。那如何重新探测新的位置呢？我先讲一个比较简单的探测方法，**线性探测**（Linear Probing）。

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。

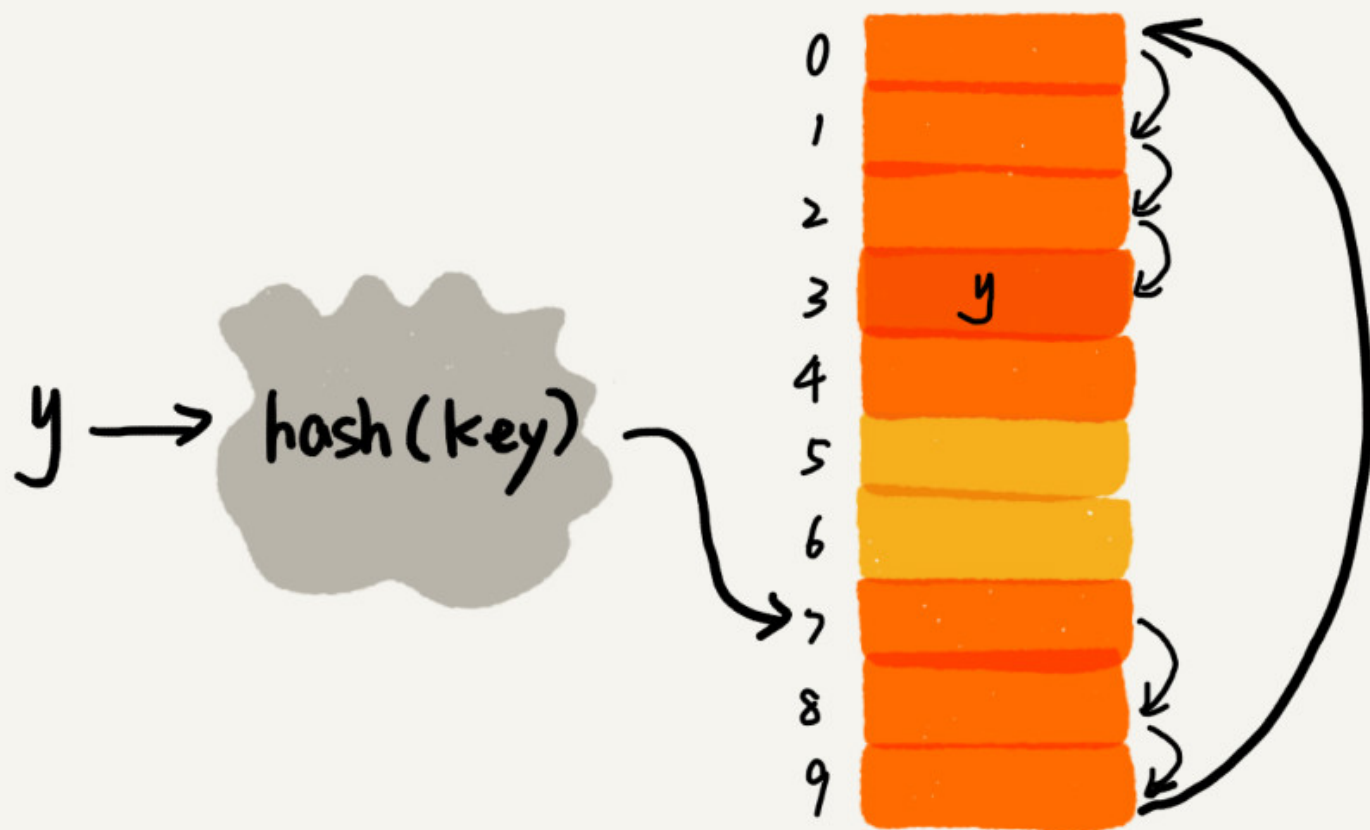
我说的可能比较抽象，我举一个例子具体给你说明一下。这里面黄色的色块表示空闲位置，橙色的色块表示已经存储了数据。



