欢迎大家参与本项目，贡献其他语言版本的代码，拥抱开源，让更多学习算法的小伙伴们收益！

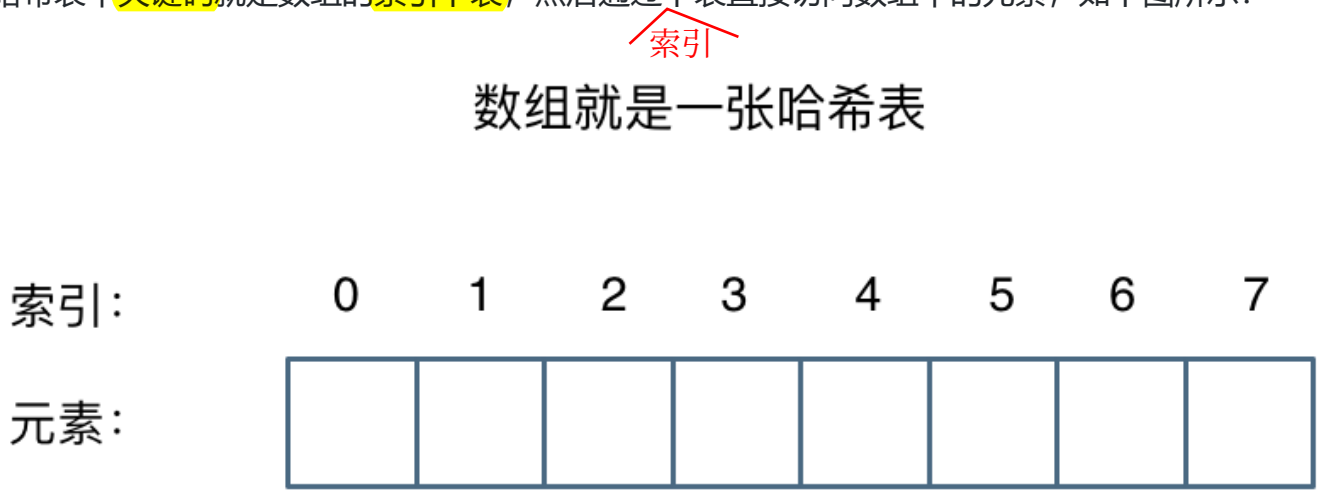
哈希表

首先什么是 哈希表，哈希表（英文名字为Hash table，国内也有一些算法书籍翻译为散列表，大家看到这两个名称知道都是指hash table就可以了）。

> 哈希表是根据关键码的值而直接进行访问的数据结构。

这么这官方的解释可能有点懵，其实直白来讲其实数组就是一张哈希表。

哈希表中关键码就是数组的索引下表，然后通过下表直接访问数组中的元素，如下图所示：



那么哈希表能解决什么问题呢，**一般哈希表都是用来快速判断一个元素是否出现集合里。**

例如要查询一个名字是否在这所学校里。

要枚举的话时间复杂度是O(n)，但如果使用哈希表的话， 只需要O(1) 就可以做到。

