Number of Longest Increasing Subsequence

Given an unsorted array of integers, find the number of longest increasing subsequence.

Example 1:

```
Input: [1,3,5,4,7]
Output: 2
Explanation: The two longest increasing subsequence are [1, 3, 4, 7] and [1, 3, 5, 7]
.
```

Example 2:

```
Input: [2,2,2,2,2]
Output: 5
Explanation: The length of longest continuous increasing subsequence is 1, and there
are 5 subsequences' length is 1, so output 5.
```

Note: Length of the given array will be not exceed 2000 and the answer is guaranteed to be fit in 32-bit signed int.

Solution 1

The idea is to use two arrays `len[n]` and `cnt[n]` to record the maximum length of Increasing Subsequence and the coresponding number of these sequence which ends with `nums[i]`, respectively. That is:

`len[i]`: the length of the Longest Increasing Subsequence which ends with `nums[i]`.
`cnt[i]`: the number of the Longest Increasing Subsequence which ends with `nums[i]`.

Then, the result is the sum of each `cnt[i]` while its corresponding `len[i]` is the maximum length.

Java version:

```
public int findNumberOfLIS(int[] nums) {
        int n = nums.length, res = 0, max_len = 0;
        int[] len =  new int[n], cnt = new int[n];
        for(int i = 0; i<n; i++){
            len[i] = cnt[i] = 1;
            for(int j = 0; j <i ; j++){
                if(nums[i] > nums[j]){
                    if(len[i] == len[j] + 1)cnt[i] += cnt[j];
                    if(len[i] < len[j] + 1){
                        len[i] = len[j] + 1;
                        cnt[i] = cnt[j];
                    }
                }
            }
            if(max_len == len[i])res += cnt[i];
            if(max_len < len[i]){
                max_len = len[i];
                res = cnt[i];
            }
        }
        return res;
    }
```

C++ version: (use `vector<pair<int, int>> dp` to combine `len[]` and `cnt[]`)

```cpp
    int findNumberOfLIS(vector<int>& nums) {
        int n = nums.size(), res = 0, max_len = 0;
        vector<pair<int,int>> dp(n,{1,1});            //dp[i]: {length, number of L
IS which ends with nums[i]}
        for(int i = 0; i<n; i++){
            for(int j = 0; j <i ; j++){
                if(nums[i] > nums[j]){
                    if(dp[i].first == dp[j].first + 1)dp[i].second += dp[j].second;
                    if(dp[i].first < dp[j].first + 1)dp[i] = {dp[j].first + 1, dp[j
].second};
                }
            }
            if(max_len == dp[i].first)res += dp[i].second;
            if(max_len < dp[i].first){
                max_len = dp[i].first;
                res = dp[i].second;
            }
        }
        return res;
    }
```

written by Vincent Cai original link here

Solution 2

The solution is based on DP.

```
For a sequence of numbers,
cnt[k] is total number of longest subsequence ending with nums[k];
len[k] is the length of longest subsequence ending with nums[k];
```

Then we have following equations

```
len[k+1] = max(len[k+1], len[i]+1) for all i <= k and nums[i] < nums[k+1];
cnt[k+1] = sum(cnt[i]) for all i <= k and nums[i] < nums[k+1] and len[i] = len[k+1]
-1;
```

Starting case and default case: cnt[0] = len[0] = 1;

```
class Solution {
public:
    int findNumberOfLIS(vector<int>& nums) {
        int n = nums.size(), maxlen = 1, ans = 0;
        vector<int> cnt(n, 1), len(n, 1);
        for (int i = 1; i < n; i++) {
            for (int j = 0; j < i; j++) {
                if (nums[i] > nums[j]) {
                    if (len[j]+1 > len[i]) {
                        len[i] = len[j]+1;
                        cnt[i] = cnt[j];
                    }
                    else if (len[j]+1 == len[i])
                        cnt[i] += cnt[j];
                }
            }
            maxlen = max(maxlen, len[i]);
        }
        // find the longest increasing subsequence of the whole sequence
       // sum valid counts
        for (int i = 0; i < n; i++)
            if (len[i] == maxlen) ans += cnt[i];
        return ans;
    }
};
```

written by zestypanda original link here

Solution 3

If you have not solved the Longest Increasing Subsequence problem, you should do so before attempting this question. The approach is very similar and only requires augmentation of the DP array.

In the Longest Increasing Subsequence problem, the DP array simply had to store the longest length. In this variant, each element in the DP array needs to store two things: (1) Length of longest subsequence ending at this index and (2) Number of longest subsequences that end at this index. I use a two element list for this purpose.

In each loop as we build up the DP array, find the longest length for this index and then sum up the numbers at these indices that contribute to this longest length.

Here I provide two versions: (1) A slower but easier to understand version and (2) Much faster and optimized version

By Yang Shun

```python
class Solution(object):
    def findNumberOfLIS(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        # Time: O(n^2)
        # Space: O(n)
        dp, longest = [[1, 1] for i in range(len(nums))], 1
        for i, num in enumerate(nums):
            curr_longest, count = 1, 0
            for j in range(i):
                if nums[j] < num:
                    curr_longest = max(curr_longest, dp[j][0] + 1)
            for j in range(i):
                if dp[j][0] == curr_longest - 1 and nums[j] < num:
                    count += dp[j][1]
            dp[i] = [curr_longest, max(count, dp[i][1])]
            longest = max(curr_longest, longest)
        return sum([item[1] for item in dp if item[0] == longest])
```

The counting step can be optimized such that we don't count from the start when we find a longer `max_len`. This improved the speed from 10% to 88%.

```python
class Solution(object):
    def findNumberOfLIS(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        dp = [[1, 1] for i in range(len(nums))]
        max_for_all = 1
        for i, num in enumerate(nums):
            max_len, count = 1, 0
            for j in range(i):
                if nums[j] < num:
                    if dp[j][0] + 1 > max_len:
                        max_len = dp[j][0] + 1
                        count = 0
                    if dp[j][0] == max_len - 1:
                        count += dp[j][1]
            dp[i] = [max_len, max(count, dp[i][1])]
            max_for_all = max(max_len, max_for_all)
        return sum([item[1] for item in dp if item[0] == max_for_all])
```

written by yangshun original link here