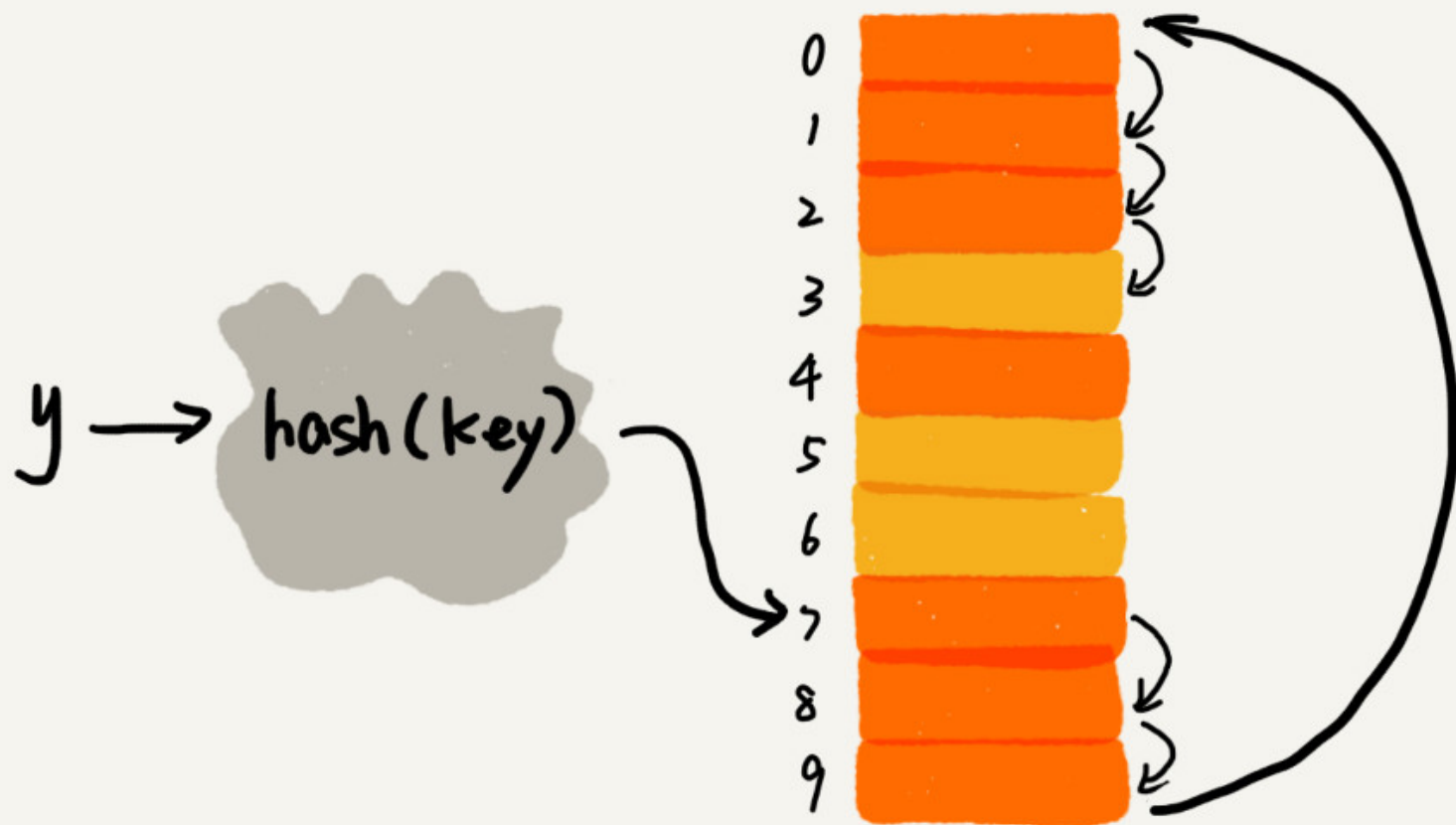
从图中可以看出，散列表的大小为 10，在元素 x 插入散列表之前，已经 6 个元素插入到散列表中。x 经过 Hash 算法之后，被散列到位置下标为 7 的位置，但是这个位置已经有数据了，所以就产生了冲突。于是我们就顺序地往后一个一个找，看有没有空闲的位置，遍历到尾部都没有找到空闲的位置，于是我们再从表头开始找，直到找到空闲位置 2，于是将其插入到这个位置。