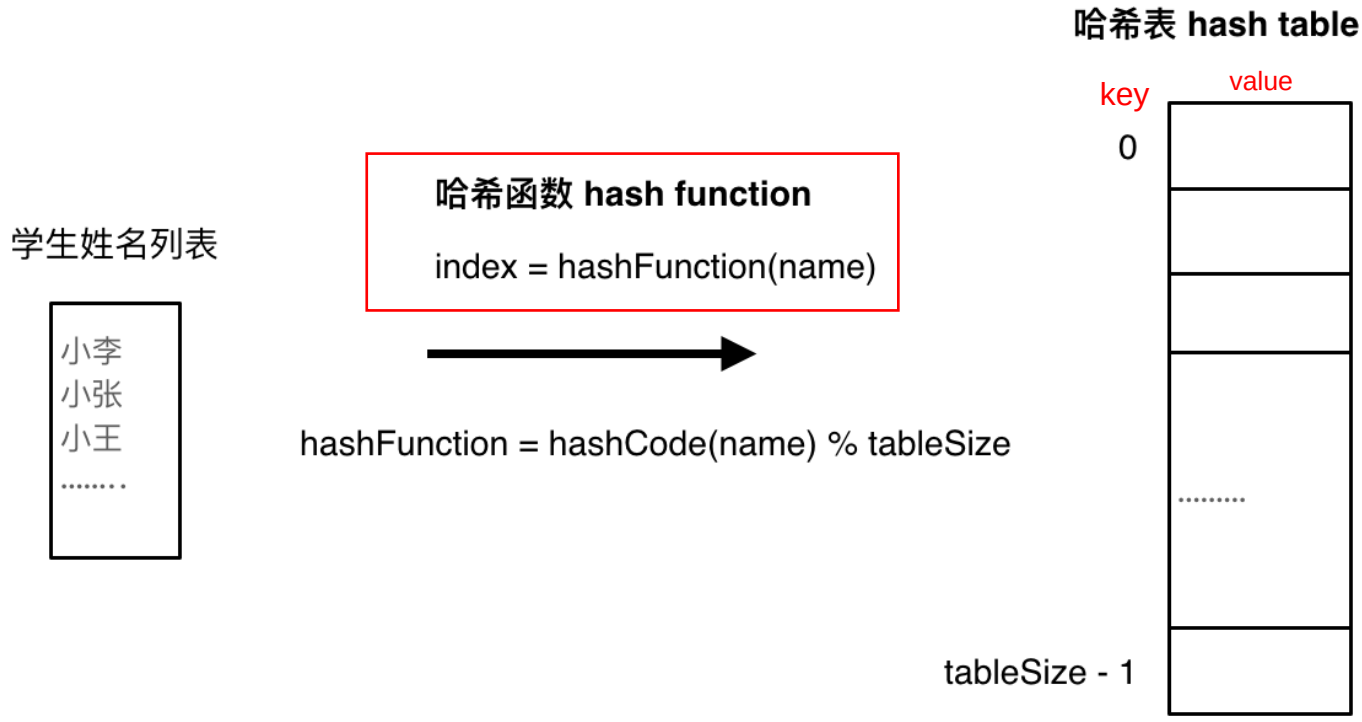
我们只需要初始化把这所学校里学生的名字都存在哈希表里，在查询的时候通过索引直接就可以知道这位同学在不在这所学校里了。

将学生姓名映射到哈希表上就涉及到了hash function ，**也就是哈希函数。**

哈希函数

哈希函数，把学生的姓名直接映射为哈希表上的索引，然后就可以通过查询索引下表快速知道这位同学是否在这所学校里了。

哈希函数如下图所示，通过hashCode把名字转化为数值，一般hashcode是通过特定编码方式，可以将其他数据格式转化为不同的数值，这样就把学生名字映射为哈希表上的索引数字了。

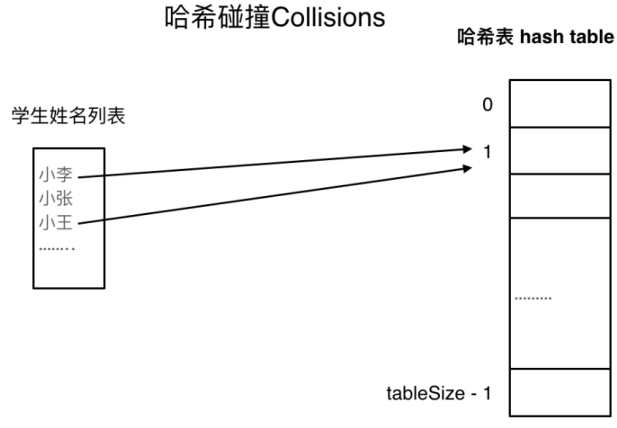


如果hashCode得到的数值大于 哈希表的大小了，也就是大于tableSize了，怎么办呢?

