80. Remove Duplicates from Sorted Array II

Medium Topics Companies

Given an integer array `nums` sorted in **non-decreasing order**, remove some duplicates **in-place** such that each unique element appea
most twice. The **relative order** of the elements should be kept the **same**.

Since it is impossible to change the length of the array in some languages, you must instead have the result be placed in the **first par**
array `nums`. More formally, if there are `k` elements after removing the duplicates, then the first `k` elements of `nums` should hold the
result. It does not matter what you leave beyond the first `k` elements.

Return `k` *after placing the final result in the first* `k` *slots of* `nums`.

Do **not** allocate extra space for another array. You must do this by **modifying the input array in-place** with O(1) extra memory.

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:

```
Input: nums = [1,1,1,2,2,3]
Output: 5, nums = [1,1,2,2,3,_]
Explanation: Your function should return k = 5, with the first five elements of nums being 1, 1,
and 3 respectively.
It does not matter what you leave beyond the returned k (hence they are underscores).
```

Example 2:

```
Input: nums = [0,0,1,1,1,1,2,3,3]
Output: 7, nums = [0,0,1,1,2,3,3,_,_]
```

Code

Java Auto

```java
class Solution {
    public int removeDuplicates(int[] nums) {
      if(nums.length<2) return nums.length;
      int cur=2;
      for(int i=2; i<nums.length; i++) {
        if(nums[i]==nums[cur-1] && nums[i]==nums[cur-2]) continue;
        nums[cur++]=nums[i];
      }
      return cur;
    }
}
```

Saved to local

Testcase Test Result

81. Search in Rotated Sorted Array II

Medium Topics Companies

There is an integer array `nums` sorted in non-decreasing order (not necessarily with **distinct** values).

Before being passed to your function, `nums` is **rotated** at an unknown pivot index `k` (`0 <= k < nums.length`) such that the resulting `[nums[k], nums[k+1], ..., nums[n-1], nums[0], nums[1], ..., nums[k-1]]` (**0-indexed**). For example, `[0,1,2,4,4,4,5,6,6,7` be rotated at pivot index `5` and become `[4,5,6,6,7,0,1,2,4,4]`.

Given the array `nums` **after** the rotation and an integer `target`, return `true` *if* `target` *is in* `nums`, *or* `false` *if it is not in* `nums`.

You must decrease the overall operation steps as much as possible.

Example 1:

```
Input: nums = [2,5,6,0,0,1,2], target = 0
Output: true
```

Example 2:

```
Input: nums = [2,5,6,0,0,1,2], target = 3
Output: false
```

Constraints:

- `1 <= nums.length <= 5000`
- `-10^4 <= nums[i] <= 10^4`
- `nums` is guaranteed to be rotated at some pivot.
- `-10^4 <= target <= 10^4`

Follow up: This problem is similar to Search in Rotated Sorted Array, but `nums` may contain **duplicates**. Would this affect the runtime complexity? How and why?

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

8K 80

Problem List Run Submit 0

Code

Java Auto

```java
class Solution {
    public boolean search(int[] nums, int target) {
      int left=0;
      int right=nums.length-1;
      while(left<=right) {

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

      return false;

    }
}
```

