Problem 1. Two Sum

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 major benefit of this algorithm is that it is very easy to understand and to implement. A second benefit is that it has $O(1)$ space complexity.

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 $j,k \in [0,n)$, $j \neq k$, of the two numbers from $(a_i)_{i\in[0,n)}$ s.t. $a_j + a_k = N$

$X \leftarrow \varnothing$;
for $i \in [0,n)$ **do**
  $x \leftarrow N - a_i$;
  if $x \in \pi_1(X)$ **then**
    return $\{\pi_2 \circ \pi_1^{-1}(x), i\}$
  else
    $X \leftarrow X \bigcup \{(a_i, i)\}$;
  end
end

The idea behind the set X is to keep track of the elements of $(a_i)_{i\in[0,n)}$ that have been "visited" during the iteration process. At the end of the k^{th} iteration, $k \in [0,n)$, the elements of X are just the ordered pairs $(a_0,0),(a_1,1),\ldots,(a_k,k)$ of the elements $a_0,a_1,\ldots,a_k \in (a_i)_{i\in[0,n)}$ and their associated indices. The functions π_α are the projection functions on X which map each (a_k,k) to the α^{th} coordinate:

$$\pi_1 : (x, i_x) \mapsto x$$
$$\pi_2 : (x, i_x) \mapsto i_x.$$

There is a bit of a technical issue with the notion of the inverse π_1^{-1} of the first projection function π_1 in that it's possible for there to exist $j, k \in [0, n)$, $j \neq k$, that satisfy $a_j = a_k$. This would mean that, if $x = a_j = a_k$, π^{-1} would map x to both j and k, violating the definition of a function. However, the constraint that there is only one valid answer ensures that if an x exists that satisfies $x = N - a_i$ for some $i \in [0, n)$, it is unique. A restriction on the range of π_1 could be made to make the definition rigorous, however just allowing the abuse of notation with the above understanding seems like the simpler and clearer choice here.

At the k^{th} iteration, the idea is to search X for a the pair $(x, i_x) \in X$ that satisfies $x + a_k = N$, or, equivalently, $x = N - a_k$. If no such x exists, the set X gets expanded by adding (a_k, k), which in essence marks a_k as having been "visited". The worst case scenario for this algorithm is when $N - a_{n-1}$ gives the first match, in which case n iterations are required. If the search operation can be performed in $O(1)$ time, the time complexity of this algorithm is therefore $O(n)$. Also in this worst case, the set x will contain $n - 1$ elements, so the space complexity is $O(n)$.

References

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

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