1. Two Sum

March 5, 2016 | 2.5M views

Previous Next

Average Rating: 4.78 (2338 votes)

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

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

Example:

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

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

Solution

Approach 1: Brute Force

The brute force approach is simple. Loop through each element x and find if there is another value that equals to $target - x$.

Java

```
public int[] twoSum(int[] nums, int target) {
    for (int i = 0; i < nums.length; i++) {
        for (int j = i + 1; j < nums.length; j++) {
            if (nums[j] == target - nums[i]) {
                return new int[] { i, j };
            }
        }
    }
    throw new IllegalArgumentException("No two sum solution");
}
```

Complexity Analysis

- Time complexity : $O(n^2)$. For each element, we try to find its complement by looping through the rest of array which takes $O(n)$ time. Therefore, the time complexity is $O(n^2)$.
- Space complexity : $O(1)$.

Approach 2: Two-pass Hash Table

To improve our run time complexity, we need a more efficient way to check if the complement exists in the array. If the complement exists, we need to look up its index. What is the best way to maintain a mapping of each element in the array to its index? A hash table.

We reduce the look up time from $O(n)$ to $O(1)$ by trading space for speed. A hash table is built exactly for this purpose, it supports fast look up in *near* constant time. I say "near" because if a collision occurred, a look up could degenerate to $O(n)$ time. But look up in hash table should be amortized $O(1)$ time as long as the hash function was chosen carefully.

A simple implementation uses two iterations. In the first iteration, we add each element's value and its index to the table. Then, in the second iteration we check if each element's complement ($target - nums[i]$) exists in the table. Beware that the complement must not be $nums[i]$ itself!

Java

```
public int[] twoSum(int[] nums, int target) {
    Map<Integer, Integer> map = new HashMap<>();
    for (int i = 0; i < nums.length; i++) {
        map.put(nums[i], i);
    }
    for (int i = 0; i < nums.length; i++) {
        int complement = target - nums[i];
        if (map.containsKey(complement) && map.get(complement) != i) {
            return new int[] { i, map.get(complement) };
        }
    }
    throw new IllegalArgumentException("No two sum solution");
}
```

Complexity Analysis:

- Time complexity : $O(n)$. We traverse the list containing n elements exactly twice. Since the hash table reduces the look up time to $O(1)$, the time complexity is $O(n)$.
- Space complexity : $O(n)$. The extra space required depends on the number of items stored in the hash table, which stores exactly n elements.

Approach 3: One-pass Hash Table

It turns out we can do it in one-pass. While we iterate and inserting elements into the table, we also look back to check if current element's complement already exists in the table. If it exists, we have found a solution and return immediately.

Java

```
public int[] twoSum(int[] nums, int target) {
    Map<Integer, Integer> map = new HashMap<>();
    for (int i = 0; i < nums.length; i++) {
        int complement = target - nums[i];
        if (map.containsKey(complement)) {
            return new int[] { map.get(complement), i };
        }
        map.put(nums[i], i);
    }
    throw new IllegalArgumentException("No two sum solution");
}
```

Complexity Analysis:

- Time complexity : $O(n)$. We traverse the list containing n elements only once. Each look up in the table costs only $O(1)$ time.
- Space complexity : $O(n)$. The extra space required depends on the number of items stored in the hash table, which stores at most n elements.

Rate this article:

Previous Next

Comments: 1303

Sort By

Type comment here... (Markdown is supported)

Preview Post

HankLiu5 ★ 758 November 7, 2018 7:07 AM

python3

```
class Solution:
    def twoSum(self, nums, target):
        """
```

Read More

754 Share Reply

SHOW 50 REPLIES

adarshnaidu2511 ★ 836 October 20, 2018 4:29 PM

How to view the solution in c

832 Share Reply

SHOW 28 REPLIES

ketong ★ 108 February 28, 2019 9:53 AM

Python one-pass has table - faster than 100%

1. Two sum

Read More

108 Share Reply

SHOW 3 REPLIES

jakesully ★ 329 November 22, 2018 12:19 PM

[JAVASCRIPT] Took me several attempts...But this code says:
"Runtime: 52 ms, faster than 100.00% of JavaScript online submissions for Two Sum."

```
const twoSum = function(nums, target) {
    const seen = {};
```

Read More

328 Share Reply

SHOW 34 REPLIES

panahi ★ 298 April 27, 2019 10:40 PM

Clean Python One-Pass solution:

```
def twoSum(self, nums, target):
    seen = {}
    for i, n in enumerate(nums):
```

Read More

213 Share Reply

SHOW 19 REPLIES

shihabl8983 ★ 91 October 11, 2018 10:49 PM

What would the fastest way be in python?

90 Share Reply

SHOW 11 REPLIES

A_xian_ ★ 180 October 10, 2018 5:45 PM

It takes 208ms to run my code.Could it faster?
Here is my code:

```
public:
    vector<int> twoSum(vector<int>& nums, int target) {
```

Read More

180 Share Reply

SHOW 5 REPLIES

kvnyang ★ 140 September 9, 2018 4:34 PM

Fastest C solution in O(n) time

```
/**
C solution in O(n) time, using open addressing hash table.
 */
```

Read More

109 Share Reply

SHOW 13 REPLIES

pradeepvrd ★ 94 September 4, 2018 11:50 PM

My Python3 solution:

```
class Solution:
    def twoSum(self, nums, target):
        """
```

Read More

94 Share Reply

SHOW 4 REPLIES

igorescobar ★ 74 March 2, 2019 10:12 PM

In Javascript:

```
// O(n) - One-pass Hash Table
var twoSum = function(nums, target) {
    let map = {};
```

Read More

72 Share Reply

SHOW 3 REPLIES