在散列表中查找元素的过程有点儿类似插入过程。我们通过散列函数求出要查找元素的键值对应的散列值，然后比较数组中下标为散列值的元素和要查找的元素。如果相等，则说明就是我们要找的元素；否则就顺序往后依次查找。如果遍历到数组中的空闲位置，还没有找到，就说明要查找的元素并没有在散列表中。

y 在散列表中



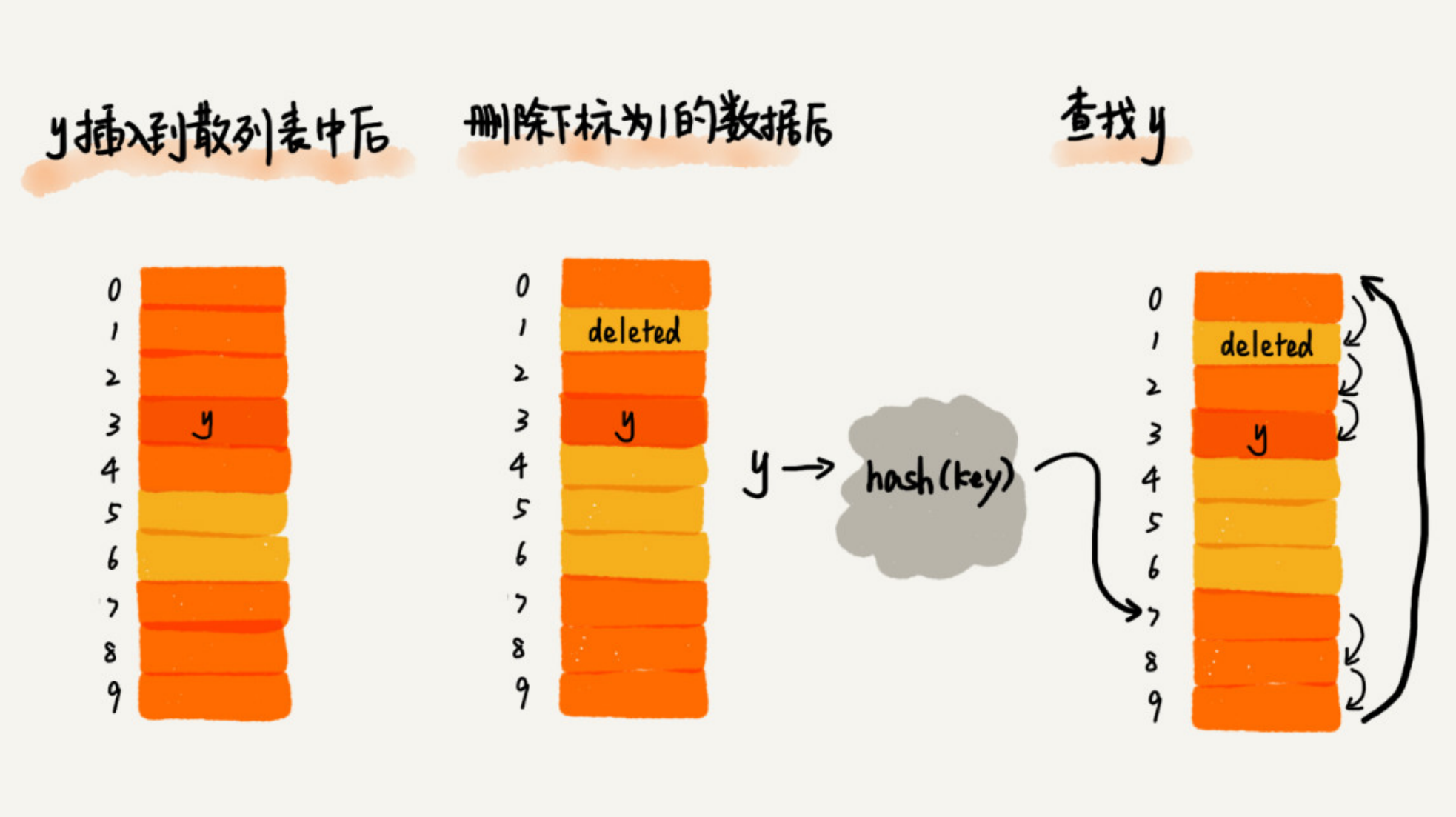
y 不在散列表中



散列表跟数组一样，不仅支持插入、查找操作，还支持删除操作。对于使用线性探测法解决冲突的散列表，删除操作稍微有些特别。我们不能单纯地把要删除的元素设置为空。这是为什么呢？

还记得我们刚讲的查找操作吗？在查找的时候，一旦我们通过线性探测方法，找到一个空闲位置，我们就可以认定散列表中不存在这个数据。但是，如果这个空闲位置是我们后来删除的，就会导致原来的查找算法失效。本来存在的数据，会被认定为不存在。这个问题如何解决呢？

我们可以将删除的元素，特殊标记为 deleted。当线性探测查找的时候，遇到标记为 deleted 的空间，并不是停下来，而是继续往下探测。



你可能已经发现了，线性探测法其实存在很大问题。当散列表中插入的数据越来越多时，散列冲突发生的可能性就会越来越大，空闲位置会越来越来少，线性探测的时间就会越来越久。极端情况下，我们可能需要探测整个散列表，所以最坏情况下的时间复杂度为 $O(n)$ 。同理，在删除和查找时，也有可能线性探测整张散列表，才能找到要查找或者删除的数据。

