04 两数之和

更新时间：2019-08-08 09:40:54

> 谁和我一样用功，谁就会和我一样成功。
>
> ——莫扎特

刷题内容

> **难度: Easy**
>
> 题目链接：https://leetcode-cn.com/problems/two-sum/。

题目描述

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，并返回它们的数组下标。

你可以假设每种输入只会对应一个答案。但是，你不能重复利用这个数组中同样的元素。

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]

题目详解

这道题目给我们一个数组，数组里面全是整数，然后再给我们一个数字 target，需要我们求出在这个数组中哪两个数字的和正好是 target。注意，你不能重复利用这个数组中同样的元素，指的是每一个位置上的数字只能被使用一次，比如数组 [3,3,5]，你可以使用第一个和第二个 3，但是你不能使用第一个 3 两次。

解题方案

思路1：时间复杂度：O(N^2) 空间复杂度：O(1)

暴力解法，双重循环遍历：

外层循环从数组中取出下标为 `i` 的元素 `num[i]`，内层循环取出 `i` 之后的元素 `nums[j]` 一一与下标为 `i` 的元素进行相加操作，判断结果是否为 `target`。

- 为什么内层循环要取`i`之后的元素的元素呢？因为如果第二轮取得`i`之前的数的话，其实我们之前就已经考虑过这种情况了（即外层循环已经遍历过此时内层循环的这个数字）。

题目只要求找到一种，所以一旦找到直接返回。时间复杂度中的 `N` 代表的是 `nums` 列表的长度。

下面我们来看代码：

Python beats 27.6%

```python
class Solution(object):
    def twoSum(self, nums, target):
        """
        :type nums: List[int]
        :type target: int
        :rtype: List[int]
        """
        # 第一轮遍历
        for i in range(len(nums)):
            # 第二轮遍历不能重复计算了
            for j in range(i+1, len(nums)):
                if nums[i] + nums[j] == target:
                    # 注意 leetcode 中要求返回的是索引位置
                    return [i, j]
```

Java beats 36.03%

```java
class Solution {
    public int[] twoSum(int[] nums, int target) {
        // 第一轮遍历
        for (int i = 0; i < nums.length; i++) {
            // 第二轮遍历不能重复计算了
            for (int j = i + 1; j < nums.length; j++) {
                if (nums[i] + nums[j] == target) {
                    // 注意 leetcode 中要求返回的是索引位置
                    return new int[]{i, j};
                }
            }
        }
        return null;
    }
}
```

C++ beats 35.36%

```
class Solution {
public:
    vector<int> twoSum(vector<int>& nums, int target) {
        int n = nums.size();
        // 第一轮遍历
        for (int i = 0;i < n;i++) {
            // 第二轮遍历不能重复计算了
            for (int j = i + 1;j < n;j++) {
                if (nums[i] + nums[j] == target) {
                    // 注意 leetcode 中要求返回的是索引位置
                    vector<int> ret(2);
                    ret[0] = i;
                    ret[1] = j;
                    return ret;
                }
            }
        }
        return vector<int>(0);
    }
};
```

Go

```
func twoSum(nums []int, target int) []int {
    // 第一轮遍历
    for i := 0; i < len(nums); i++ {
        // 第二轮遍历不能重复计算了
        for j := i+1; j < len(nums); j++ {
            if nums[i] + nums[j] == target {
                // 注意 leetcode 中要求返回的是索引位置
                return []int{i, j}
            }
        }
    }
    return []int{}
}
```

可以看到上述代码的beats(代码优劣程度)明显较低，这是因为我们的时间复杂度是O(N^2)的。

思路2：时间复杂度：O(N) 空间复杂度：O(N)

上面的思路1时间复杂度太高了，是典型的加快时间的方法，这样做是不可取的。其实我们可以牺牲空间来换取时间。

我们希望，在我们顺序遍历取得一个数 `num1` 的时候，就知道和它配对的数 `target-num1` 是否在我们的 `nums` 里面，并且不单单只存在一个。比如说`target`为`4`， `nums` 为`[2,3]`，假设我们此时取得的`num1`为`2`，那么和它配对的`2`确实在`nums`中，但是数字`2`在`nums`中只出现了一次，我们无法取得两次，所以也是不行的。