Saved to local

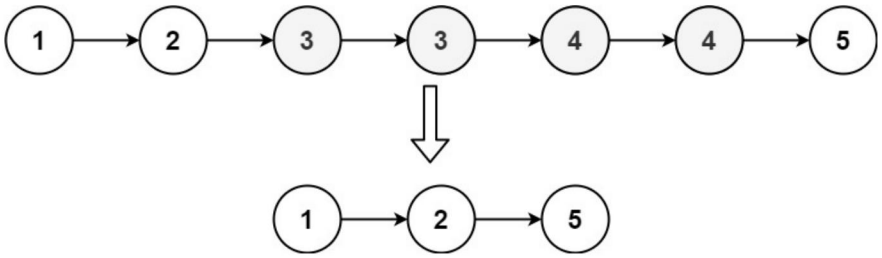
Testcase Test Result

82. Remove Duplicates from Sorted List II

Medium Topics Companies

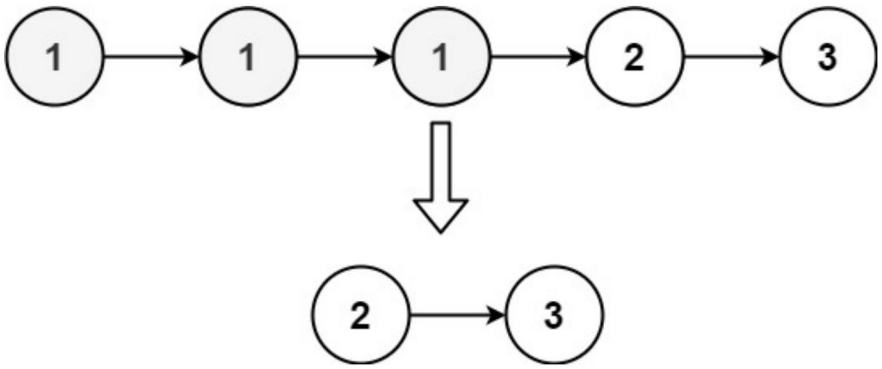
Given the `head` of a sorted linked list, *delete all nodes that have duplicate numbers, leaving only distinct numbers from the original list. the linked list **sorted** as well.*

Example 1:




```
Input: head = [1,2,3,3,4,4,5]
Output: [1,2,5]
```

Example 2:




```
Input: head = [1,1,1,2,3]
Output: [2,3]
```

Constraints:

- The number of nodes in the list is in the range `[0, 300]`.
- `-100 <= Node.val <= 100`

Code

Java
Auto

```java
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) { val = x; }
 * }
 */
class Solution {
    public ListNode deleteDuplicates(ListNode head) {
      ListNode tmp=new ListNode(0)    ;
      tmp.next=head;
      ListNode cur=tmp;

      while(cur.next!=null) {
        if(cur.next.next!=null && cur.next.val == cur.next.next.val) {
          //found dup
          ListNode dup=cur.next.next;
          int val = cur.next.val;
          while(dup.next!=null && dup.next.val==val)
            dup=dup.next;
          cur.next=dup.next;
        }else{
          cur=cur.next;
        }
      }
      return tmp.next;
    }
}
```

Saved to local

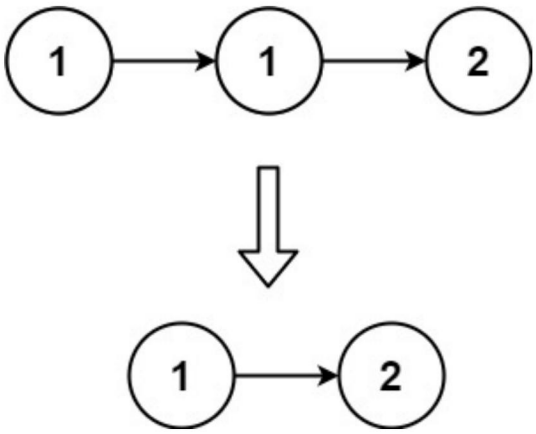
Testcase
Test Result

83. Remove Duplicates from Sorted List

Easy Topics Companies

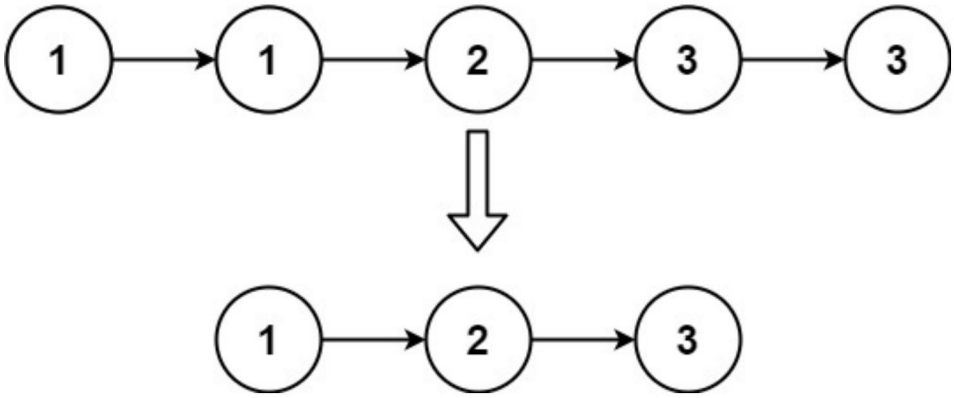
Given the `head` of a sorted linked list, *delete all duplicates such that each element appears only once*. Return *the linked list* ***sorted*** *as w*