对于开放寻址冲突解决方法，除了线性探测方法之外，还有另外两种比较经典的探测方法，**二次探测**（Quadratic probing）和**双重散列**（Double hashing）。

所谓二次探测，跟线性探测很像，线性探测每次探测的步长是 1，那它探测的下标序列就是 $\text{hash}(\text{key})+0$, $\text{hash}(\text{key})+1$, $\text{hash}(\text{key})+2$而二次探测探测的步长就变成了原来的“二次方”，也就是说，它探测的下标序列就是 $\text{hash}(\text{key})+0$, $\text{hash}(\text{key})+1^2$, $\text{hash}(\text{key})+2^2$

所谓双重散列，意思就是不仅要使用一个散列函数。我们使用一组散列函数 $\text{hash1}(\text{key})$, $\text{hash2}(\text{key})$, $\text{hash3}(\text{key})$我们先用第一个散列函数，如果计算得到的存储位置已经被占用，再用第二个散列函数，依次类推，直到找到空闲的存储位置。

不管采用哪种探测方法，当散列表中空闲位置不多的时候，散列冲突的概率就会大大提高。为了尽可能保证散列表的操作效率，一般情况下，我们会尽可能保证散列表中有一定比例的空闲槽位。我们用**装载因子**（load factor）来表示空位的多少。

