cnblogs.com

[LeetCode 1713] Minimum Operations to Make a Subsequence - Review->Improve

Review->Improve 粉丝 - 5 关注 - 8

3-4 minutes

You are given an array `target` that consists of distinct integers and another integer array `arr` that can have duplicates.

In one operation, you can insert any integer at any position in `arr`. For example, if `arr = [1,4,1,2]`, you can add `3` in the middle and make it `[1,4,3,1,2]`. Note that you can insert the integer at the very beginning or end of the array.

Return *the minimum number of operations needed to make* `target` *a subsequence of* `arr`.

A subsequence of an array is a new array generated from the original array by deleting some elements (possibly none) without changing the remaining elements' relative order. For example, `[2,7,4]` is a subsequence of `[4,2,3,7,2,1,4]` (the underlined elements), while `[2,4,2]` is not.

Example 1:

```
arr
arr
```

```
arr
```

Example 2:

Input: target = [6,4,8,1,3,2],

```
arr
```

= [4,7,6,2,3,8,6,1]

Output: 3

Constraints:

- `1 <= target.length, arr.length <= 10^5`
- `1 <= target[i], arr[i] <= 10^9`
- `target` contains no duplicates.

We start by observing the follwing key points:

1. Numbers that do not exist in target array do not matter, they can be ignored;

2. Target array only has distinct integers, this means we can map each unique integer to its idx in target array;

3. For array arr, if we ignore numbers that do not exist in target but replace all the existing numbers with their idx in target, this problem becomes this: Given N unique increasing numbers from 0 to N - 1, (N is target array's length) and an array A that can only contains number from 0 to N - 1, find the minimum insertions needed such that A has an increasing subsequence from 0 to N - 1.

We can further reduce the above problem to: find the longest increasing subsequence in A, call it Length(LIS), N - Length(LIS) will be the minimum insertions needed.

```
class Solution {
    public int minOperations(int[] target, int[] arr) {
        Map<Integer, Integer> map = new HashMap<>();
        int n = target.length;
        for(int i = 0; i < n; i++) {
            map.put(target[i], i);
        }
        List<Integer> list = new ArrayList<>();
        for(int v : arr) {
            if(map.containsKey(v)) {
                list.add(map.get(v));
            }
        }
        return n - lengthOfLIS(list);
    }
    private int lengthOfLIS(List<Integer> list) {
        List<ArrayDeque<Integer>> qList = new ArrayList<>();
        for(int v : list) {
            int idx = binarySearch(qList, v);
            if(idx == qList.size()) {
                qList.add(new ArrayDeque<>());
            }
            ArrayDeque<Integer> q = qList.get(idx);
            q.addFirst(v);
        }
        return qList.size();
    }
    private int binarySearch(List<ArrayDeque<Integer>> qList, int v)
```

```
{
        if(qList.size() > 0) {
            int l = 0, r = qList.size() - 1;
            while(l < r) {
                int mid = l + (r - l) / 2;
                if(qList.get(mid).peekFirst() < v) {
                    l = mid + 1;
                }
                else {
                    r = mid;
                }
            }
            if(qList.get(l).peekFirst() >= v) {
                return l;
            }
        }
        return qList.size();
    }
}
```

Related Problems

[LeetCode 300] Longest Increasing Subsequence