Example 1:




```
Input: head = [1,1,2]
Output: [1,2]
```

Example 2:




```
Input: head = [1,1,2,3,3]
Output: [1,2,3]
```

Code

Java Auto

```java
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) { val = x; }
 * }
 */
class Solution {
    public ListNode deleteDuplicates(ListNode head) {
      if(head==null) return head;
      ListNode tmp=new ListNode(0);
      tmp.next=head;
      ListNode cur=head;
      ListNode pre=tmp;
      while(cur.next!=null) {
        if(cur.next.val == cur.val) {
          pre.next=cur.next;
        }else {
          pre=cur;

        }
        cur=cur.next;
      }
      return tmp.next;
    }
}
```

Saved to local

Testcase Test Result

Description Editorial Solutions Submissions

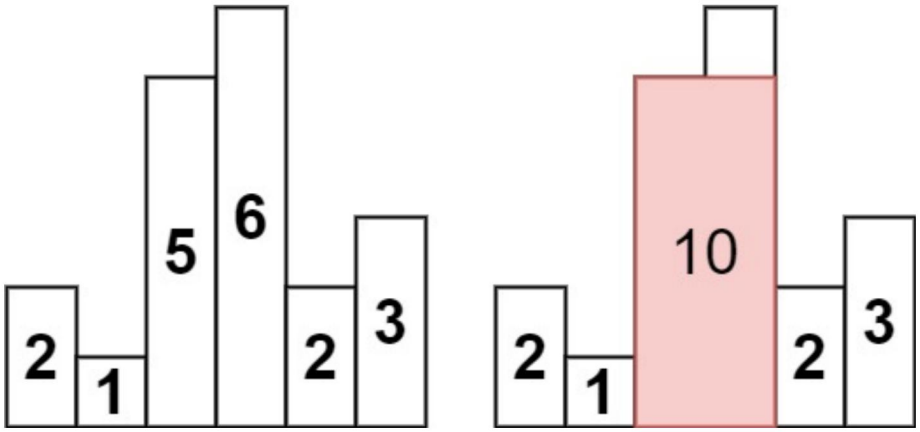
84. Largest Rectangle in Histogram

So

Hard Topics Companies

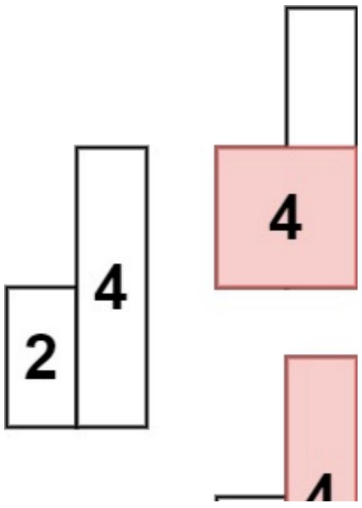
Given an array of integers `heights` representing the histogram's bar height where the width of each bar is `1`, return *the area of the lc rectangle in the histogram.*

Example 1:




```
Input: heights = [2,1,5,6,2,3]
Output: 10
Explanation: The above is a histogram where width of each bar is 1.
The largest rectangle is shown in the red area, which has an area = 10 units.
```

Example 2:



