<https://leetcode.com/problems/number-of-longest-increasing-subsequence/description/>

Given an integer array nums, return *the number of longest increasing subsequences.*

**Notice** that the sequence has to be **strictly** increasing.

**Example 1:**

Input: nums = [1,3,5,4,7]

Output: 2

Explanation: The two longest increasing subsequences are [1, 3, 4, 7] and [1, 3, 5, 7].

**Example 2:**

Input: nums = [2,2,2,2,2]

Output: 5

Explanation: The length of the longest increasing subsequence is 1, and there are 5 increasing subsequences of length 1, so output 5.

**Constraints:**

* 1 <= nums.length <= 2000
* -106 <= nums[i] <= 106

**Attempt 1: 2023-04-09**

**Solution 1: DP (360 min)**

**Style 1**

class Solution {

public int findNumberOfLIS(int[] nums) {

int result = 0;

int maxLen = 0;

int n = nums.length;

// len[i]: the length of the Longest Increasing Subsequence which ends with nums[i]

// count[i]: the number of the Longest Increasing Subsequence which ends with nums[i]

int[] len = new int[n];

int[] count = new int[n];

Arrays.fill(len, 1);

Arrays.fill(count, 1);

for(int i = 0; i < n; i++) {

for(int j = 0; j < i; j++) {

// If combining with i makes an increasing subsequence

if(nums[j] < nums[i]) {

// If combining with i makes another longest increasing subsequence

if(len[i] == len[j] + 1) {

/\*\*

Why is doing count[i] += count[j] and not simply increasing

the count[i] += 1 ?

Consider this example [[1,3,5,4,9,8,10]

This is the step by step output for the last index:

Length [1, 2, 3, 3, 4, 4, 2]

Count [1, 1, 1, 1, 2, 2, 1]

Length [1, 2, 3, 3, 4, 4, 3]

Count [1, 1, 1, 1, 2, 2, 1]

Length [1, 2, 3, 3, 4, 4, 4]

Count [1, 1, 1, 1, 2, 2, 1]

Length [1, 2, 3, 3, 4, 4, 4]

Count [1, 1, 1, 1, 2, 2, 2]

Length [1, 2, 3, 3, 4, 4, 5]

Count [1, 1, 1, 1, 2, 2, 2]

Length [1, 2, 3, 3, 4, 4, 5]

Count [1, 1, 1, 1, 2, 2, 4]

Let's understand when does the count for 10 get incremented from 1 to 2.

When I get a sequence of 1,3,4,10. So the reason i have 2 at the last

index is because I got two sequences with the same length

e.g. 1 3 5 10 and 1 3 4 10

Now, when we include a 9 in the sequence, why do we still have 2

sequences, because 9 goes inside those existing sequences, so the

sequences become 1 3 5 9 10 and 1 3 4 9 10. So u still have 2 sequences.

Now, comes the 8

The reason we have 4 as the answer at the last index is because think

about it, there were 2 ways to get to 8 i.e 1 3 5 8 and 1 3 4 8

Appending 10 will still get us to those two ways. But 10 has two ways

already that it can be reached without even considering 8 at all which

are 1 3 5 9 10 and 1 3 4 9 10. So the total answer is no of ways to reach

10 via 8 + ways not via 8 which is 4

\*/

count[i] += count[j];

}

// If combining with i makes a longer increasing subsequence

if(len[i] < len[j] + 1) {

len[i] = len[j] + 1;

count[i] = count[j];

}

}

}

// The result is the sum of each count[i] while its corresponding

// len[i] is the maximum length

if(maxLen == len[i]) {

result += count[i];

}

if(maxLen < len[i]) {

maxLen = len[i];

result = count[i];

}

}

return result;

}

}

**Refer to**

<https://leetcode.com/problems/number-of-longest-increasing-subsequence/solutions/107293/java-c-simple-dp-solution-with-explanation/>

The idea is to use two arrays len[n] and cnt[n] to record the maximum length of Increasing Subsequence and the corresponding 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.