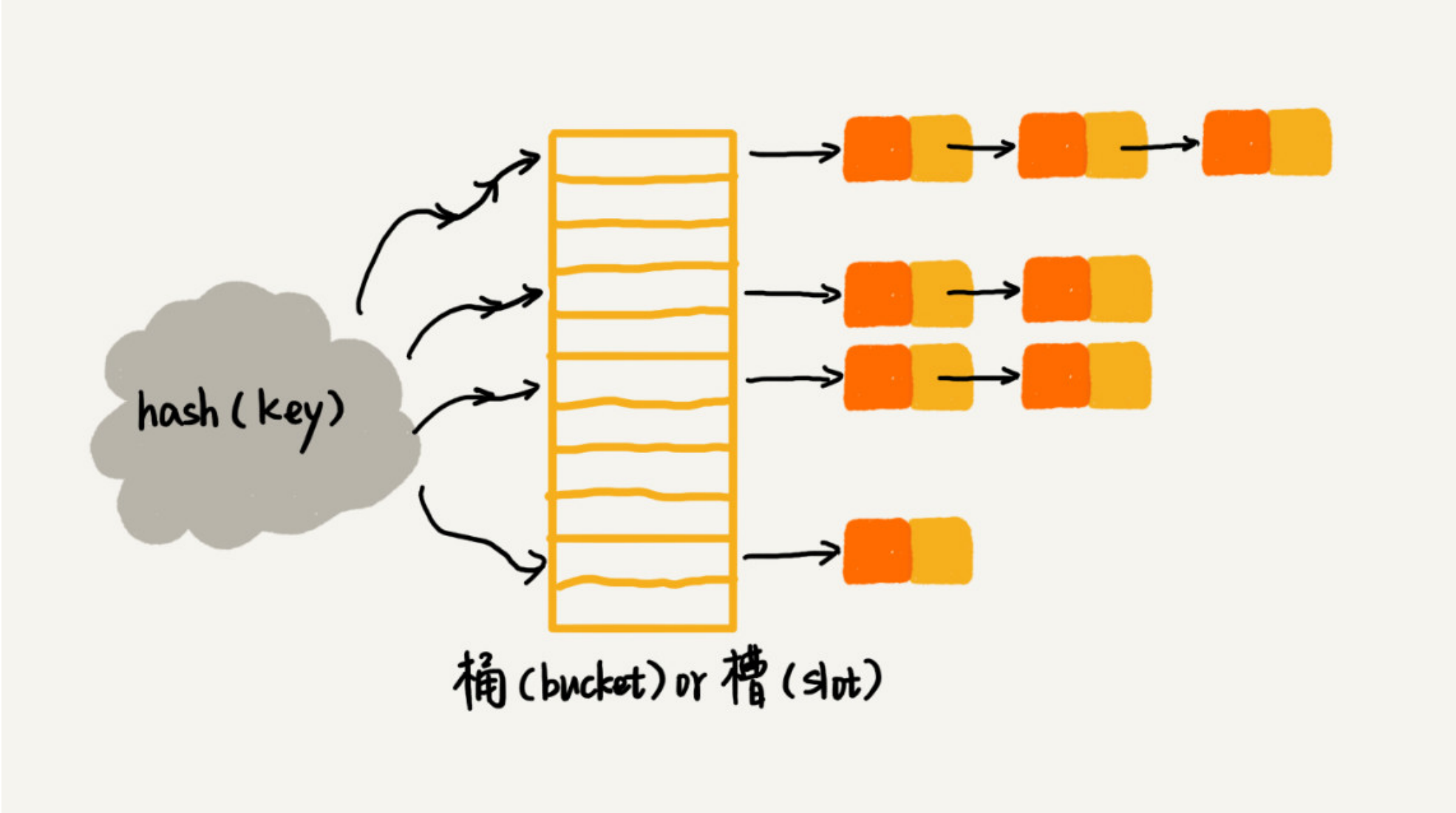
装载因子的计算公式是：

散列表的装载因子 = 填入表中的元素个数 / 散列表的长度

装载因子越大，说明空闲位置越少，冲突越多，散列表的性能会下降。

2. 链表法

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“桶（bucket）”或者“槽（slot）”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。



当插入的时候，我们只需要通过散列函数计算出对应的散列槽位，将其插入到对应链表中即可，所以插入的时间复杂度是 $O(1)$ 。当查找、删除一个元素时，我们同样通过散列函数计算出对应的槽，然后遍历链表查找或者删除。那查找或删除操作的时间复杂度是多少呢？

实际上，这两个操作的时间复杂度跟链表的长度 k 成正比，也就是 $O(k)$ 。对于散列比较均匀的散列函数来说，理论上讲， $k=n/m$ ，其中 n 表示散列中数据的个数， m 表示散列表中“槽”的个数。