16.2K 80

Run Submit 0

Code

Java Auto

```java
class Solution {
    public int largestRectangleArea(int[] heights) {
      Stack<Integer> stack = new Stack<>();
      int ret=0;
      for(int i=0; i<=heights.length; i++) {
        int h=0;
        if(i!=heights.length) h=heights[i];
        //用单调递增stack，因为如果碰到下降，比如6,2,那么6不可能和右边高于2的形成valid rectangel,
        //此时可以计算6可能参与的rectangle，然后ignore (pop) 6.
        //继续向左直到低于2的1，因为1可以继续和2以后的形成rectangle,所以保留1在stack
        while(!stack.empty() && heights[stack.peek()]>=h) {
          int height = heights[stack.pop()];
          int width = i;
          if(!stack.empty()) width=i-1-stack.peek();
          int area=height*width;
          if(ret<area)
            ret=area;

        }
        stack.push(i);
       // System.out.println(""+i+" "+ ret);
      }
      return ret;
    }
}
```

Saved to local

Testcase Test Result

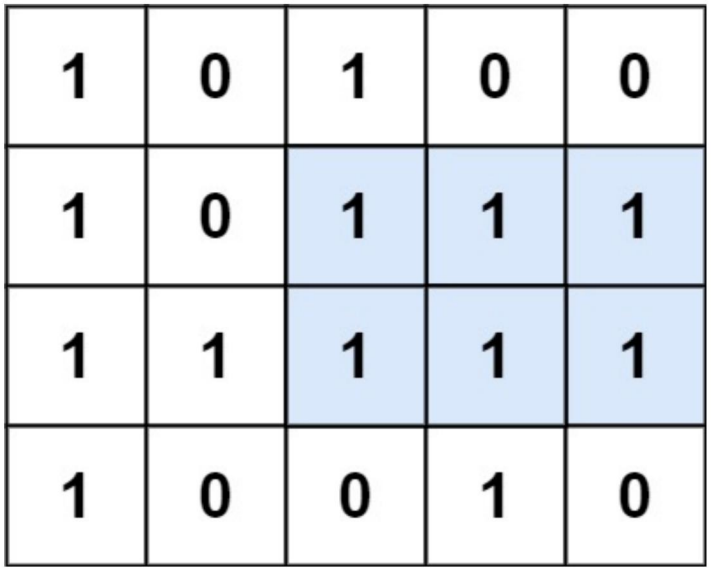
85. Maximal Rectangle

So

Hard Topics Companies

Given a `rows x cols` binary `matrix` filled with `0`'s and `1`'s, find the largest rectangle containing only `1`'s and return *its area*.

Example 1:




```
Input: matrix = [["1","0","1","0","0"],["1","0","1","1","1"],["1","1","1","1","1"],
["1","0","0","1","0"]]
Output: 6
Explanation: The maximal rectangle is shown in the above picture.
```

Example 2:

```
Input: matrix = [["0"]]
Output: 0
```

Example 3:

```
Input: matrix = [["1"]]
Output: 1
```

Constraints:

Code

Java Auto

```java
class Solution {

// The DP solution proceeds row by row, starting from the first row. Let the maximal rectangle area at row i and column
computed by [right(i,j) - left(i,j)]*height(i,j).

// All the 3 variables left, right, and height can be determined by the information from previous row, and also informat
from the current row. So it can be regarded as a DP solution. The transition equations are:

// left(i,j) = max(left(i-1,j), cur_left), cur_left can be determined from the current row

// right(i,j) = min(right(i-1,j), cur_right), cur_right can be determined from the current row

// height(i,j) = height(i-1,j) + 1, if matrix[i][j]=='1';

// height(i,j) = 0, if matrix[i][j]=='0'


    public int maximalRectangle(char[][] matrix) {

       if(matrix.length==0 || matrix[0].length==0) return 0;
      int m = matrix.length; int n = matrix[0].length;
      int[] left=new int[n]; int[] right=new int[n]; int[] height=new int[n];
      Arrays.fill(right, n);
      int ret=0;
      for(int i=0; i<m; i++ ) {
        int curleft=0; int curright=n;
        for(int j=0; j<n; j++) {
          if(matrix[i][j]=='0') {
            height[j]=0;
          }else {
            height[j]+=1;
          }
        }

        for(int j=0; j<n; j++) {
          if(matrix[i][j]=='0') {
            curleft=j+1;
            left[j]=0;
          }else {
            left[j]=Math.max(left[j], curleft);
          }
        }

        for(int j=n-1; j>=0; j--) {
          if(matrix[i][j]=='0' ) {
            curright=j;
            right[j]=n;
          }else {
            right[j]=Math.min(right[j], curright);
          }
        }
```

