LeetCode #1 - Two Sum

September 26, 2020

Hello happy people 👋! Today we are going to discuss the very first problem on the LeetCode.

0001 - Two Sum.

Problem Statement

Given an array of integers `nums` and an integer `target`, return indices of the two numbers such that they add up to `target`.

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

You can return the answer in any order.

Constraints

- $2 <= nums.length <= 10^5$
- $-10^9 <= nums[i] <= 10^9$
- $-10^9 <= target <= 10^9$
- Only one valid answer exists.

Analysis

So, we have to find two numbers among all the numbers in an array such that when they are added, they give us a third number equals to `target` .

For e.g.,

```
If nums = [2,7,11,15] and target = 9,
then we should return 2 and 7 because 2 + 7 = 9
```

In simpler words, we need to find -

```
c = a + b
where,
c ⇒ target
a and b ⇒ elements from the given array
```

Also, the given constraints suggest that there is only **ONE** valid answer along with the ranges of the elements in the array and the target.

Approach

After we have analyzed our problem, now it's time to think about the solution.

Approach #1

The first solution that comes to mind is -

- Take one element
- Add this element with every other element
- After adding, compare the sum with the given target
- If the sum is equal to the target, return the indices of these two elements
- If the sum is not equal to the target, we check for the next pair

Pretty simple, eh 😉? Let's analyze the **time** and **space** complexities.

- ***Time Complexity***

Let's say the array has `n` elements then -

```
For 1st element - we will check for (n - 1) elements
For 2nd element - we will check for (n - 2) elements
For 3rd element - we will check for (n - 3) elements
and so on ...

Thus, the total iterations would be - [(n - 1) + (n - 2) + (n - 3) + ... + 2 + 1]
```

If we simplify the above expression we will get -

$n * (n - 1) / 2 = n^2 - 2n \approx n^2$

Thus, our time complexity would be - ***O(n^2)***.

- ***Space Complexity***

Since we are not using any extra data structure hence our space complexity would be ***O(1)***.

This approach is simple and intuitive but it takes quadratic time which is bad for large inputs. Let's see if we have any efficient approach to solve this problem. As a matter of fact, we do 😄!

Approach #2

In the earlier approach, we did what naturally came to us. But let's try to really **think** through this problem. Hmm… 😉

When we were analyzing the problem, we came across the following equation

```
c = a + b
```

but can we write the above equation as -

```
b = c - a
```

Of course we can. We are mathematicians after all 😁.

The benefit of writing this equation is that now we can iterate through the array only once and calculate the difference of `target (c)` and the `current element (a)` and find the `other element (b)`. The idea is as follows -

- Loop through the array
- Check if we have encountered the current element before
- If we have, we will return the index of this element and the index of the difference of the `target` and the `current element`. We will only encounter this element before only if we have saved the difference of `target` and the element which when added to the current element gives the `target` itself. This is confusing I know but once it clicks, it's very obvious.
- If we haven't, the we will save the difference of the `target` and `current element`.
- Repeat the process

Since we need to store our `difference`, we need a data structure which can store the `difference` and its corresponding index. So what do you think, which data structure can help us? Of course, we need a `Map` where we will store the difference as `key` and the corresponding index as `value`.

- ***Time Complexity***

Since we are iterating the array only once, the time complexity would be ***O(n)***.

- ***Space Complexity***

Since we need a Map of the size of the array, the space complexity would be ***O(n)***.

Code

Now we have an approach to solve this problem, let's write some code -

Java

```java
package org.redquark.tutorials.leetcode;

import java.util.Arrays;
import java.util.HashMap;
import java.util.Map;

public class TwoSum {

    public int[] twoSum(int[] nums, int target) {
        // Array to store result
        int[] result = new int[2];
        // This map will store the difference and the corresponding index
        Map<Integer, Integer> map = new HashMap<>();
        // Loop through the entire array
        for (int i = 0; i < nums.length; i++) {
            // If we have seen the current element before
```

```
                // It means we have already encountered the other number of the pair
                if (map.containsKey(nums[i])) {
                    // Index of the current element
                    result[0] = i;
                    // Index of the other element of the pair
                    result[1] = map.get(nums[i]);
                }
                // If we have not seen the current before
                // It means we have not yet encountered any number of the pair
                else {
                    // Save the difference of the target and the current element
                    // with the index of the current element
                    map.put(target - nums[i], i);
                }
            }
            return result;
        }
    }
```

Python

```
from typing import List


def twoSum(nums: List[int], target: int) -> List[int]:
    # List to store results
    result = []
    # Dictionary to store the difference and its index
    index_map = {}
    # Loop for each element
    for i, n in enumerate(nums):
        # Difference which needs to be checked
        difference = target - n
```

```
            if difference in index_map:
                result.append(i)
                result.append(index_map[difference])
                break
            else:
                index_map[n] = i
        return result
```

JavaScript

```
var twoSum = function (nums, target) {
    // Array to store the result
    result = [];
    // Map to store the difference and its index
    index_map = new Map();
    // Loop for each element in the array
    for (let i = 0; i < nums.length; i++) {
        let difference = target - nums[i];
        if (index_map.has(difference)) {
            result[0] = i;
            result[1] = index_map.get(difference);
            break;
        } else {
            index_map.set(nums[i], i);
        }
    }
    return result;
};
```

Kotlin

```
package org.redquark.tutorials.leetcode

fun twoSum(nums: IntArray, target: Int): IntArray {
    // Array to store result
    val result = IntArray(2)
    // This map will store the difference and the corresponding index
    val map: MutableMap<Int, Int> = HashMap()
    // Loop through the entire array
    for (i in nums.indices) {
        // If we have seen the current element before
        // It means we have already encountered the other number of the pair
        if (map.containsKey(nums[i])) {
            // Index of the current element
            result[0] = i
            // Index of the other element of the pair
            result[1] = map[nums[i]]!!
            break
        } else {
            // Save the difference of the target and the current element
            // with the index of the current element
            map[target - nums[i]] = i
        }
    }
    return result
}
```

Conclusion

I hope you liked this post. Here, we solved the very first problem on LeetCode in ***O(n)*** time and ***O(n)*** space.

You can find the complete source code on GitHub. If you find it useful, consider giving it a star ⭐.

- Java
- Python
- JavaScript
- Kotlin Feel free to share your thoughts about this post in comments' section. I'd love to hear your feedback.

Happy coding 😄 and Namaste 🙏!

Created and maintained by **@Anirudh Sharma**

I love to learn and share. Hence, this site has no ads, no affiliation links, or any BS. If you like what you see, give me a thumbs up.

← Day 20 - Queries In AEM (Part II)

LeetCode #2 - Add Two Numbers Represented By Linked Lists →

3 Comments - *powered by utteranc.es*

pinkflyod1 commented 2 months ago

Very nicely explained. This article along with one on code recipe helped me clear all my doubts on this problem. Leaving the link to the other article in case someone finds it useful: https://www.code-recipe.com/post/two-sum

AvinashSuresh19 commented 3 weeks ago

Can you help us learn in c++?

ani03sha commented a week ago Owner

@AvinashSuresh19, I am familiar with C++ but don't have mastery in it. This is a nice website that has solutions in C++, Java, and Python. I think this will prove useful to you - https://walkccc.me/LeetCode/.

Write Preview

Sign in to comment

Styling with Markdown is supported

Sign in with GitHub