

LeetCode Review

C++ function API参考网站，查STL API和库函数：<https://en.cppreference.com/w/>

国际版：<https://leetcode.com/>；中文版：<https://leetcode-cn.com/>

做题方法论：

1. 思考。想不明白的时候在纸上写写画画，帮助思考。特别是图论的题目，拿最简单的test case画出来就能帮助思考，效果拔群。
2. 手机倒计时。时间：10~20分钟。20分钟足够把解题相关的知识和技巧拿出来枚举一遍了。想不到方法多半因为有的知识你还不知道。快去看答案吧。

做题顺序：

第一遍，按照题号做。easy >> medium >> hard，题目编号前300题。

easy：熟悉cpp语法以及STL的用法

medium：重温基础的数据结构和算法，重点总结各种常见数据结构和算法

hard：去discuss上学习大神的解法思路，自己写一遍提交。其他看discuss的情况：自己写的代码很繁杂，逻辑不清晰，自己理不顺；题目tag里面提示有别的解法的时候；想了半个小时无思路的时候。

第二遍，按照tag做。

按照题号做完一遍之后，按照tag刷第二遍，总结同类题型的套路。发现共性，总结套路和模版，然后才能斩瓜切菜。目标效果：

- 思路和代码要变得简洁。大部分题目代码量控制在50行以内。
- bug free 快速写出常用代码块(union find, dfs/bfs 几个变种, binary search, partition)。熟练掌握的模版和套路，就可以bug free解决medium的题目。
- 掌握各种数据结构的时空复杂度，实现的难易程度。选用更加简单的数据结构，运行时间更快。时间复杂度相同时，可以选更简单的数据结构，化繁为简。用vector，不用 unordered_map

评价：见多识广也是能力，熟练套路不等于生搬硬套。快速识别出题目套路并且解决已经是非常优秀，活用知识，甚至发明算法则需要更深入的理解。

LeetCode的OJ也可以试一试。

LeetCode Habits

- 按照Tag刷，同时用不同颜色标记做过和没有做过的题目
 - 红蓝绿：表示有做过，难度已被标记
 - 无色：还没有做过
- 做题速度
 - 一般做的出来不限制时间
 - 做不出来，限制10-15min后看答案
 - 看懂至少一个有效答案，记忆思路方法，然后不要看答案，按照自己的记忆和思路重新写一遍
 - 如果不止brutal force 解法，了解多个Ans来补充知识面
 - 总结code
 - 不同解法的代码 + 代码注释放在code.c, 当时为什么要这样做，答案为什么这样写
 - 解题思路放在md上 (?)

Q1 - Add two sums

First submission/brutal force:

Success Details >

Runtime: **412 ms**, faster than **32.98%** of C++ online submissions for Two Sum.

Memory Usage: **9.3 MB**, less than **13.08%** of C++ online submissions for Two Sum.

Next challenges:

3Sum

4Sum

Two Sum II - Input array is sorted

Two Sum III - Data structure design

Subarray Sum Equals K

Two Sum IV - Input is a BST

Two Sum Less Than K

Show off your acceptance:



Time Submitted	Status	Runtime	Memory	Language
11/02/2020 21:22	Accepted	412 ms	9.3 MB	cpp
08/13/2020 11:14	Accepted	16 ms	10.3 MB	cpp
08/12/2020 22:48	Accepted	12 ms	10.5 MB	cpp

- 知识点

1. `std::vector` 的 public member function 有哪些，该怎么用？

Capacity

<code>empty</code>	checks whether the container is empty (public member function)
<code>size</code>	returns the number of elements (public member function)
<code>max_size</code>	returns the maximum possible number of elements (public member function)
<code>reserve</code>	reserves storage (public member function)
<code>capacity</code>	returns the number of elements that can be held in currently allocated storage (public member function)
<code>shrink_to_fit</code> (C++11)	reduces memory usage by freeing unused memory (public member function)

Modifiers

<code>clear</code>	clears the contents (public member function)
<code>insert</code>	inserts elements (public member function)
<code>emplace</code> (C++11)	constructs element in-place (public member function)
<code>erase</code>	erases elements (public member function)
<code>push_back</code>	adds an element to the end (public member function)
<code>emplace_back</code> (C++11)	constructs an element in-place at the end (public member function)
<code>pop_back</code>	removes the last element (public member function)
<code>resize</code>	changes the number of elements stored (public member function)
<code>swap</code>	swaps the contents (public member function)

size: 目前已经加进去的元素的数量

capacity: `vector` 的大小是动态分配的, 因此大小延展到一定程度就会自动扩展占据的内存大小。就像一个地产项目不断增加楼盘之前, 首先要开辟出更大的建筑面积。capacity就是所谓的“更大的建筑面积”, 即这个object所占据的内存。

clear: AOE, 把 `vector` 里的东西像垃圾一样全部丢掉。

erase: 定点清除, 把你不想要的数字; 不想要的索引范围; 通通干掉

push_back: 把新的元素放在 `vector` 的最后一个位置

insert: 定点插入到vector的开头、末尾、或者中间某个位置; 可以定点插入另一个array!

resize: 改变 `vector` 的size。如果缩短了, 砍掉多余的元素; 如果变长了, 增加default数字

swap: Exchanges the contents of the container with those of other。就vector1 和 vector2内容互换

2. `std::unordered_map/hash_map` 怎么用?

