Gmail William Ting <williamdjting@gmail.com>

FW: Solution for Two Sum

1 message

William Ting <william_ting_2@sfu.ca> Wed, May 3, 2023 at 5:10 AM
To: "williamdjting@gmail.com" <williamdjting@gmail.com>

From: Daily Byte
Sent: Wednesday, May 3, 2023 5:10:06 AM (UTC-08:00) Pacific Time (US & Canada)
To: William Ting
Subject: Solution for Two Sum

The Daily Byte

Hi there,

We recently sent you a problem called Two Sum and we thought this would be a good time to go over the solution to that problem. Normally these solutions are for our paid customers but we wanted to make sure that you are getting the most out of your free subscription. While there are always simple answers to the problems it is important to keep in mind Big-O notation and to try and analyze whether you are achieving an optimal solution. To recap, the question was as follows:

Given an array of integers, return whether or not two numbers sum to a given target,
`k` . Note: you may not sum a number with itself

```
Ex: [1, 3, 8, 2], k = 10 -> true (8 + 2)
Ex: [3, 9, 13, 7], k = 8 -> false
Ex: [4, 2, 6, 5, 2], k = 4 -> true (2 + 2)
```

A simple solution to this problem might try picking two numbers and seeing if they sum to k. If they do we return true and if we iterate through all the numbers not being able to find such a pair, we return false.

```
public boolean twoSum(int[] nums, int target) {
    for (int i = 0; i < nums.length; i++) {
        for (int j = i + 1; j < nums.length; j++) {
            if (nums[i] + nums[j] == k) {
                return true;
            }
        }
    }
    return false;
}
```

While this approach works, it's rather slow resulting from its $O(N^2)$ runtime. N^2 coming from us comparing each of our `N` numbers to every other number in our array. By using a bit of space, we can knock down this runtime. Instead of doing a linear scan to find a number to complement our current number, we can introduce a hash map to remember what numbers we have seen. Now for each number, we can ask the question, "have we seen a number that when added to our current number will give us a sum of k?" If we have, we can return true, and if we haven't can simply add our current number to our hash map to remember for the future. Following below is the solution described here

```
public boolean twoSum(int[] nums, int k) {
    HashMap<Integer, Integer> map = new HashMap<>();
    for(int i = 0; i < nums.length; i++) {
        int difference = k - nums[i];
        if(map.containsKey(difference)) {
            return true;
        }
        map.put(nums[i], i);
    }
    return false;
}
```

Big-O Analysis

Runtime: O(N) where N is the number of integers in our list.
Space complexity: O(N) because we use a hash map to store all N numbers in the worst case.

We hope this was helpful, as always we'd love to hear any feedback or questions you might have. If you want to receive these solutions every day you can sign up for them here.