1. Two Sum

Description (/problems/two-sum/description/) Hints (/problems/two-sum/hints/) Submissions (/problems/two-sum/submissions/)

Quick Navigation View in Article (/articles/two-sum/)

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 Copy

```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.

Notes

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

Copy

```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

Copy

```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");
}
```

Notes

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.

Comments: 581

Sort By

Type comment here... (Markdown is supported)

Preview Post

Em_Qi (/em_qi) ★ 0 7 hours ago

My Python3 solution:

```python
class Solution:
    def twoSum(self, nums, target):
        for i,num in enumerate(nums):
```

Read More

0 Share Reply

Quentin_YANG (/quentin_yang) ★ 0 7 hours ago

C++ solution:

```
class Solution {
public:
```

Read More

0 Share Reply

scalaian (/scalaian) 0 9 hours ago

scala solution:

```
    def twoSum(nums: Array[Int], target: Int): Array[Int] =
      nums.zipWithIndex.
        filter(x =>
```

Read More

0 Share Reply

nikunj.gupta150 (/nikunjgupta150) 0 a day ago

If a map is built like (nums[i], i) won't repeating values of nums[i] replace the previous values as keys must be unique? Why not make map like (i, num[i]) ?

0 Share Reply

SHOW 1 REPLY

AnushreeAnkola (/anushreeankola) 1 2 days ago

My Python solution is :

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

Read More

1 Share Reply

sathwikmatsa (/sathwikmatsa) 7 2 days ago

Python 3 solution 0(nlogn) time complexity:

Read More

6 Share Reply

hassonhigh (/hassonhigh) 4 August 31, 2018 8:27 AM

c++ solution, hash_table is not invalid ,another vector could i use?

Notes

4 Share Reply

SHOW 1 REPLY

singhdi1 (/singhdi1) 4 August 31, 2018 6:07 AM

JavaScript solution, one pass with just a for loop

```
var twoSum = function(nums, target) {
for (var i=0; i<nums.length; i++) {
    if (nums.slice(i+1,).indexOf(target - nums[i]) > -1){
```

Read More

Notes

4 Share Reply

SHOW 1 REPLY

ErXiao (/erxiao) 1 August 29, 2018 4:54 PM

if i want use 'two-pass hash table', how to be elegant?
below is my solution, please hint me, thanks:
class Solution:
def twoSum(self, nums, target):
"""

Read More

1 Share Reply

coder419 (/coder419) 1 August 29, 2018 4:01 AM

C Hash solution
int solve(int *arr, int N, int target, int *index1, int *index2)
{
int MAX_NUM = 100;
*int harr = (int)calloc(1, MAX_NUM*sizeof(int));

Read More

1 Share Reply

1 2 3 4 5 6 ... 58 59