Saved to local

Testcase Test Result

Code

Java Auto

```java

// height(i,j) = 0, if matrix[i][j]=='0'


    public int maximalRectangle(char[][] matrix) {

       if(matrix.length==0 || matrix[0].length==0) return 0;
      int m = matrix.length; int n = matrix[0].length;
      int[] left=new int[n]; int[] right=new int[n]; int[] height=new int[n];
      Arrays.fill(right, n);
      int ret=0;
      for(int i=0; i<m; i++ ) {
        int curleft=0; int curright=n;
        for(int j=0; j<n; j++) {
          if(matrix[i][j]=='0') {
            height[j]=0;
          }else {
            height[j]+=1;
          }
        }

        for(int j=0; j<n; j++) {
          if(matrix[i][j]=='0') {
            curleft=j+1;
            left[j]=0;
          }else {
            left[j]=Math.max(left[j], curleft);
          }
        }

        for(int j=n-1; j>=0; j--) {
          if(matrix[i][j]=='0' ) {
            curright=j;
            right[j]=n;
          }else {
            right[j]=Math.min(right[j], curright);
          }
        }
        //System.out.println(Arrays.toString(left));
        //System.out.println(Arrays.toString(right));
        //System.out.println(Arrays.toString(height));

        for(int j=0; j<n; j++) {
          ret=Math.max(ret, (right[j]-left[j])*height[j] );
        }

      }

      return ret;
    }
}
```

Saved to local

Testcase Test Result

86. Partition List

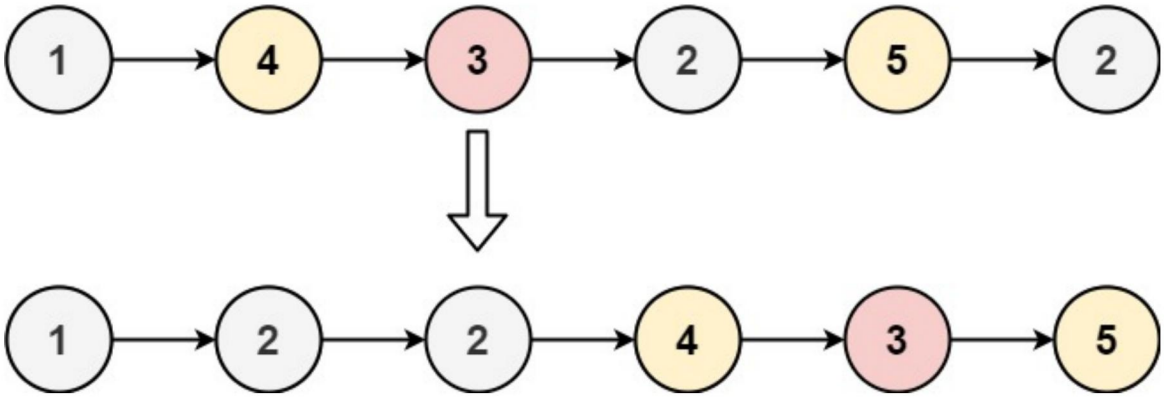
So

Medium Topics Companies

Given the `head` of a linked list and a value `x`, partition it such that all nodes **less than** `x` come before nodes **greater than or equal** t

You should **preserve** the original relative order of the nodes in each of the two partitions.