Success Details >

Runtime: 12 ms, faster than 97.06% of C++ online submissions for Two Sum.

Memory Usage: 10.3 MB, less than 24.13% of C++ online submissions for Two Sum.

- 知识点

`unordered_map` 的成员函数该怎么用? `unordered_map` 是通过 `hash_table` 来实现的。

Capacity

<code>empty</code>	checks whether the container is empty (public member function)
<code>size</code>	returns the number of elements (public member function)
<code>max_size</code>	returns the maximum possible number of elements (public member function)

Modifiers

<code>clear</code>	clears the contents (public member function)
<code>insert</code>	inserts elements or nodes (since C++17) (public member function)
<code>insert_or_assign</code> (C++17)	inserts an element or assigns to the current element if the key already exists (public member function)
<code>emplace</code>	constructs element in-place (public member function)
<code>emplace_hint</code>	constructs elements in-place using a hint (public member function)
<code>try_emplace</code> (C++17)	inserts in-place if the key does not exist, does nothing if the key exists (public member function)
<code>erase</code>	erases elements (public member function)
<code>swap</code>	swaps the contents (public member function)
<code>extract</code> (C++17)	extracts nodes from the container (public member function)
<code>merge</code> (C++17)	splices nodes from another container (public member function)

Lookup

<code>at</code>	access specified element with bounds checking (public member function)
<code>operator[]</code>	access or insert specified element (public member function)
<code>count</code>	returns the number of elements matching specific key (public member function)
<code>find</code>	finds element with specific key (public member function)
<code>contains</code> (C++20)	checks if the container contains element with specific key (public member function)
<code>equal_range</code>	returns range of elements matching a specific key (public member function)

- Insert的2种用法:

```
container.insert(pair<int, int>(nums[i], i));  
//container.insert({nums[i], i});
```

具体的还有更多种用法。

- `map.find(key)` 返回的是什么值?

iterator。一般用map.begin() 和 map.end() 去接这个函数。如果想要测试map.find(key)是否存在，看它是否等于 map.end()。map.end() 这个iterator处于最后一个元素的后一位。

一般如果要取key对应的value，有两种方法：

```
map<int,int>::iterator iter; iter = container.find(difference); int a = iter->second;
int a = container.find(difference)->second;
```

就可以顺利取出value的值了。

总结

- 第一种解法：暴力搜索。挑出每个元素*i*，依次遍历剩下的元素以寻找配对。已经被pick过的元素*i*，在接下来其余的搜索中不会再出现。时间复杂度： $O(n^2)$ ，空间复杂度 $O(1)$ 。算法稳定、常用、直接，但是不够聪明。
- 第二种解法：遍历2次哈希表。把元素从vector 中一个个放进哈希表中，先全部放进去。放进去以后，利用哈希表 $O(1)$ 的查找来寻找差值。全部放进 Hash_Table 需要 $O(n)$ ， n 次哈希表查找需要 $O(n)$ 。具体实现是利用 unordered_map：它是用 hashtable 实现的。map 的数据结构为 <key,value>，因此把元素的(index, element) 以 (value, key)放进去，即以element为key，我们的value是index(题目要求返回2个数的index)。利用 unordered_map 近似于 $O(1)$ 的查找，实现 $O(n)$ 。总和： $O(n)$ ， $O(n) = O(2n) = O(n)$
- 第三种解法：遍历一次哈希表。放一个元素进 hash 的时候，同时查找表，如果已经发现互补，直接return。如果没有发现互补，继续放元素。放元素需要 $O(n)$ ，查找 n 次 $O(1)$ 需要 $O(n)$ ，共需要 $O(2n)$ 次。第三种解法跟第二种时间空间复杂度差不多，只是第三种写法更为简洁一些。

Q：如果不用 hash_map，vector 的元素放到另一个 vector 里，可以达到时空复杂度的降低吗？

A：不可以，放入第*n*个元素需要 $O(1)$ ，同时需要*n-1*次顺序查找。放入： $O(n)$ ；查找： $1 + 2 + .. + n-1 = O(n^2)$ 。跟暴力搜索没有差别。