# 1. Two Sum

March 5, 2016 hash-table (/articles/?tag=hash-table)

Question | Editorial Solu

## Question

> 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.
>
> **Example:**
>
> ```
> Given nums = [2, 7, 11, 15], target = 9,
>
> Because nums[0] + nums[1] = 2 + 7 = 9,
> return [0, 1].
> ```
>
> **UPDATE (2016/2/13):**
> The return format had been changed to **zero-based** indices. Please read the above updated description carefully.

**Quick Navigation**



## Solution

### Approach #1 (Brute Force) [Accepted]

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

```
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, th time complexity is $O(n^2)$.
- Space complexity : $O(1)$.

## Approach #2 (Two-pass Hash Table) [Accepted]

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 loc 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!

```
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) [Accepted]

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.

```
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.

Join the conversation

Login to R

**jettwu1130** commented 30 minutes ago

@StefanPochmann (https://discuss.leetcode.com/uid/591) Thanks!
iscuss.leetcode.com/user/jettwu1130)

**StefanPochmann** commented 33 minutes ago

@jettwu1130 (https://discuss.leetcode.com/uid/76108) You saved a constant almost nothing and you **added** nums.length subtractions.
iscuss.leetcode.com/user/stefanpochmann)
You quite possibly made it **slower**.

**jettwu1130** commented 1 hour ago

In Approach #1, why should we go through all the element in the outer loop.
iscuss.leetcode.com/user/jettwu1130)
How about save one outer loop: for (int i = 0; i < nums.length -1; i++)

**dnunn** commented last week

@yanhotim56 (https://discuss.leetcode.com/uid/74023) you are not initializing the pos array correctly. You can change it to something lik
iscuss.leetcode.com/user/dnunn)
int[] pos = new int[2];
This way, the complier knows to expect the 2 values that you will be passing in.
Alternatively, you can also initialize pos inside of your conditional as follows:
int[] pos = {i, j};

**charliecha** commented last week

// I can not return two values in my scheme so I changed to use a void method.
iscuss.leetcode.com/user/charliecha)
public class TwoSum_Map {

```
public static void TwoSum(int target)
{
        int []numbers ={2,7,11,13,15};   //  I  defined []numbers here instead of adding [ ]parameters in the TwoSum method
    Map <Integer,Integer> map =new HashMap<>();
        for(int i=0;i<numbers.length;i++)
        {
                map.put(numbers[i], i);
        }

         for(int i=0;i<numbers.length;i++)
         {
                 int componet = target-numbers[i];
                 if(map.containsKey(componet)&&map.get(componet)!=i)
                 {
                         System.out.println(i+" and "+map.get(componet));
                          break;    //     break needs to be here to prevent from the same but position-swapped answer
                 }

         }

}

public static void main(String []args)
{
            TwoSum(9);
}
```

}

**ManuelP** commented last week

I suspect that if we had statistics about solutions wrongly accused of being wrong, these would win...
iscuss.leetcode.com/user/manuelp)

**yarving** commented 2 weeks ago

There's a bug in `Approach #2 (Two-pass Hash Table) [Accepted]` for this condition:
iscuss.leetcode.com/user/yarving)
nums = [3, 3, 4, 4], target=6
The map's key 3 only count once.

**yanhotim56** commented 2 weeks ago

iscuss.leetcode.com/user/yanhotim56)

public class Solution {
public int[] twoSum(int[] nums, int target) {
int length = nums.length;
int[] pos ;
for (int i = 0 ; i < length;i++){
for (int j = 0 ; j < length ; j++){
if (nums[i]+nums[j] == target){
pos [0] = i;
pos [1] = j;
i = length;
j = length;
}
}
}
return pos;
}

My code is as the above and I dont know why it gives me error saying pos[] not declared.

**ehorse** commented 2 weeks ago

iscuss.leetcode.com/user/ehorse)

@adamburns (https://discuss.leetcode.com/uid/71201)
I have working c# code. Don't use Dictionary.Add() since it will throw if there is duplicate key. Use Dictionary[key] = value;

public static int[] Leet_001_TwoSum_v1(int[] nums, int target)
{
Dictionary<int, int> ht = new Dictionary<int, int>();
for(int i= 0; i < nums.Length; i++)
{
// DONOT use ht.Add(). Otherwise it will throw if there is duplicate key
ht[nums[i]] = i;
}

```
        for(int i=0; i < nums.Length; i++)
        {
            int complement = target - nums[i];
            if(ht.ContainsKey(complement) && ht[complement] != i)
            {
                return new int[] { i, ht[complement] };
            }
        }

        throw new ArgumentException();
    }

    // One pass hash table
    public static int[] Leet_001_TwoSum_v2(int[] nums, int target)
    {
        Dictionary<int, int> ht = new Dictionary<int, int>();
        for (int i = 0; i < nums.Length; i++)
        {
            int complement = target - nums[i];
            if (ht.ContainsKey(complement))
            {
                return new int[] { i, ht[complement] };
            }
            ht[nums[i]] = i;
        }

        throw new ArgumentException();
    }
```

**nagarjuna_chikoti** commented 2 weeks ago

iscuss.leetcode.com/user/nagarjuna_chikoti)

The below error is resolved. Hence, pls ignore below question. Sorry for the trouble guys.

View original thread (https://discuss.leetcode.com/topic/29)

Load more commer