1. Two Sum

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

Question | Editorial Solution

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, 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].
```

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, the 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 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!

```
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.

sapocaly commented 2 days ago

@mr.bin (https://discuss.leetcode.com/uid/148689) thats because the question is changed since the book published. the onlyonly thing need to change is the index that returned start from 0 now
liscuss.leetcode.com/user/sapocaly)

Mr.Bin commented 2 days ago

The very first code in your eBook is not Accepted. I reget to buy this eBook and do not recommend anyone to buy it.
liscuss.leetcode.com/user/mr-

```
public int[] twoSum(int[] numbers, int target) {
Map<Integer, Integer> map = new HashMap<>();
for (int i = 0; i < numbers.length; i++) {
int x = numbers[i];
if (map.containsKey(target - x)) {
return new int[] { map.get(target - x) + 1, i + 1 };
}
map.put(x, i);
}
throw new IllegalArgumentException("No two sum solution");
}
```

sapocaly commented 5 days ago

@ziyang1103 (https://discuss.leetcode.com/uid/161342) @dieson (https://discuss.leetcode.com/uid/160763) @kevin.huang.x3 (https://discuss.leetcode.com/uid/160288) First, any hashmap can only have one value map to one key while you can have multiple keys map to one value. But here, it doesn't matter at all. Two duplicated value will not be a problem since either you found you complement(the duplicate one) and return the indices or you replace the old index with the newly found one. No error no special case for duplicated values.
liscuss.leetcode.com/user/sapocaly)

Ziyang1103 commented 6 days ago

I just doubt that the hashmap could only be used by the list with different value, or it might has the error.
liscuss.leetcode.com/user/ziyang1103)

DieSon commented 6 days ago

@kevin.huang.x3 (https://discuss.leetcode.com/uid/160288) in the hash map ,one map only corresponding to one key ,so this function only is useful to the array with different value.
liscuss.leetcode.com/user/dieson)

kevin.huang.x3 commented last week

Will this work with duplicates? Like if you have the same input twice: [1,2,1,2] target 4. Then the answer would be 2+2=4. But I think the code says the complement can't equal itself? map.get(complement) != i
liscuss.leetcode.com/user/kevin-

nandwana92 commented last week

@samnguyen Got it, thanks.
liscuss.leetcode.com/user/nandwana92)

sam123 commented last week

@nandwana92 (https://discuss.leetcode.com/uid/157236) Two separate loops through the elements is O(n) + O(n) = O(2n) = O(n). If the second loop occurred within the first it would be O(n)*O(n) = O(n^2)
liscuss.leetcode.com/user/sam123)

nandwana92 commented last week

For "Approach #2 (Two-pass Hash Table)", as we have to loop through the list of elements twice, once to construct the hash map and other time to find the complement, how is the time complexity O(n), my understanding says it should be O(n2). Can someone explain this to me?
liscuss.leetcode.com/user/nandwana92)

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

Load more comments...