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！

1. 两数之和

https://leetcode-cn.com/problems/two-sum/

给定一个整数数组 nums 和一个目标值 target，请你在该数组中找出和为目标值的那两个 整数，并返回他们的数组下标。

你可以假设每种输入只会对应一个答案。但是，数组中同一个元素不能使用两遍。

找到就可以直接输出，结束

示例:

给定 nums = [2, 7, 11, 15], target = 9

因为 nums[0] + nums[1] = 2 + 7 = 9

所以返回 [0, 1]

思路

很明显暴力的解法是两层for循环查找，时间复杂度是O(n^2)。

建议大家做这道题目之前，先做一下这两道

- 242. 有效的字母异位词
- 349. 两个数组的交集

242. 有效的字母异位词 这道题目是用数组作为哈希表来解决哈希问题，349. 两个数组的交集这道题目是通过set作为哈希表来解决哈希问题。

本题呢，则要使用map，那么来看一下使用数组和set来做哈希法的局限。

哈希表中的关键码的值？？？

- 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。???

此时就要选择另一种数据结构：map ，map是一种key value的存储结构，可以用key保存数值，用value在保存数值所在的下表。

C++中map，有三种类型：

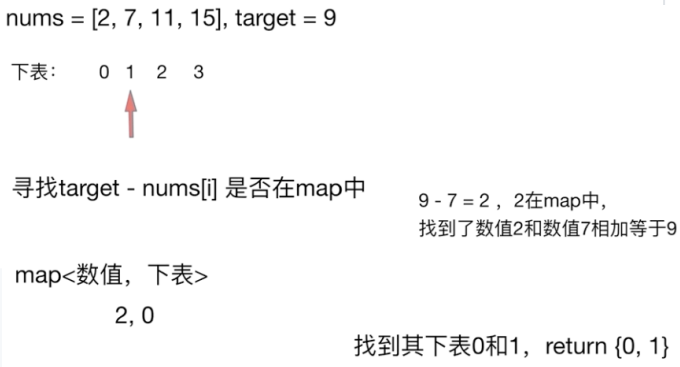
映射	底层实现	是否有序	数值是否可以重复	能否更改数值	查询效率	增删效率
std::map	红黑树	key有序	key不可重复	key不可修改	O(logn)	O(logn)
std::multimap	红黑树	key有序	key可重复	key不可修改	O(logn)	O(logn)
std::unordered_map	哈希表	key无序	key不可重复	key不可修改	O(1)	O(1)

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）。 更多哈希表的理论知识请看关于哈希表，你该了解这些！。

这道题目中并不需要key有序，选择std::unordered_map 效率更高！

解题思路动画如下：



C++代码：

<key, value>

```
class Solution {
public:
    vector<int> twoSum(vector<int>& nums, int target) {
        std::unordered_map <int,int> map;
        for(int i = 0; i < nums.size(); i++) {
            auto iter = map.find(target - nums[i]);
            if(iter != map.end()) {
                return {iter->second, i};
            }
            map.insert(pair<int, int>(nums[i], i));
        }
        return {};
    }
};
```

这一步里面map没有初始化，是空的

其他语言版本

Java：

```java
public int[] twoSum(int[] nums, int target) {
    int[] res = new int[2];
    if(nums == null || nums.length == 0){
        return res;
    }
    Map<Integer, Integer> map = new HashMap<>();
    for(int i = 0; i < nums.length; i++){
        int temp = target - nums[i];
        if(map.containsKey(temp)){
            res[1] = i;
            res[0] = map.get(temp);
        }
        map.put(nums[i], i);
    }
    return res;
}
```

Python：

```python
class Solution:
    def twoSum(self, nums: List[int], target: int) -> List[int]:
        hashmap={}
        for ind,num in enumerate(nums):
            hashmap[num] = ind
        for i,num in enumerate(nums):
            j = hashmap.get(target - num)
            if j is not None and i!=j:
                return [i,j]
```

Go：

```go
func twoSum(nums []int, target int) []int {
    for k1, _ := range nums {
        for k2 := k1 + 1; k2 < len(nums); k2++ {
            if target == nums[k1] + nums[k2] {
                return []int{k1, k2}
            }
        }
    }
    return []int{}
}
```

```go
// 使用map方式解题，降低时间复杂度
func twoSum(nums []int, target int) []int {
    m := make(map[int]int)
    for index, val := range nums {
        if preIndex, ok := m[target-val]; ok {
```

```
            return []int{preIndex, index}
        } else {
            m[val] = index
        }
    }
    return []int{}
}
```

Rust

```
use std::collections::HashMap;

impl Solution {
    pub fn two_sum(nums: Vec<i32>, target: i32) -> Vec<i32> {
        let mut map = HashMap::with_capacity(nums.len());

        for i in 0..nums.len() {
            if let Some(k) = map.get(&(target - nums[i])) {
                if *k != i {
                    return vec![*k as i32,  i as i32];
                }
            }
            map.insert(nums[i], i);
        }
        panic!("not found")
    }
}
```

Javascript

```
var twoSum = function (nums, target) {
  let hash = {};
  for (let i = 0; i < nums.length; i++) {
    if (hash[target - nums[i]] !== undefined) {
      return [i, hash[target - nums[i]]];
    }
    hash[nums[i]] = i;
  }
  return [];
};
```

- 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