Project Write-Up: Two Sum

Author(s): Shalom

Date: 12/18/2024

# Table of Contents

1. Executive Summary  
2. Introduction  
3. System Design and Architecture  
4. Implementation Details  
5. Testing and Validation  
6. Usage Instructions  
7. Challenges and Solutions  
8. Future Improvements  
9. Conclusion  
10. Appendix (if needed)

# 3. Executive Summary

This project addresses the problem of finding two indices in an array `nums` whose values sum to a given target. The solution uses a nested loop approach to check all pairs and return the indices of the two numbers. The current implementation has a time complexity of O(n^2), though a more efficient approach exists.

# 4. Introduction

Objective: The goal is to find two numbers in an integer array that add up to a specific target, and return their indices.

Background: The Two Sum problem is a common algorithmic problem used to practice array manipulation and basic iteration.

Scope: The solution focuses on solving the Two Sum problem using a straightforward approach, while also hinting at more efficient algorithms with lower time complexities.

# 5. System Design and Architecture

High-Level Design: The solution iterates through the array with two pointers (`p1` and `p2`), comparing the sum of the elements at these pointers to the target.

Component Overview:  
- `nums`: An array of integers.  
- `target`: The integer target that we want to match by summing two elements of `nums`.  
- `sum`: A two-element array that stores the indices of the two numbers that add up to the target.

Design Decisions:  
- We use nested loops to iterate through each pair of numbers in the array. This is a brute-force solution that checks all combinations, leading to an O(n^2) time complexity.

# 6. Implementation Details

Code Structure:   
The method `twoSum(int[] nums, int target)` checks all possible pairs of elements from `nums` to find the two that sum to `target`.

Algorithm:   
- Step 1: Initialize a two-element array `sum` to store the indices of the solution.  
- Step 2: Use two nested loops, `p1` and `p2`, to compare the sum of each pair of elements.  
- Step 3: If the sum of the pair matches the target, store the indices in `sum` and return it.  
- Step 4: If no solution is found, continue iterating through the array.

Code Snippet:

class Solution {  
 public int[] twoSum(int[] nums, int target) {  
 int [] sum = new int[2];  
 int p1 = 0;  
 for(int i = 0; i< nums.length-1; i++)  
 {  
 int p2 = p1+1;  
 if(nums[p1] + nums[p2] == target)  
 {  
 sum[0] = p1;  
 sum[1] = p2;  
 }  
 else   
 {  
 while (p2 < nums.length-1)  
 {  
 p2++;  
 if(nums[p1] + nums[p2] == target)  
 {  
 sum[0] = p1;  
 sum[1] = p2;  
 }  
 }  
 p1++;  
 }  
 }  
 return sum;  
 }  
}

# 7. Testing and Validation

Testing Approach:  
The solution is tested using different arrays and target values, ensuring that the function returns the correct indices when two numbers sum to the target.

Test Cases:  
1. `nums = [2, 7, 11, 15], target = 9` returns `[0, 1]`  
2. `nums = [3, 2, 4], target = 6` returns `[1, 2]`  
3. `nums = [3, 3], target = 6` returns `[0, 1]`

Known Issues: The current implementation has a time complexity of O(n^2) because it uses nested loops to check all pairs. This can be improved to O(n) using a hash map.

# 8. Usage Instructions

Installation: This function can be added to any Java project. No installation is required.

Running the Project: To use the `twoSum` function, call it with an array of integers and the target value:

int[] nums = {2, 7, 11, 15};  
int target = 9;  
Solution solution = new Solution();  
int[] result = solution.twoSum(nums, target);  
System.out.println(Arrays.toString(result)); // Output: [0, 1]

Example Output:  
[0, 1]

# 9. Challenges and Solutions

Challenges:  
The main challenge was ensuring that all pairs are checked for the sum, and handling the condition where each number can only be used once.

Solutions:  
The solution correctly handles these conditions, but the brute-force approach is inefficient. A hash map-based solution could reduce the time complexity to O(n).

# 10. Future Improvements

Optimizations: To improve efficiency, the algorithm can be modified to use a hash map, where the key is the number and the value is its index. This would allow us to find the complement of each number in O(1) time, reducing the time complexity to O(n).

Known Issues: There are no further issues with this solution, but optimization is possible.

# 11. Conclusion

The current solution solves the Two Sum problem using a brute-force approach, but it can be optimized to achieve better performance. Despite its time complexity, it demonstrates the logic needed to solve the problem and return the correct indices.

# 12. Appendix

References: LeetCode Problem 1: Two Sum (https://leetcode.com/problems/two-sum/)