Author: Xiaoqiang Yan

LeetCode 673 Number of LIS

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 subs
equence is 1, and there are 5 subsequences' length is 1, so o
utput 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

At the beginning, I think if we find the LIS, and we can scan elements in

dp, and then count number of LIS. But this is wrong! like `nums = [1,3,5,4,7], dp = [1,2,3,3,4]`, we only find one `4`, but there are `2` LIS.

So we need a 2D DP to store the LIS and number of LIS in meanwhile.

Code

```
int findNumberOfLIS(vector<int>& nums) {
    int n = nums.size();
    if(n==0) return 0;
    vector<vector<int>> dp(2,vector<int>(n,1));
    int count = 0;
    int LIS = findLIS(nums, dp);
    //use one loop to count all LIS number
    for(int i=0;i<nums.size();++i){
        if(dp[0][i] == LIS)
            count +=dp[1][i];
    }
    return count;
}
private:
int findLIS(vector<int>& nums, vector<vector<int>>& dp){
    int res = 1;
    for(int i=1;i<nums.size(); ++i){
        for(int j= i-1;j>=0;--j){
            if(nums[i] > nums[j]){
                int temp = dp[0][j] + 1;
                if(temp > dp[0][i]) {
```

```
                    dp[0][i] = temp;
                    dp[1][i] = dp[1][j];
                }
                else if(temp == dp[0][i])
                    dp[1][i] += dp[1][j];
            }
        }
        res = max(res, dp[0][i]);
    }
    return res;
}
```

Note

in `findLIS()` function, we should set `res=1` because the shortest IS is 1. and also, we start from `i=1`, if there's only one number in nums, will not enter for loop, so return `res = 1`.

Summary

2D DP