Example 1:




```
Input: head = [1,4,3,2,5,2], x = 3
Output: [1,2,2,4,3,5]
```

Example 2:

```
Input: head = [2,1], x = 2
Output: [1,2]
```

Constraints:

- The number of nodes in the list is in the range `[0, 200]`.
- `-100 <= Node.val <= 100`
- `-200 <= x <= 200`

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

7K 74

Code

Java Auto

```java
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode() {}
 *     ListNode(int val) { this.val = val; }
 *     ListNode(int val, ListNode next) { this.val = val; this.next = next; }
 * }
 */
class Solution {
    public ListNode partition(ListNode head, int x) {
      ListNode dummy1 = new ListNode(0), dummy2 = new ListNode(0);  //dummy heads of the 1st and 2nd queues
      ListNode curr1 = dummy1, curr2 = dummy2;      //current tails of the two queues;
      while (head!=null){
          if (head.val<x) {
              curr1.next = head;
              curr1 = head;
          }else {
              curr2.next = head;
              curr2 = head;
          }
          head = head.next;
      }
      curr2.next = null;          //important! avoid cycle in linked list. otherwise u will get TLE.
      curr1.next = dummy2.next;
      return dummy1.next;
    }
}
```

Saved to local

Testcase Test Result

Description Editorial Solutions Submissions

87. Scramble String

Hard Topics Companies

We can scramble a string s to get a string t using the following algorithm:

1. If the length of the string is 1, stop.

   If the length of the string is > 1, do the following:
   - Split the string into two non-empty substrings at a random index, i.e., if the string is `s`, divide it to `x` and `y` where `s = x + y`.
   - **Randomly** decide to swap the two substrings or to keep them in the same order. i.e., after this step, `s` may become `s = x + y` or `s = y + x`.
2. Apply step 1 recursively on each of the two substrings `x` and `y`.

Given two strings `s1` and `s2` of **the same length**, return `true` if `s2` is a scrambled string of `s1`, otherwise, return `false`.

Example 1:

```
Input: s1 = "great", s2 = "rgeat"
Output: true
Explanation: One possible scenario applied on s1 is:
"great" --> "gr/eat" // divide at random index.
"gr/eat" --> "gr/eat" // random decision is not to swap the two substrings and keep them in orde
"gr/eat" --> "g/r / e/at" // apply the same algorithm recursively on both substrings. divide at
index each of them.
"g/r / e/at" --> "r/g / e/at" // random decision was to swap the first substring and to keep the
second substring in the same order.
"r/g / e/at" --> "r/g / e/ a/t" // again apply the algorithm recursively, divide "at" to "a/t".
"r/g / e/ a/t" --> "r/g / e/ a/t" // random decision is to keep both substrings in the same orde
The algorithm stops now, and the result string is "rgeat" which is s2.
As one possible scenario led s1 to be scrambled to s2, we return true.
```

Example 2:

```
Input: s1 = "abcde", s2 = "caebd"
Output: false
```

Example 3:

```
Input: s1 = "a", s2 = "a"
Output: true
```

Code

Java Auto

```java
class Solution {
    public boolean isScramble(String s1, String s2) {
      if (s1.length() != s2.length()) return false;
      if (s1.equals(s2)) return true;
      int[] cnt = new int[26];
      for (int i = 0; i < s1.length(); i++) {
        cnt[s1.charAt(i)-'a']++;
        cnt[s2.charAt(i)-'a']--;
      }
      for (int i = 0; i < cnt.length; i++) {
        if (cnt[i] != 0 ) return false;
      }
      int n = s1.length();
      for (int i = 1; i < s1.length(); i++) {
        if (isScramble(s1.substring(0, i), s2.substring(0, i)) &&
            isScramble(s1.substring(i), s2.substring(i)) ) return true;
        if (isScramble(s1.substring(0, i), s2.substring(n-i)) &&
            isScramble(s1.substring(i), s2.substring(0, n-i)) ) return true;

      }
      return false;
    }
}
```

Saved to local

Testcase Test Result

88. Merge Sorted Array

Easy Topics Companies Hint

You are given two integer arrays `nums1` and `nums2`, sorted in **non-decreasing order**, and two integers `m` and `n`, representing the nu elements in `nums1` and `nums2` respectively.

Merge `nums1` and `nums2` into a single array sorted in **non-decreasing order**.