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;

}

**Why doing count[i] += count[j] and not simply increasing the count[i] += 1 ?**

**Refer to**

<https://leetcode.com/problems/number-of-longest-increasing-subsequence/solutions/500880/java-dp-with-explanation/comments/619873>

<https://leetcode.com/problems/number-of-longest-increasing-subsequence/solutions/500880/java-dp-with-explanation/comments/748606>

This is the step by step output for the last index:

Length [1, 2, 3, 3, 4, 4, 2]

Count [1, 1, 1, 1, 2, 2, 1]

Length [1, 2, 3, 3, 4, 4, 3]

Count [1, 1, 1, 1, 2, 2, 1]

Length [1, 2, 3, 3, 4, 4, 4]

Count [1, 1, 1, 1, 2, 2, 1]

Length [1, 2, 3, 3, 4, 4, 4]

Count [1, 1, 1, 1, 2, 2, 2]

Length [1, 2, 3, 3, 4, 4, 5]

Count [1, 1, 1, 1, 2, 2, 2]

Length [1, 2, 3, 3, 4, 4, 5]

Count [1, 1, 1, 1, 2, 2, 4]

Let's understand when does the count for 10 get incremented from 1 to 2. When I get a sequence of 1,3,4,10. So the reason i have 2 at the last index is because I got two sequences with the same length, e.g. 1 3 5 10 and 1 3 4 10.

Now, when we include a 9 in the sequence, why do we still have 2 sequences, because 9 goes inside those existing sequences, so the sequences become 1 3 5 9 10 and 1 3 4 9 10. So u still have 2 sequences

Now, comes the 8.The reason we have 4 as the answer at the last index is because think about it, there were 2 ways to get to 8 i.e. 1 3 5 8 and 1 3 4 8.

Appending 10 will still get us to those two ways. But 10 has two ways already that it can be reached without even considering 8 at all which are 1 3 5 9 10 and 1 3 4 9 10. So the total answer is no of ways to reach 10 via 8 + ways not via 8 which is 4.

**Style 2: Another format which split out the for loop focus on final result sum up**

class Solution {

public int findNumberOfLIS(int[] nums) {

if(nums == null || nums.length == 0) {

return 0;

}

// state:

// The idea is to use two arrays f[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:

// f[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 f[i] is the maximum length.

int n = nums.length;

// intialize:

int result = 0;

int max = 0;

int[] f = new int[n];

int[] cnt = new int[n];

for(int i = 0; i < n; i++) {

f[i] = 1;

cnt[i] = 1;

}

// function

for(int i = 0; i < n; i++) {

for(int j = 0; j < i; j++) {

if(nums[j] < nums[i]) {

// Refer to

// https://discuss.leetcode.com/topic/103020/java-c-simple-dp-solution-with-explanation/9

// f[i] == f[j] + 1 means that you find another subsequence with the

// same length of LIS which ends with nums[i].

// While f[i] > f[j] + 1 means that you find a subsequence, but its

// length is smaller compared to LIS which ends with nums[i]. --> so

// for this case we will ignore

if(f[i] == f[j] + 1) {

// Important: not ++

cnt[i] += cnt[j];

}

if(f[i] < f[j] + 1) {

f[i] = f[j] + 1;

cnt[i] = cnt[j];

}

}

}

// if(max\_len == f[i]) {

// result += cnt[i];

// }

// if(max\_len < f[i]) {

// max\_len = f[i];

// result = cnt[i];

// }

// Refer to

// https://discuss.leetcode.com/topic/102976/java-with-explanation-easy-to-understand

// we can change the above section in same style as previous two question

// https://leetcode.com/problems/longest-continuous-increasing-subsequence/description/

// https://leetcode.com/problems/longest-increasing-subsequence/description/

max = Math.max(max, f[i]);

}

for(int i = 0; i < n; i++) {

if(max == f[i]) {

result += cnt[i];

}

}

// answer

return result;

}

}

**Refer to**

<https://leetcode.com/problems/number-of-longest-increasing-subsequence/solutions/1230468/c-clean-dp-solution-easy-and-explained/>

class Solution {

public:

int findNumberOfLIS(vector<int>& nums) {

const int n = nums.size();

vector<int> lis(n,1); // stores length of longest sequence till i-th position

vector<int> count(n,1); // stores count of longest sequence of length lis[i]

int maxLen = 1; // maximum length of lis

// O(N^2) DP Solution

for(int i=1;i<n;i++){

for(int j=0;j<i;j++){

if(nums[i]>nums[j]){

if(lis[j] + 1 > lis[i]){ // strictly increasing

lis[i] = lis[j] + 1;

count[i] = count[j];

}

// this means there are more subsequences of same length ending at length lis[i]

else if(lis[j]+1 == lis[i]){

count[i] += count[j];

}

}

}

maxLen = max(maxLen,lis[i]);

}

int numOfLIS = 0;

// count all the subseq of length maxLen

for(int i=0;i<n;i++){

if(lis[i]==maxLen)

numOfLIS += count[i];

}

return numOfLIS;

}

};