Two Sum

18 февраля 2020 г. 13:35

The Task

I. Condition

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].
```

II. Given Prototype

Programming Language: Python3

```
class Solution:
    def twoSum(self, nums: List[int], target: int) -> List[int]:
```

The Idea

I. Condition understanding

First of all, I think it's essential to comprehend the condition.

1) According the condition and the prototype we have:
   a. A list of **integers** numbers
   b. A target which is **integer** too
2) Also from the condition we receive that **each input has exactly one solution**, therefore we don't need to consider the case when there is no solution or there is not only one solution .
3) We can't use the same element (For example, nums[0] + nums[0])
4) The sum of two elements with different indices have to be equal the target (For example, nums[i] + nums[j] = target)
5) We should derive the indices of the suitable elements.
   (For example, nums[i] + nums[j] = target, then we need the [I, j]

II. Solutions

1) The first idea which I got is a simple brute:

$$\begin{cases} return\ [i,j], & x[i]+y[j]=target \\ i=i+1 & x[i]+y[j]\neq target, \\ j=j+1 & i=n \end{cases}$$

where,

$$\begin{cases} x[i],y[j]\in X, X\in Z^n \\ i,j\ \in [0;n-1] \\ target\ \in Z \end{cases}$$

In other words, we make an effort to sum each element with the remain elements in the array. For example:

a.

2	7	11	15

2	7	11	15

b.

2	7	11	15

2	7	11	15

And so on. Obviously, we do this until the sum is equal **target.**
The complexity of this method is $O(n^2)$, due to in the worst case we need to iterate the array $(n\ -1)*n$ times.

2) The other idea it's the usage of the hash-map.

$$\begin{cases} return\ [i,j] & \exists hm[key]\ (target\ -x[i]=key) \\ i=i+1 & \\ key=x[i] & \exists hm[key]\ (target\ -x[i]=key), \end{cases}$$

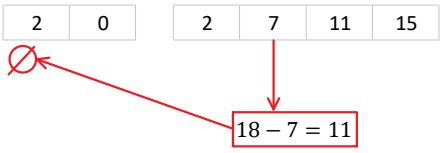
$$\begin{cases} return\ [i,j] & \exists hm[key]\ (target\ - x[i] = key) \\ \begin{cases} i = i + 1 \\ key = x[i] \\ hm[key] = i \end{cases} & \nexists hm[key]\ (target\ - x[i] = key)' \end{cases}$$

where,

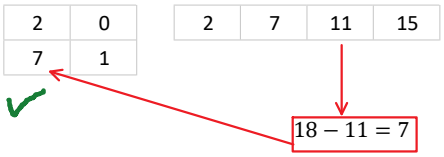
$$\begin{cases} x[i] \in X, X \in Z^n \\ i\ \in [1; n-1] \\ hm\ - hash\ map, where\ \boldsymbol{key}\ it's\ \boldsymbol{x[i]}\ and\ \boldsymbol{hm[key]}\ it's\ \boldsymbol{i} \end{cases}$$

In other words, we're look for the element equal to the $(target\ - x[i])$, if we're within an array and didn't find such $x[i]$, then we're paste the $x[i]$ and i into the hash map. It will look like that $hm[x[i]] = i$. It looks like that (where target = 18):

a.



b.



The complexity of this method is $O(n+1)$, because of using the hash-map where complexity of the element's deriving is $O(1)$ and the moving through the array $O(n)$ the worst case will $O(n+1)$.

My Solution

Of course I chose the second method. The code was written using Python3:

```
class Solution:
    def twoSum(self, nums: List[int], target: int) -> List[int]:
        dict = {str(nums[0]): 0}
        for i in range(1, len(nums)):
            supposed_index = dict.get(str(target-nums[i]))
            if supposed_index != None:
                return ([supposed_index, i])
            else:
                dict.update({str(nums[i]): i})
```