seanpgallivan
Posted on Feb 4, 2021

Solution: Longest Harmonious Subsequence

#algorithms #javascript #leetcode

Leetcode Solutions (161 Part Series)

1	Solution: Next Permutation
2	Solution: Trim a Binary Search Tree
...	**157 more parts...**
160	Solution: Out of Boundary Paths
161	Solution: Redundant Connection

This is part of a series of Leetcode solution explanations (index). If you liked this solution or found it useful, ***please like*** *this post and/or* ***upvote*** *my solution post on Leetcode's*

forums.

Leetcode Problem #594 (*Easy*): Longest Harmonious Subsequence

---

Description:

We define a harmonious array as an array where the difference between its maximum value and its minimum value is ***exactly*** `1`.

Given an integer array `nums`, *return the length of its longest harmonious subsequence among all its possible subsequences.*

A ***subsequence*** *of array is a sequence that can be derived from the array by deleting some or no elements without changing the order of the remaining elements.*

Examples:

Example 1:	
Input:	nums = [1,3,2,2,5,2,3,7]
Output:	5
Explanation:	The longest harmonious subsequence is [3,2,2,2,3].

Example 2:	
Input:	nums = [1,2,3,4]
Output:	2

Example 3:	

Input:	nums = [1,1,1,1]
Output:	0

Constraints:

- 1 <= nums.length <= 2 * 10^4
- -10^9 <= nums[i] <= 10^9

Idea:

Since our target harmonious array is dealing with the absolute value of its elements and since it's a subsequence of our numbers array (**N**), we don't need to worry about the order of numbers or their index in **N**.

If all we care about is *what* numbers appear in **N** and not their order or index, then it means that we should start by building a **frequency map** from **N**.

Then we can just iterate through the entries in our frequency map (**fmap**) and keep track of the largest value found by adding each number's (**key**) frequency (**val**) with the frequency of **key+1**.

We should then **return** the best result (**ans**).

Implementation:

Since javascript's **Map()** stores its keys as strings, you need to use some method of converting the key back into a number before adding **1**. The normal way to do this is with **parseInt()**, but applying a double **bitwise NOT** (**~**) does the same thing far more efficiently, as long as the number is greater than **-2^31** and less than **2^31**.

Javascript Code:

```
var findLHS = function(N) {
    let fmap = new Map(), ans = 0
    for (let num of N)
        fmap.set(num, (fmap.get(num) || 0) + 1)
    for (let [key,val] of fmap)
        if (fmap.has(~~key+1))
            ans = Math.max(ans, val + fmap.get(~~key+1))
    return ans
};
```


Leetcode Solutions (161 Part Series)

1 Solution: Next Permutation

2 Solution: Trim a Binary Search Tree

... **157 more parts...**

160 Solution: Out of Boundary Paths

161 Solution: Redundant Connection

Discussion (0)

Code of Conduct • Report abuse

seanpgallivan

Fledgling software developer; the struggle is a Rational Approximation.

LOCATION

Seattle, WA, USA

EDUCATION

Flatiron School (Software Engineering)

WORK

Full Stack Software Engineer

JOINED

Dec 16, 2019

More from seanpgallivan

Solution: Redundant Connection

#algorithms #javascript #java #python

Solution: Out of Boundary Paths

#algorithms #javascript #java #python

Solution: Pascal's Triangle

#algorithms #javascript #java #python