解答开篇

有了前面这些基本知识储备，我们来看一下开篇的思考题：Word 文档中单词拼写检查功能是如何实现的？

常用的英文单词有 20 万个左右，假设单词的平均长度是 10 个字母，平均一个单词占用 10 个字节的内存空间，那 20 万英文单词大约占 2MB 的存储空间，就算放大 10 倍也就是 20MB。对于现在的计算机来说，这个大小完全可以放在内存里面。所以我们可以用散列表来存储整个英文单词词典。

当用户输入某个英文单词时，我们拿用户输入的单词去散列表中查找。如果查到，则说明拼写正确；如果没有查到，则说明拼写可能有误，给予提示。借助散列表这种数据结构，我们就可以轻松实现快速判断是否存在拼写错误。

内容小结

今天我讲了一些比较基础、比较偏理论的散列表知识，包括散列表的由来、散列函数、散列冲突的解决方法。

散列表来源于数组，它借助散列函数对数组这种数据结构进行扩展，利用的是数组支持按照下标随机访问元素的特性。散列表两个核心问题是**散列函数设计**和**散列冲突解决**。散列冲突有两种常用的解决方法，开放寻址法和链表法。散列函数设计的好坏决定了散列冲突的概率，也就决定散列表的性能。

针对散列函数和散列冲突，今天我只讲了一些基础的概念、方法，下一节我会更贴近实战、更加深入探讨这两个问题。

课后思考

- 1. 假设我们有 10 万条 URL 访问日志，如何按照访问次数给 URL 排序？
- 2. 有两个字符串数组，每个数组大约有 10 万条字符串，如何快速找出两个数组中相同的字符串？

欢迎留言和我分享，我会第一时间给你反馈。



版权归极客邦科技所有，未经许可不得转载

精选留言

Smallfly

👍 36

1. 假设我们有 10 万条 URL 访问日志，如何按照访问次数给 URL 排序？

遍历 10 万条数据，以 URL 为 key，访问次数为 value，存入散列表，同时记录下访问次数的最大值 K，时间复杂度 $O(N)$ 。

如果 K 不是很大，可以使用桶排序，时间复杂度 $O(N)$ 。如果 K 非常大（比如大于 10 万），就使用快速排序，复杂度 $O(N\log N)$ 。

2. 有两个字符串数组，每个数组大约有 10 万条字符串，如何快速找出两个数组中相同的字符串？

以第一个字符串数组构建散列表，key 为字符串，value 为出现次数。再遍历第二个字符串数组，以字符串为 key 在散列表中查找，如果 value 大于零，说明存在相同字符串。时间复杂度 $O(N)$ 。

2018-10-31

作者回复

👍 这条留言可以顶上去了 其他同学都看看吧

2018-11-01

五岳寻仙

👍 17

今天学习了散列表的原理，以及两种解决hash冲突的方法：开放地址法和链表法。
课后思考题第一题，我觉得可以用hash表的链表法解决。访问次数作为slot，访问次数相同的URL放入同一个slot所对应的一条链表中，这样只需要扫一遍所有的URL就排好序了，时间复杂度为 $O(n)$
第二题跟老师讲的word拼写检查有点像，我觉得可以将一个字符串数组做成hash表，然后扫描另一个字符串数组，就能找到重复的字符串。制作和扫描hash表的算法复杂度都是 $O(n)$

2018-10-31

leo

👍 6

Redis的字典是使用链式法来解决散列冲突的，并且使用了渐进式rehash的方式来进行哈希表的弹性扩容（<https://cloud.tencent.com/developer/article/1353754>，请大家斧正）。
两道思考题使用哈希表都可以解决，第二道题也可以对字符串数组进行排序后使用双指针判断，但字符串的比较成本较高，如果是整数类型更加适用。另外，哈希表比较经典的应用还有bitmap和布隆过滤器，其中布隆过滤器也可以用于文本判重，但是有一定的误判概率，可以根据场景使用。

