March 2, 2022

Leetcode
Problem 1: Two Sum

Mitch Sullivan

Problem

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.

Example 1:

```
Input: nums = [2,7,11,15], target = 9
Output: [0,1]
Explanation: Because nums[0] + nums[1] == 9, we return [0, 1].
```

Example 2:

```
Input: nums = [3,2,4], target = 6
Output: [1,2]
```

Example 3:

```
Input: nums = [3,3], target = 6
Output: [0,1]
```

Constraints:

- $2 \leq$ `nums.length` $\leq 10^4$
- $-10^9 \leq$ `nums[i]` $\leq 10^9$
- $-10^9 \leq$ `target` $\leq 10^9$
- **Only one valid answer exists.**

Follow-up:

Can you come up with an algorithm that is less than $O(n^2)$ time complexity?

Solution(s)

Solution 1: Brute Force

The brute force approach here is an $O(n^2)$ algorithm using nested `for` loops.

Data: A sequence $(a_i)_{i\in[0,n)}$ of n integers
Data: An integer N
Result: The unique unordered pair $\{j,k\}$ of the indices $j,k \in [0,n)$ of the two numbers from $(a_i)_{i\in[0,n)}$ s.t. $a_j + a_k = N$

for $j \in [0,n)$ **do**
  for $k \in (j,n)$ **do**
    if $a_j + a_k = N$ **then**
      return $\{j,k\}$
    end
  end
end

The worst case scenario for this algorithm is when the numbers are the last two in the array, in which case the total number of iterations is

$$(n-1)+(n-2)+\cdots+2+1 = \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} - n = \frac{1}{2}n^2 - \frac{3}{2}n.$$

Therefore, the algorithm has, as stated above, $O(n^2)$ time complexity.

The benefit of this algorithm is that it is very easy to understand and to implement.

Solution 2

A more efficient approach is as follows:

Data: A sequence $(a_i)_{i\in[0,n)}$ of n integers
Data: An integer N
Result: The unique unordered pair $\{j,k\}$ of the indices $j,k \in [0,n)$ of the two numbers from $(a_i)_{i\in[0,n)}$ s.t. $a_j + a_k = N$

$X_0 \leftarrow \{(a_0, 0)\}$;
for $j \in (0,n)$ **do**
  $x \leftarrow N - a_j$;
  if $x \in \pi_{j1}(X_j)$ **then**
    return $\{\pi_{j2}(x), j\}$
  else
    $X_j \leftarrow X_{j-1} \bigcup \{(a_j, j)\}$;
  end
end

The idea behind the sets $X_0, X_1, \ldots, X_{n-1}$ is to keep track of the elements of $(a_i)_{i\in[0,n)}$ that have been "visited" during the iteration process, whose elements are the ordered pairs of the elements $a_0, a_1, \ldots, a_{n-1} \in (a_i)_{i\in[0,n)}$ and their associated indices, so that, for each $k \in [0,n)$, $X_k = \{(a_0,0),(a_1,1),\ldots,(a_k,k)\}$. For each $k \in [0,n)$, the functions $\pi_{k\alpha}$ are the projection functions on X_k which map each (a_k,k) to the α^{th} coordinate:

$$\pi_{k1} : (a_k,k) \mapsto a_k$$
$$\pi_{k2} : (a_k,k) \mapsto k.$$

At the k^{th} iteration, the idea is to search for a number $x \in (a_i)_{i\in[0,n)}$ that satisfies $x + a_k = N$, or, equivalently, search for an $x \in (a_i)_{i\in[0,n)}$ that satisfies $x = N - a_k$.

The worst case scenario for this algorithm is when the numbers are the last two in the array, in which case $n-1$ iterations would be required. If the search operation can be performed in $O(1)$ time, this algorithm is therefore $O(n)$.

References

[1] https://leetcode.com/problems/two-sum/

[2] https://www.code-recipe.com/post/two-sum