Description | Editorial | Solutions (12.6K) | Submissions

26. Remove Duplicates from Sorted Array

Hint

Easy | 11.1K | 15K

Companies

Given an integer array `nums` sorted in **non-decreasing order**, remove the duplicates **in-place** such that each unique element appears only **once**. The **relative order** of the elements should be kept the **same**. Then return *the number of unique elements in* `nums`.

Consider the number of unique elements of `nums` to be `k`, to get accepted, you need to do the following things:

- Change the array `nums` such that the first `k` elements of `nums` contain the unique elements in the order they were present in `nums` initially. The remaining elements of `nums` are not important as well as the size of `nums`.
- Return `k`.

Custom Judge:

The judge will test your solution with the following code:

```
int[] nums = [...]; // Input array
int[] expectedNums = [...]; // The expected answer with
correct length

int k = removeDuplicates(nums); // Calls your implementation

assert k == expectedNums.length;
for (int i = 0; i < k; i++) {
    assert nums[i] == expectedNums[i];
}
```

If all assertions pass, then your solution will be **accepted**.

Example 1:

Java | Auto

```java
class Solution {
    public int removeDuplicates(int[] nums) {
        int i = 0;
        for (int j = 1; j < nums.length; j++) {
            if (nums[i] != nums[j]) {
                i++;
                nums[i] = nums[j];
            }
        }
        return i + 1;
    }
}
```

Testcase | Result

Accepted Runtime: 0 ms

Case 1 | Case 2

Input

nums =
[1,1,2]

Output

[1,2]

Expected

[1,2]

Console | Run | Submit

Description | Editorial | Solutions (1.9K) | Submissions

1376. Time Needed to Inform All Employees

Hint

Medium | 3.5K | 239

Companies

A company has `n` employees with a unique ID for each employee from `0` to `n - 1`. The head of the company is the one with `headID`.

Each employee has one direct manager given in the `manager` array where `manager[i]` is the direct manager of the `i-th` employee, `manager[headID] = -1`. Also, it is guaranteed that the subordination relationships have a tree structure.

The head of the company wants to inform all the company employees of an urgent piece of news. He will inform his direct subordinates, and they will inform their subordinates, and so on until all employees know about the urgent news.

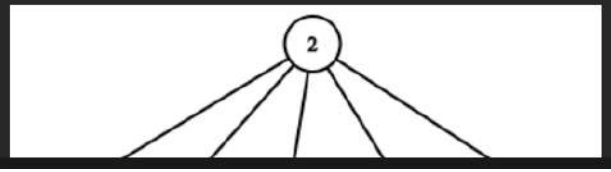
The `i-th` employee needs `informTime[i]` minutes to inform all of his direct subordinates (i.e., After informTime[i] minutes, all his direct subordinates can start spreading the news).

Return *the number of minutes* needed to inform all the employees about the urgent news.

Example 1:

```
Input: n = 1, headID = 0, manager = [-1], informTime = [0]
Output: 0
Explanation: The head of the company is the only employee in the company.
```

Example 2:



Java | Auto

```java
class Solution {
    private int dfs(int manager, int[] informTime, Map<Integer, List<Integer>> adjList) {
        int maxTime = 0;
        if (adjList.containsKey(manager)) {
            for (int subordinate : adjList.get(manager)) {
                maxTime = Math.max(maxTime, dfs(subordinate, informTime, adjList));
            }
        }
        return maxTime + informTime[manager];
    }

```

Testcase | Result

Accepted Runtime: 1 ms

Case 1 | Case 2

Input

n =
1

headID =
0

manager =
[-1]

Console | Run | Submit

Description | Editorial | Solutions (14.6K) | Submissions

35. Search Insert Position

Easy | 13.4K | 586 | Companies

Given a sorted array of distinct integers and a target value, return the index if the target is found. If not, return the index where it would be if it were inserted in order.

You must write an algorithm with `O(log n)` runtime complexity.

