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.
>
> **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]
  - Approach #2 (Two-pass Hash Table) [Accepted]
  - Approach #3 (One-pass Hash Table) [Accepted]

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.

(/ratings/107/7/?return=/articles/two-sum/) (/ratings/107/7/?return=/articles/two-sum/) (/ratings/107/7/?return=/articles/two-sum/) (/ratings/107/7/?return=/articles/two-sum/) (/ratings/107/7/?return=/articles/two-sum/)

Average Rating: 4.79 (279 votes)

< Previous (/articles/reverse-linked-list/)   Next > (/articles/first-bad-version/)

Subscribe

subscribe for articles.

Join the conversation

Login Register

Strill commented 5 days ago

(/discuss.leetcode.com/user/strill)

@miziaiba (https://discuss.leetcode.com/uid/62080) You'll get [0,3].

miziaiba commented last week

(/discuss.leetcode.com/user/miziaiba)

For approach 2 and 3, if there is two repeated value in the array(e.g. [0, 3, 4, 0] and target = 0), when I find the index of (0 - 0), which index will i get???

✉ Send Feedback (mailto:admin@leetcode.com?subject=Feedback)

brianliu7 commented last week

(/discuss.leetcode.com/user/brianliu7)

@ayuanx (https://discuss.leetcode.com/uid/58648) I believe you are *technically* correct about the O(1) lookup vs the O(logN). I think for learning purposes, e.g. this problem is meant to illustrate the advantages of hashing vs. array as a data structure and ignores whats going on in reality in the background (using a tree).

reading more about it it seems that it performs closer to O(1), but given a really really unlucky set of collisions worst case running scenario can be O(n), but extremely rare / only with edge cases http://stackoverflow.com/questions/1055243/is-a-java-hashmap-really-o1 (http://stackoverflow.com/questions/1055243/is-a-java-hashmap-really-o1)

brianliu7 commented last week

(/discuss.leetcode.com/user/brianliu7)

@chenxiaokai (https://discuss.leetcode.com/uid/58972) I just ran your code (c code) and in the comments it mentions you must use malloc() to allocate memory and the caller will use free on the object; so the code did not work for me

brianliu7 commented last week

(/discuss.leetcode.com/user/brianliu7)

@chenxiaokai (https://discuss.leetcode.com/uid/58972) you are returning the INDICES of the array, which are correct (0 and 2), NOT the VALUES THEMSELVES. (0 and 3)