此时为了保证映射出来的索引数值都落在哈希表上，我们会在再次对数值做一个取模的操作，就要我们就保证了学生姓名一定可以映射到哈希表上了。

此时问题又来了，哈希表我们刚刚说过，就是一个数组。

如果学生的数量大于哈希表的大小怎么办，此时就算哈希函数计算的再均匀，也避免不了会有几位学生的名字同时映射到哈希表 同一个索引下表的位置。



接下来**哈希碰撞**登场

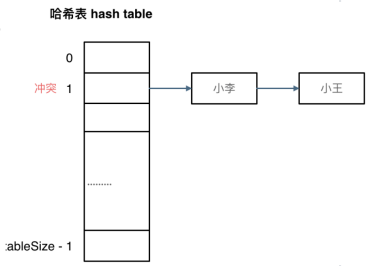
哈希碰撞

如图所示，小李和小王都映射到了索引下表 1的位置，**这**

一般哈希碰撞有两种解决方法， 拉链法和线性探测法。

拉链法

刚刚小李和小王在索引1的位置发生了冲突，发生冲突的元素都被存储在链表中。通过索引找到小李和小王了



哈希表4

（数据规模是dataSize， 哈希表的大小为tableSize）

其实拉链法就是要选择适当的哈希表的大小，这样既不会因为数组空值而浪费大量内存，也不会因为链表太长而在查找上浪费太多时间。

线性探测法

使用线性探测法，一定要保证tableSize大于dataSize。 我们需要依靠哈希表中的空位来解决碰撞问题。

例如冲突的位置，放了小李，那么就向下找一个空位放置小王的信息。所以要求tableSize一定要大于dataSize ，要不然哈希表上就没有空置的位置来存放 冲突的数据了。如图所示：

哈希表5

其实关于哈希碰撞还有非常多的细节，感兴趣的同学可以再好好研究一下，这里我就不再赘述了。

常见的三种哈希结构

当我们想使用哈希法来解决问题的时候，我们一般会选择如下三种数据结构。

- 数组 array
- set （集合）
- map(映射)

这里数组就没啥可说的了，我们来看一下set。

在C++中，set 和 map 分别提供以下三种数据结构，其底层实现以及优劣如下表所示：

集合	底层实现	是否有序	数值是否可以重复	能否更改数值	查询效率	增删效率
std::set	红黑树	有序	否	否	O(logn)	O(logn)
std::multiset	红黑树	有序	是	否	O(logn)	O(logn)
std::unordered_set	哈希表	无序	否	否	O(1)	O(1)

std::unordered_set底层实现为哈希表，std::set 和std::multiset 的底层实现是红黑树，红黑树是一种平衡二叉搜索树，所以key值是有序的，但key不可以修改，改动key值会导致整棵树的错乱，所以只能删除和增加。

映射	底层实现	是否有序	数值是否可以重复	能否更改数值	查询效率	增删效率
std::map	红黑树	key有序	key不可重复	key不可修改	O(logn)	O(logn)
std::multimap	红黑树	key有序	key可重复	key不可修改	O(logn)	O(logn)
std::unordered_map	哈希表	key无序	key不可重复	key不可修改	O(1)	O(1)

std::unordered_map 底层实现为哈希表，std::map 和std::multimap 的底层实现是红黑树。同理，std::map 和std::multimap 的key也是有序的（这个问题也经常作为面试题，考察对语言容器底层的理解）。

当我们要使用集合来解决哈希问题的时候，优先使用unordered_set，因为它的查询和增删效率是最优的，如果需要集合是有序的，那么就用set，如果要求不仅有序还要有重复数据的话，那么就用multiset。

那么再来看一下map ，在map 是一个key value 的数据结构，map中，对key是有限制，对value没有限制的，因为key的存储方式使用红黑树实现的。 ????查看eg

其他语言例如：java里的HashMap ， TreeMap 都是一样的原理。可以灵活贯通。

虽然std::set、std::multiset 的底层实现是红黑树，不是哈希表，但是std::set、std::multiset 依然使用哈希函数来做映射，只不过底层的符号表使用了红黑树来存储数据，所以使用这些数据结构来解决映射问题的方法，我们依然称之为哈希法。 map也是一样的道理。

这里在说一下，一些C++的经典书籍上 例如STL源码剖析，说到了~~hash_set hash_map~~，这个与unordered_set，unordered_map又有什么关系呢？

实际上功能都是一样一样的， 但是unordered_set在C++11的时候被引入标准库了，而~~hash_set并没有~~，所以建议还是使用unordered_set比较好，这就好比一个是官方认证的，~~hash_set，hash_map 是C++11标准之前民间高手自发造的轮子~~。

C++标准库

- unordered_set
- unordered_map
- multiset
- multimap

民间高手造的轮子

- hash_set
- hash_map
- hash_multiset
- hash_multimap

总结

总结一下，**当我们遇到了要快速判断一个元素是否出现集合里的时候，就要考虑哈希法。**

但是哈希法也是**牺牲了空间换取了时间**，因为我们要使用额外的数组，set或者是map来存放数据，才能实现快速的查找。

如果在做面试题目的时候遇到需要判断一个元素是否出现过的场景也应该第一时间想到哈希法!

- 作者微信：程序员Carl
- B站视频：代码随想录
- 知识星球：代码随想录