因此我们有了下面的步骤：

1. 建立字典 `lookup` 存放第一个数字，并存放该数字的 `index`；
2. 判断 `lookup` 中是否存在 `target - 当前数字cur`， 则当前值`cur`和某个`lookup`中的`key`值相加之和为 `target`；
3. 如果存在，则返回： `target - 当前数字cur` 的 `index` 与 当前值 `cur` 的 `index`；
4. 如果不存在则将当前数字 `cur` 为key，当前数字的 `index` 为值存入 `lookup`。

下面我们来看下代码：

Python beats 100%

```python
class Solution(object):
    def twoSum(self, nums, target):
        """
        :type nums: List[int]
        :type target: int
        :rtype: List[int]
        """
        lookup = {}
        for i, num in enumerate(nums):
            if target - num in lookup:
                return [lookup[target-num], i]
            else: # 每一轮都存下当前num和其index到map中
                lookup[num] = i
```

就像之前提到的特殊情况一样，这里注意我们要边遍历边将 `num: idx` 放入 `lookup` 中，而不是在做遍历操作之前就将所有内容放入 `lookup` 中。例如数组 [3,5]，target = 6，如果我们一次性全部放进了 `lookup` 里面，那么当我们遍历到3的时候，我们会发现 target - num = 6-3 = 3 在 `lookup` 中，但其实我们只有一个 3 在数组中，而我们使用了两次，误以为我们有2个3。

```python
class Solution:
    def twoSum(self, nums, target):
        """
        :type nums: List[int]
        :type target: int
        :rtype: List[int]
        """
        lookup = {}
        for i, num in enumerate(nums):
            lookup[num] = i
        for i, num in enumerate(nums):
            if target - num in lookup:
                return [i, lookup[target-num]]
```

```
Wrong Answer

Input
[3,2,4]
6

Output
[0,0]

Expected
[1,2]
```

Java beats 99.39%

```java
class Solution {
    public int[] twoSum(int[] nums, int target) {
        HashMap<Integer, Integer> map = new HashMap<>();
        for (int i = 0; i < nums.length; i++) {
            // 将原本为两个目标值切换为一个目标值，只需要每次从 map 中寻找目标值即可
            int num = target - nums[i];
            if (map.containsKey(num)) {
                return new int[]{map.get(num), i};
            }
            // 每次遍历过的值都存储到 map 中，这样之后就能从 map 中寻找需要的目标值
            map.put(nums[i], i);
        }
        return null;
    }
}
```

C++ beat 86.53%

```cpp
class Solution {
public:
    vector<int> twoSum(vector<int>& nums, int target) {
        unordered_map<int, int> indexMap;
        unordered_map<int, int>::iterator it;
        for (int i = 0;i < nums.size();i++) {
            it = indexMap.find(target - nums[i]);
            if (it != indexMap.end()) {
                vector<int> ret(2);
                ret[0] = it->second;
                ret[1] = i;
                return ret;
            } else { // 每一轮都存下当前num和其index到map中
                indexMap[nums[i]] = i;
            }
        }
        return vector<int>(0);
    }
};
```

Go

```go
func twoSum(nums []int, target int) []int {
    lookup := map[int]int{}
    for j, num := range nums {
        if i, ok := lookup[target - num]; ok {
            return []int{i, j}
        }
        lookup[num] = j // 每一轮都存下当前num和其index到map中
    }
    return []int{}
}
```

这里的代码快多了，说明我们的空间换时间成功了。

总结

- 拿到题目之后我们可以用最朴素和最暴力的方式来解答。这样做虽然没有什么问题，但我希望你在用最简单的方式去解答完成之后，多想想怎么可以去优化一下你的解答方式，培养你的“最优”思维。习惯这样想之后你的代码才会越来越优秀；

- 当然。不是任何时候“最优”即最合适，我们也要考虑一下条件限制的问题，在各种环境中找出最合适的方式才是真正的“最优”；
- 常见的减小时间复杂度的方式有：用空间来弥补多余的计算。

学习更多

- 数组应用场景
- 散列表原理

}

03 时间复杂度与空间复杂度分析

05 整数反转