Longest Consecutive Sequence

Given an unsorted array of integers, find the length of the longest consecutive elements sequence.

For example,
Given `[100, 4, 200, 1, 3, 2]`,
The longest consecutive elements sequence is `[1, 2, 3, 4]`. Return its length: `4`.

Your algorithm should run in O(*n*) complexity.

Solution 1

We will use HashMap. The key thing is to keep track of the sequence length and store that in the boundary points of the sequence. For example, as a result, for sequence {1, 2, 3, 4, 5}, map.get(1) and map.get(5) should both return 5.

Whenever a new element **n** is inserted into the map, do two things:

1. See if **n - 1** and **n + 1** exist in the map, and if so, it means there is an existing sequence next to **n**. Variables **left** and **right** will be the length of those two sequences, while **0** means there is no sequence and **n** will be the boundary point later. Store **(left + right + 1)** as the associated value to key **n** into the map.
2. Use **left** and **right** to locate the other end of the sequences to the left and right of **n** respectively, and replace the value with the new length.

Everything inside the **for** loop is O(1) so the total time is O(n). Please comment if you see something wrong. Thanks.

```java
public int longestConsecutive(int[] num) {
    int res = 0;
    HashMap<Integer, Integer> map = new HashMap<Integer, Integer>();
    for (int n : num) {
        if (!map.containsKey(n)) {
            int left = (map.containsKey(n - 1)) ? map.get(n - 1) : 0;
            int right = (map.containsKey(n + 1)) ? map.get(n + 1) : 0;
            // sum: length of the sequence n is in
            int sum = left + right + 1;
            map.put(n, sum);

            // keep track of the max length
            res = Math.max(res, sum);

            // extend the length to the boundary(s)
            // of the sequence
            // will do nothing if n has no neighbors
            map.put(n - left, sum);
            map.put(n + right, sum);
        }
        else {
            // duplicates
            continue;
        }
    }
    return res;
}
```

written by dchen0215 original link here

Solution 2

use a hash map to store boundary information of consecutive sequence for each element; there 4 cases when a new element i reached:

1) neither i+1 nor i-1 has been seen: m[i]=1;

2) both i+1 and i-1 have been seen: extend m[i+m[i+1]] and m[i-m[i-1]] to each other;

3) only i+1 has been seen: extend m[i+m[i+1]] and m[i] to each other;

4) only i-1 has been seen: extend m[i-m[i-1]] and m[i] to each other.

```
int longestConsecutive(vector<int> &num) {
    unordered_map<int, int> m;
    int r = 0;
    for (int i : num) {
        if (m[i]) continue;
        r = max(r, m[i] = m[i + m[i + 1]] = m[i - m[i - 1]] = m[i + 1] + m[i - 1]
+ 1);
    }
    return r;
}
```

written by mzchen original link here

Solution 3

First turn the input into a *set* of numbers. That takes O(n) and then we can ask in O(1) whether we have a certain number.

Then go through the numbers. If the number n is the start of a streak (i.e., n-1 is not in the set), then test m = n+1, n+2, n+3, ... and stop at the first number m *not* in the set. The length of the streak is then simply m-n and we update our global best with that. Since we check each streak only once, this is overall O(n). This ran in 44 ms on the OJ, one of the fastest Python submissions.

```python
class Solution:
    def longestConsecutive(self, nums):
        nums = set(nums)
        best = 0
        for n in nums:
            if n - 1 not in nums:
                m = n + 1
                while m in nums:
                    m += 1
                best = max(best, m - n)
        return best
```

written by StefanPochmann original link here

From Leetcoder.