Example 1:

```
Input: nums = [1,3,5,6], target = 5
Output: 2
```

Example 2:

```
Input: nums = [1,3,5,6], target = 2
Output: 1
```

Example 3:

```
Input: nums = [1,3,5,6], target = 7
Output: 4
```

Constraints:

- `1 <= nums.length <= 10^4`
- `-10^4 <= nums[i] <= 10^4`
- `nums` contains **distinct** values sorted in **ascending** order.

Java | Auto

```java
class Solution {
    public int searchInsert(int[] nums, int target) {
        int start = 0;
        int end = nums.length-1;

        while (start <= end) {
            int mid = start + (end-start)/2;
            if (nums[mid] == target) return mid;
            else if (nums[mid] > target) end = mid-1;
            else start = mid+1;
        }

        return start;
```

Testcase | Result

Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3

Input

nums =
[1,3,5,6]

target =
5

Output

2

Console | Run | Submit

Description | Editorial | Solutions (12.9K) | Submissions

28. Find the Index of the First Occurrence in a String

Easy | 3.6K | 187

Companies

Given two strings `needle` and `haystack`, return the index of the first occurrence of `needle` in `haystack`, or `-1` if `needle` is not part of `haystack`.

Example 1:

```
Input: haystack = "sadbutsad", needle = "sad"
Output: 0
Explanation: "sad" occurs at index 0 and 6.
The first occurrence is at index 0, so we return 0.
```

Example 2:

```
Input: haystack = "leetcode", needle = "leeto"
Output: -1
Explanation: "leeto" did not occur in "leetcode", so we return -1.
```

Constraints:

- `1 <= haystack.length, needle.length <= 10^4`
- `haystack` and `needle` consist of only lowercase English characters.

Accepted 1.8M | Submissions 4.4M | Acceptance Rate 39.5%

Seen this question in a real interview before? 1/4

Java | Auto

```java
class Solution {
    public int strStr(String haystack, String needle) {
        int haylength=haystack.length();
        int needlelength=needle.length();
        if(haylength<needlelength)
            return -1;
        for(int i=0;i<=haystack.length()-needle.length();i++){
            int j=0;
            while(j<needle.length() && haystack.charAt(i+j)==needle.charAt(j))
                j++;
            if(j==needle.length()){
                return i;
```

Testcase | Result

Accepted Runtime: 0 ms

Case 1 | Case 2

Input

haystack =
"sadbutsad"

needle =
"sad"

Output

0

Console | Run | Submit

Description | Editorial | Solutions (10.8K) | Submissions

27. Remove Element

Hint

Easy 92 84

Companies

Given an integer array `nums` and an integer `val`, remove all occurrences of `val` in `nums` **in-place**. The order of the elements may be changed. Then return *the number of elements in `nums` which are not equal to `val`*.

Consider the number of elements in `nums` which are not equal to `val` be `k`, to get accepted, you need to do the following things:

- Change the array `nums` such that the first `k` elements of `nums` contain the elements which are not equal to `val`. The remaining elements of `nums` are not important as well as the size of `nums`.
- Return `k`.

Custom Judge:

The judge will test your solution with the following code:

```
int[] nums = [...]; // Input array
int val = ...; // Value to remove
int[] expectedNums = [...]; // The expected answer with
correct length.
                            // It is sorted with no values
equaling val.

int k = removeElement(nums, val); // Calls your implementation

assert k == expectedNums.length;
sort(nums, 0, k); // Sort the first k elements of nums
for (int i = 0; i < actualLength; i++) {
    assert nums[i] == expectedNums[i];
}
```

If all assertions pass, then your solution will be **accepted**.

Java | Auto

```java
class Solution {
    public int removeElement(int[] nums, int val) {
        int i = 0;
        for (int j = 0; j < nums.length; j++) {
            if (nums[j] != val) {
                int temp = nums[i];
                nums[i] = nums[j];
                nums[j] = temp;
                i++;
            }
        }
        return i;
    }
```

Testcase | Result

Accepted Runtime: 0 ms

Case 1 | Case 2

Input

nums =
[3,2,2,3]

val =
3

Output

[2,2]

Console | Run | Submit