2018-10-31

他城之途

👍 5

关于课后习题，基于某种语言的sdk实现起来可能比较容易，显然老师问的是思想，下面是我的理解，望老师和大家指正。

习题1，先分组累加次数再排序: 遍历10万数据，通过hash把相同url分组到同一个bucket下，如果bucket已存在，则取出已有次数+当前次数后再set进去，遍历完了整体再排序。

习题2，显然不是循环嵌套循环，那样时间复杂度不可接受。应该分别独立遍历两个数组，通过hash把相同的字符串扔到同一个bucket, 完了之后统计bucket长度>1的就行了。

2018-10-31



醉比

看到链表那一块感觉是hashmap的实现原理呀

2018-10-31

4



黄金的太阳

请教老师，当我在查找元素时候，在相同散列值的链表中遍历如何区分哪个是我要找的元素？毕竟查找时查询条件只包含KEY的信息吧

2018-10-31

4

作者回复

相同散列值 但是key不同的 可以再对比key

2018-10-31



万里晴空

可以写代码进行分析讲解不，这样更能感受到

2018-10-31

3



姜威

总结：

一、散列表的由来？

- 1.散列表来源于数组，它借助散列函数对数组这种数据结构进行扩展，利用的是数组支持按照下标随机访问元素的特性。
- 2.需要存储在散列表中的数据我们称为键，将键转化为数组下标的方法称为散列函数，散列函数的计算结果称为散列值。
- 3.将数据存储在散列值对应的数组下标位置。

二、如何设计散列函数？

总结3点设计散列函数的基本要求

- 1.散列函数计算得到的散列值是一个非负整数。
- 2.若key1=key2，则hash(key1)=hash(key2)
- 3.若key≠key2，则hash(key1)≠hash(key2)

正是由于第3点要求，所以产生了几乎无法避免的散列冲突问题。

三、散列冲突的解放方法？

- 1.常用的散列冲突解决方法有2类：开放寻址法（open addressing）和链表法（chaining）
- 2.开放寻址法

①核心思想：如果出现散列冲突，就重新探测一个空闲位置，将其插入。

②线性探测法（Linear Probing）：

插入数据：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之后，存储的位置已经被占用了，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。

查找数据：我们通过散列函数求出要查找元素的键值对应的散列值，然后比较数组中下标为散列值的元素和要查找的元素是否相等，若相等，则说明就是我们要查找的元素；否则，就顺序往后依次查找。如果遍历到数组的空闲位置还未找到，就说明要查找的元素并没有在散列表中。

删除数据：为了不让查找算法失效，可以将删除的元素特殊标记为deleted，当线性探测查找的时候，遇到标记为deleted的空间，并不是停下来，而是继续往下探测。

结论：最坏时间复杂度为O(n)

③二次探测（Quadratic probing）：线性探测每次探测的步长为1，即在数组中一个一个探测，而二次探测的步长变为原来的平方。

④双重散列（Double hashing）：使用一组散列函数，直到找到空闲位置为止。

⑤线性探测法的性能描述：

用“装载因子”来表示空位多少，公式：散列表装载因子=填入表中的个数/散列表的长度。

装载因子越大，说明空闲位置越少，冲突越多，散列表的性能会下降。

3.链表法（更常用）

插入数据：当插入的时候，我们需要通过散列函数计算出对应的散列槽位，将其插入到对应的链表中即可，所以插入的时间复杂度为O(1)。

查找或删除数据：当查找、删除一个元素时，通过散列函数计算对应的槽，然后遍历链表查找或删除。对于散列比较均匀的散列函数，链表的节点个数k=n/m，其中n表示散列表中数据的个数，m表示散列表中槽的个数，所以是时间复杂度为O(k)。

四、思考

1.Word文档中单词拼写检查功能是如何实现的？

字符串占用内存大小为8字节，20万单词占用内存大小不超过20MB，所以用散列表存储20万英文词典单词，然后对每个编辑进文档的单词进行查找，若未找到，则提示拼写错误。

