LeetCode 300

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

Yifeng Zeng

Description

300. Longest Increasing Subsequence

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

For example,

Given [10, 9, 2, 5, 3, 7, 101, 18],

The longest increasing subsequence is [2, 3, 7, 101], therefore the length is 4. Note that there may be more than one LIS combination, it is only necessary for you to return the length.

Idea Report

We can have an array int[] f where f[i] means ending at i-th number, what is the longest increasing subsequence. So the base cases are f[i] = 1 because ending at each position i, at least we can get itself as a subsequence which length is 1. For the state transition, to get f[i], we need to see all the f[j] before f[i], if the number at that posiiton nums[j] is smaller than nums[i], then f[i] = f[j] + 1. We enumerate all f[j] to get the maximum f[i].
For example for our example input, we can have a matrix:

i	0	1	2	3	4	5	6	7
nums[i]	10	9	2	5	3	7	101	18
f[i]	1	1	1	2	2	3	4	4

To gte f[5], we need to see f[0], f[1], f[2], f[3], f[4]. If nums[5] > nums[2], nums[5] > nums[3], nums[5] > nums[4], we only calculate from these three indeces, because nums[0] or nums[1] will not construct an increasing subsequence with nums[5]. For nums[2], f[2] = 1, so f[5] can be 1 + 1 = 2. For nums[3], f[3] = 2, so f[5] can be 2 + 1 = 3. For nums[4], f[4] = 2, so f[5] can be 2 + 1 = 3. We get the maximum so f[5] = 3. We plus 1 because using the current f[5], the subsequence length increases 1. And at the end, we iterate the array f to get the maximum number and return. This takes O(n^2) time.

Code:

```
class Solution {
    public int lengthOfLIS(int[] nums) {
        if (nums == null || nums.length == 0) {
            return 0;
        }
        int[] f = new int[nums.length];
        Arrays.fill(f, 1);
        int max = 1;
        for (int i = 1; i < nums.length; i++) {
            for (int j = 0; j < i; j++) {
                if (nums[i] > nums[j]) {
                    f[i] = Math.max(f[i], f[j] + 1);
                    max = Math.max(max, f[i]);
                }
            }
        }

        return max;
    }
}
```

How do we speed up? We can try O(nlogn), then we may want to try binary search. We can somehow maintain a sorted array and we traversing a new number, we use binary search to see where this new number belongs, and update the sorted array in order to make up the longest subsequence.
Again we use the example input [10, 9, 2, 5, 3, 7, 101, 18]. Maximum length is 4.

1. 10 -> [10]
2. 9 -> [9]
3. 2 -> [2]
4. 5 -> [2, 5]
5. 3 -> [2, 3]

6. 7 -> [2, 3, 7]
7. 101 -> [2, 3, 7, 101]
8. 18 -> [2, 3, 7, 18]

We have a subsequence array int[] arr. For each traversing number nums[i], in int[] arr, we find the last number that is smaller than nums[i], we put it at the next index of that number, for example at step 5, we find out 2 is the last number that is smaller than 3, we put it at 5's location, because compares [2,3] and [2,5], 5 is no longer useful when calculating the longest subsequence. During the traversing, we record the longest subsequence length appeared and return. For each number in nums, we do a binary search and we have n numbers, so we use O(nlogn) time.

Code:

```
class Solution {
    public int lengthOfLIS(int[] nums) {
        if (nums == null || nums.length == 0) {
            return 0;
        }
        int[] arr = new int[nums.length];
        Arrays.fill(arr, Integer.MAX_VALUE);
        arr[0] = nums[0];
        int res = 1;
        int index = 0;
        for (int i = 1; i < nums.length; i++) {
            index = binarySearch(arr, nums[i]);
            arr[index] = nums[i];
            res = Math.max(res, index + 1);
        }

        return res;
    }

    // find the first number that is larger or equal to target
    private int binarySearch(int[] nums, int target) {
        int start = 0;
        int end = nums.length - 1;
        while (start + 1 < end) {
            int mid = (end - start) / 2 + start;
            if (nums[mid] >= target) {
                end = mid;
            } else {
                start = mid;
            }
        }

        if (nums[start] >= target) {
            return start;
        }
        return end;
    }
}
```

Summary

- Subsequence in array DP problem nomarlly need to fix one end, or choose one number as the end of the subsequence.
- Binary search.