

master ▾

...

leetcode-master / problems / 哈希表总结.md



youngyangyang04 Update

History

1 contributor

134 lines (72 sloc) 7.52 KB

[PDF下载](#) [代码随想录](#) [刷题](#) [微信群](#) [B站](#) [代码随想录](#) [知识星球](#) [代码随想录](#)

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！

哈希表总结篇如约而至

## 哈希表理论基础

在[关于哈希表，你该了解这些！](#)中，我们介绍了哈希表的基础理论知识，不同于枯燥的讲解，这里介绍了都是对刷题有帮助的理论知识点。

一般来说哈希表都是用来快速判断一个元素是否出现集合里。

对于哈希表，要知道[哈希函数](#)和[哈希碰撞](#)在哈希表中的作用。

哈希函数是把传入的key映射到符号表的索引上。

哈希碰撞处理有[多个key映射到相同索引上](#)时的情景，处理碰撞的普遍方式是拉链法和线性探测法。

接下来是常见的三种哈希结构：

- 数组
- set（集合）
- map（映射）

在C++语言中，set 和 map 都分别提供了三种数据结构，每种数据结构的底层实现和用途都有所不同，在[关于哈希表，你该了解这些！](#)中我给出了详细分析，[这一知识点很重要！](#)

例如什么时候用std::set, 什么时候用std::multiset, 什么时候用std::unordered\_set, 都是很有考究的。

只有对这些数据结构的底层实现很熟悉, 才能灵活使用, 否则很容易写出效率低下的程序。

# 哈希表经典题目

## 数组作为哈希表

一些应用场景就是为数组量身定做的。

在242.有效的字母异位词中, 我们提到了数组就是简单的哈希表, 但是数组的大小是受限的!

这道题目包含小写字母, 那么使用数组来做哈希最合适不过。

在383.赎金信中同样要求只有小写字母, 那么就给我们浓浓的暗示, 用数组!

本题和242.有效的字母异位词很像, 242.有效的字母异位词是求 字符串a 和 字符串b 是否可以相互组成, 在383.赎金信中是求字符串a能否组成字符串b, 而不用管字符串b 能不能组成字符串a。

一些同学可能想, 用数组干啥, 都用map不就完事了。

上面两道题目用map确实可以, 但使用map的空间消耗要比数组大一些, 因为map要维护红黑树或者符号表, 而且还要做哈希函数的运算。所以数组更加简单直接有效!

## set作为哈希表

在349. 两个数组的交集中我们给出了什么时候用数组就不行了, 需要用set。

这道题目没有限制数值的大小, 就无法使用数组来做哈希表了。

主要因为如下两点:

- 数组的大小是有限的, 受到系统栈空间 (不是数据结构的栈) 的限制。
- 如果数组空间够大, 但哈希值比较少、特别分散、跨度非常大, 使用数组就造成空间的极大浪费。

所以此时一样的做映射的话, 就可以使用set了。

关于set, C++ 给提供了如下三种可用的数据结构: (详情请看[关于哈希表, 你该了解这些!](#))

- std::set
- std::multiset
- std::unordered\_set

std::set和std::multiset底层实现都是红黑树, std::unordered\_set的底层实现是哈希, 使用unordered\_set 读写效率是最高的, 本题并不需要对数据进行排序, 而且还不要让数据重复, 所以选择unordered\_set。

在[202.快乐数](#)中，我们再次使用了unordered\_set来判断一个数是否重复出现过。

## map作为哈希表

在[1.两数之和](#)中map正式登场。

来说一说：使用数组和set来做哈希法的局限。

- 数组的大小是受限制的，而且如果元素很少，而哈希值太大会造成内存空间的浪费。
- set是一个集合，里面放的元素只能是一个key，而两数之和这道题目，不仅要判断y是否存在而且还要记录y的下表位置，因为要返回x和y的下表。所以set也不能用。

map是一种 `<key, value>` 的结构，本题可以用key保存数值，用value在保存数值所在的下表。所以使用map最为合适。

C++提供如下三种map：：（详情请看[关于哈希表，你该了解这些！](#)）

- `std::map`
- `std::multimap`
- `std::unordered_map`

`std::unordered_map` 底层实现为哈希，`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），[1.两数之和](#)中并不需要key有序，选择`std::unordered_map` 效率更高！

在[454.四数相加](#)中我们提到了其实需要哈希的地方都能找到map的身影。

本题咋眼一看好像和[18. 四数之和](#)，[15.三数之和](#)差不多，其实差很多！

**关键差别是本题为四个独立的数组，只要找到 $A[i] + B[j] + C[k] + D[l] = 0$ 就可以，不用考虑重复问题，而[18. 四数之和](#)，[15.三数之和](#)是一个数组（集合）里找到和为0的组合，可就难很多了！**

用哈希法解决了两数之和，很多同学会感觉用哈希法也可以解决三数之和，四数之和。

其实是可以解决，但是非常麻烦，需要去重导致代码效率很低。

在[15.三数之和](#)中我给出了哈希法和双指针两个解法，大家就可以体会到，使用哈希法还是比较麻烦的。

所以[18. 四数之和](#)，[15.三数之和](#)都推荐使用双指针法！

## 总结

对于哈希表的知识相信很多同学都知道，但是没有成体系。

本篇我们从哈希表的理论基础到数组、set和map的经典应用，把哈希表的整个全貌完整的呈现给大家。

同时也强调虽然map是万能的，详细介绍了什么时候用数组，什么时候用set。

相信通过这个总结篇，大家可以对哈希表有一个全面的了解。

---

- 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](#)