2.假设我们有10万条URL访问日志，如何按照访问次数给URL排序？

字符串占用内存大小为8字节，10万条URL访问日志占用内存不超过10MB，通过散列表统计url访问次数，然后用TreeMap存储散列表的元素值（作为key）和数组下标值（作为value）

每个字符串数组值，每个数组值，存在10万个字符串，然后经过计算出每个数组值对应的字符串。

×

抢到的用户




ionizing

1

个人的疑问：

1. 关于开放空间的散列冲突：既然存在散列冲突问题，插入时可以通过分配新的 key 来插入存在散列冲突的元素，那么在访问时又是如何解决散列冲突的呢？比如有两个键值对 {key1: val1}, {key2: val2} 它们的 key 在生成时是冲突的，key2 经过重新分配，现在访问 {key2: val2} 时应该如何通过hash函数得到正确的 key2 呢？假如删除 {key1: val1}，现在要访问 {key2: val2}，那么执行 hash(string) 后得到的 key1 并不存在，应该怎么实现对 {key2: val2} 的正确访问呢？

2018-11-01



吴彪

1

为什么数组的存储空间有限，也会加大散列冲突的概率呢？hash函数得出来的散列值相同的概率应该是很低的，比如git hash-object，几乎不可能有碰撞，为啥在散列表里碰撞的可能性就这么大

2018-10-31

作者回复

我们还要把散列值转化为数组下标的 单纯散列值是没法直接拿来当下标的

2018-11-01



上善若水（德水）

1

word估计不太会用这种存储来检查，因为他还要给建议autosuggestion？

散列冲突后，这种存储后，查找的时候如何知道要查找的位置就一定是我要找的那个？

2018-10-31

作者回复

后面会讲trie树 支持auto suggestion

还会再比较一下的

2018-11-01




yaya

1

第一题，利用次数作为键值，冲突用链表解决，时间复杂度o(n)

第二题，将第一个字符串数组中的字符串，hash到各个区域，然后将第二个数组中的字符串利用同样的hash值处理，冲突遍历对应链表，时间复杂度o(n).

2018-10-31



拉欧

1

第二条很简单，把其中一个数组做成key为字符串的hashtable，然后第二个数组循环遍历就OK了；第一个问题，key为访问次数，value为URL，相同key的URL做成链表

2018-10-31



肖小强

0

老师，关于置顶的那个回答有些疑问。

比如第一题的解答说到“url为key，出现次数为value”

我的疑问是，hash(key)=VALUE，这个VALUE经过处理后不应该是一个随机的数组的下标吗？然后把出现次数value存入到这个位置中并不断更新。我对上面那句话的理解是hash(url)=value，所以为什么可以把出现次数作为value，value不应该是一个随机值吗？还是这个value本来就不是那个VALUE？

2018-11-04



asdf100

0

散列表的长度是固定的吗，说是类似数组，数组的内存是连续的。

2018-11-03



唐朝农民

0

Word 单词验证 是不是用 Trie 树更好，大神讲讲这个数据结构，尤其是编码这块

2018-11-02

×

抢到的用户



良辰美景

👍 0

假设我们有 10 万条 URL 访问日志，如何按照访问次数给 URL 排序？

1:按访问次数一个个往平衡二叉树里扔， 然后通过中序遍历就是一个有序的列表了

2018-11-02



BD

👍 0

关于Smallfly同学的第二题答案，如果字符串数组2在散列表里查找时，如果出现不同字符串的散列值相同的情况不就不准确了吗？我觉得应该再找到相同散列值的时候还要比较下字符串的值吧

2018-11-02



ALAN

👍 0

老师，有个问题请教下。开放寻址法查询的时候，碰到散列表为空的位置后，就不继续往后找了吗？这样设计不合理吧，因为存储的时候，存数据的散列表的位置是随机的，空的位置后面也许存了数据呢？如果是继续找的话，那为什么删除数据后，要进行特殊标记，这样标记也没意义啊，反正碰到空的位置，还是会继续找，这样标不标记都无所谓啊？

2018-11-02