The final sorted array should not be returned by the function, but instead be *stored inside the array* `nums1`. To accommodate this, `num` length of `m + n`, where the first `m` elements denote the elements that should be merged, and the last `n` elements are set to `0` and s ignored. `nums2` has a length of `n`.

Example 1:

```
Input: nums1 = [1,2,3,0,0,0], m = 3, nums2 = [2,5,6], n = 3
Output: [1,2,2,3,5,6]
Explanation: The arrays we are merging are [1,2,3] and [2,5,6].
The result of the merge is [1,2,2,3,5,6] with the underlined elements coming from nums1.
```

Example 2:

```
Input: nums1 = [1], m = 1, nums2 = [], n = 0
Output: [1]
Explanation: The arrays we are merging are [1] and [].
The result of the merge is [1].
```

Example 3:

```
Input: nums1 = [0], m = 0, nums2 = [1], n = 1
Output: [1]
Explanation: The arrays we are merging are [] and [1].
The result of the merge is [1].
Note that because m = 0, there are no elements in nums1. The 0 is only there to ensure the merge
result can fit in nums1.
```

Constraints:

- `nums1.length == m + n`
- `nums2.length == n`

Problem List Run Submit 0

Code

Java Auto

```java
class Solution {
    public void merge(int[] nums1, int m, int[] nums2, int n) {

      int i=m-1; int j=n-1; int k=m+n-1;

      //found the bigger one move to the end of nums1
      while(k>=0) {
        if(i<0 || i>=0 && j>=0 && nums1[i] < nums2[j]) {
            nums1[k--]=nums2[j--];

        } else {
            nums1[k--]=nums1[i--];
        }

      }

    }
}
```

Saved to local

Testcase Test Result

89. Gray Code

Medium Topics Companies

An **n-bit gray code sequence** is a sequence of 2^n integers where:

- Every integer is in the **inclusive** range $[0, 2^n - 1]$,
- The first integer is 0,
- An integer appears **no more than once** in the sequence,
- The binary representation of every pair of **adjacent** integers differs by **exactly one bit**, and
- The binary representation of the **first** and **last** integers differs by **exactly one bit**.

Given an integer n, return *any valid* ***n-bit gray code sequence***.

Example 1:

```
Input: n = 2
Output: [0,1,3,2]
Explanation:
The binary representation of [0,1,3,2] is [00,01,11,10].
- 00 and 01 differ by one bit
- 01 and 11 differ by one bit
- 11 and 10 differ by one bit
- 10 and 00 differ by one bit
[0,2,3,1] is also a valid gray code sequence, whose binary representation is [00,10,11,01].
- 00 and 10 differ by one bit
- 10 and 11 differ by one bit
- 11 and 01 differ by one bit
- 01 and 00 differ by one bit
```

Example 2:

```
Input: n = 1
Output: [0,1]
```

Constraints:

- $1 <= n <= 16$

Code

Java Auto

```java
class Solution {
    public List<Integer> grayCode(int n) {
      List<Integer> ret=new ArrayList<>()    ;
      ret.add(0);
      for(int i=0; i<n; i++) {
        int sz=ret.size();
        for(int j=sz-1; j>=0; j--) {
          ret.add( ret.get(j) | 1<<i);
        }
      }
      return ret;
    }
}



    // vector<int> grayCode(int n) {
    //     vector<int> res;
    //     unordered_set<int> s;
    //     helper(n, s, 0, res);
    //     return res;
    // }
    // void helper(int n, unordered_set<int>& s, int out, vector<int>& res) {
    //     if (!s.count(out)) {
    //         s.insert(out);
    //         res.push_back(out);
    //     }
    //     for (int i = 0; i < n; ++i) {
    //         int t = out;
    //         if ((t & (1 << i)) == 0) t |= (1 << i);
    //         else t &= ~(1 << i);
    //         if (s.count(t)) continue;
    //         helper(n, s, t, res);
    //         break;
    //     }
    // }
```

Saved to local

Testcase Test Result

90. Subsets II

So

Medium Topics Companies

Given an integer array `nums` that may contain duplicates, return *all possible subsets (the power set)*.

