链滴

# daily leetcode - Two-Sum - !

作者：lonuslan
原文链接：https://ld246.com/article/1579593302515
来源网站：链滴

# 题目地址

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

# 题目描述

Given an array of integers, return indices of the two numbers such that they add up to a specif c target.

You may assume that each input would have *exactly* one solution, and you may not use th *same* element twice.

Example:
Given nums = [2, 7, 11, 15], target = 9,

Because nums[0] + nums[1] = 2 + 7 = 9,
return [0, 1].

# 思路

啦啦啦，欢迎开启 LeetCode 刷题的旅程，这将是一段漫长而又艰辛的旅程，这是一条攀登珠穆朗玛皑皑雪山路，这是通向 One Piece 宝藏的伟大航路，这是成为火影的无比残酷的修罗场，这是打破丝与高富帅之间界限的崩玉。但请不要害怕，在老船长 Grandyang 博主的带领下，必将一路披荆斩，将各位带到成功的彼岸，不过一定要牢记的是，不要下船，不要中途放弃，要坚持，要自我修炼，断成长！那么，起航吧 ~这道 Two Sum 的题目作为 LeetCode 的开篇之题，乃是经典中的经典，所谓 ‘**平生不识 TwoSum，刷尽 LeetCode 也枉然**’ ，就像英语单词书的第一个单词总是 Abando 一样，很多没有毅力坚持的人就只能记住这一个单词，所以通常情况下单词书就前几页有翻动的痕迹后面都是崭新如初，道理不需多讲，鸡汤不必多灌，明白的人自然明白。

这道题给了我们一个数组，还有一个目标数 target，让找到两个数字，使其和为 target，乍一看就感可以用暴力搜索，但是猜到 OJ 肯定不会允许用暴力搜索这么简单的方法，于是去试了一下，果然是 T me Limit Exceeded，这个算法的时间复杂度是 O(n^2)。那么只能想个 O(n) 的算法来实现，由于力搜索的方法是遍历所有的两个数字的组合，然后算其和，这样虽然节省了空间，但是时间复杂度高一般来说，为了提高时间的复杂度，需要用空间来换，这算是一个 trade off 吧，但这里只想用线性时间复杂度来解决问题，就是说只能遍历一个数字，那么另一个数字呢，可以事先将其存储起来，使一个 HashMap，来建立数字和其坐标位置之间的映射，由于 HashMap 是常数级的查找效率，这样遍历数组的时候，用 target 减去遍历到的数字，就是另一个需要的数字了，直接在 HashMap 中查其是否存在即可，注意要判断查找到的数字不是第一个数字，比如 target 是 4，遍历到了一个 2，那另外一个 2 不能是之前那个 2，整个实现步骤为：先遍历一遍数组，建立 HashMap 映射，然后再遍一遍，开始查找，找到则记录 index。

# 关键点解析

同思路

# 代码

C++ 解法一：

```
class Solution {
public:
    vector<int> twoSum(vector<int>& nums, int target) {
        unordered_map<int, int> m;
        vector<int> res;
        for (int i = 0; i < nums.size(); ++i) {
            m[nums[i]] = i;
        }
        for (int i = 0; i < nums.size(); ++i) {
            int t = target - nums[i];
            if (m.count(t) && m[t] != i) {
                res.push_back(i);
                res.push_back(m[t]);
                break;
            }
        }
        return res;
    }
};
```

Java 解法一：

```
public class Solution {
    public int[] twoSum(int[] nums, int target) {
        HashMap<Integer, Integer> m = new HashMap<Integer, Integer>();
        int[] res = new int[2];
        for (int i = 0; i < nums.length; ++i) {
            m.put(nums[i], i);
        }
        for (int i = 0; i < nums.length; ++i) {
            int t = target - nums[i];
            if (m.containsKey(t) && m.get(t) != i) {
                res[0] = i;
```

```
            res[1] = m.get(t);
            break;
        }
    }
    return res;
  }
}
```

或者可以写的更加简洁一些，把两个 for 循环合并成一个：

C++ 解法二：

```
class Solution {
public:
  vector<int> twoSum(vector<int>& nums, int target) {
    unordered_map<int, int> m;
    for (int i = 0; i < nums.size(); ++i) {
      if (m.count(target - nums[i])) {
        return {i, m[target - nums[i]]};
      }
      m[nums[i]] = i;
    }
    return {};
  }
};
```

Java 解法二：

```
public class Solution {
  public int[] twoSum(int[] nums, int target) {
    HashMap<Integer, Integer> m = new HashMap<Integer, Integer>();
    int[] res = new int[2];
    for (int i = 0; i < nums.length; ++i) {
      if (m.containsKey(target - nums[i])) {
        res[0] = i;
        res[1] = m.get(target - nums[i]);
        break;
      }
      m.put(nums[i], i);
    }
    return res;
  }
}
```

本文参考自：

https://github.com/grandyang/leetcode/issues/1