The solution set **must not** contain duplicate subsets. Return the solution in **any order**.

Example 1:

```
Input: nums = [1,2,2]
Output: [[],[1],[1,2],[1,2,2],[2],[2,2]]
```

Example 2:

```
Input: nums = [0]
Output: [[],[0]]
```

Constraints:

- `1 <= nums.length <= 10`
- `-10 <= nums[i] <= 10`

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

Yes No

Accepted 815.9K Submissions 1.4M Acceptance Rate 56.7%

Topics

Companies

Similar Questions

Discussion (40)

9.2K 40

Code

Java Auto

```java
class Solution {

    public List<List<Integer>> sol_1(int[] nums) {
      List<List<Integer>> ret = new ArrayList<>();
      Arrays.sort(nums);
      ret.add(new ArrayList<Integer>());
      int size=1;
      int last=nums[0];
      for(int i=0;  i<nums.length; i++) {
        if(nums[i]!=last) {
          last=nums[i];
          size=ret.size();
        }

        int n=ret.size();
        for(int j=n-size; j<n; j++) {
          ret.add(new ArrayList<Integer>(ret.get(j)));
          ret.get(ret.size()-1).add(nums[i]);
        }
      }

      return ret;
    }


    public void helper(int[] nums, List<List<Integer>> ret, List<Integer> curList, int start ) {
      ret.add(new ArrayList<Integer>(curList));
      for(int i=start; i<nums.length; i++ ){
        if(i>start && nums[i]==nums[i-1])
          continue;
        curList.add(nums[i]);
        helper(nums, ret, curList, i+1);
        curList.remove(curList.size()-1);
      }
    }
    public List<List<Integer>> sol_2(int[] nums) {
      List<List<Integer>> ret = new ArrayList<>();
      Arrays.sort(nums);
      helper(nums, ret, new ArrayList<Integer>(), 0);
      return ret;
    }


    public List<List<Integer>> subsetsWithDup(int[] nums) {
      return(sol_1(nums))  ;
      //return(sol_2(nums))  ;
    }
}
```

Saved to local

Testcase Test Result

91. Decode Ways

So

Medium Topics Companies

A message containing letters from `A-Z` can be **encoded** into numbers using the following mapping:

```
'A' -> "1"
'B' -> "2"
...
'Z' -> "26"
```

To **decode** an encoded message, all the digits must be grouped then mapped back into letters using the reverse of the mapping abov
(there may be multiple ways). For example, `"11106"` can be mapped into:

- `"AAJF"` with the grouping `(1 1 10 6)`
- `"KJF"` with the grouping `(11 10 6)`

Note that the grouping `(1 11 06)` is invalid because `"06"` cannot be mapped into `'F'` since `"6"` is different from `"06"`.

Given a string `s` containing only digits, return *the **number** of ways to **decode** it*.

The test cases are generated so that the answer fits in a **32-bit** integer.

Example 1:

```
Input: s = "12"
Output: 2
Explanation: "12" could be decoded as "AB" (1 2) or "L" (12).
```

Example 2:

```
Input: s = "226"
Output: 3
Explanation: "226" could be decoded as "BZ" (2 26), "VF" (22 6), or "BBF" (2 2 6).
```

Example 3:

```
Input: s = "06"
Output: 0
Explanation: "06" cannot be mapped to "F" because of the leading zero ("6" is different from "06
```

Code

Java Auto

```java
class Solution {
    public int numDecodings(String s) {
      int n =s.length();

      if(n>0 && s.charAt(0)=='0') return 0;
      int[] dp=new int[n+1];
      dp[0]=1;      dp[1]=1;
      for(int i=1; i<n; i++) {
        if(s.charAt(i)=='0') {
          if(s.charAt(i-1)!='1' && s.charAt(i-1)!='2') return 0;
          dp[i+1]=dp[i-1];
          dp[i]=dp[i-1];
        }else if(s.charAt(i-1)=='1' || (s.charAt(i-1)=='2' && s.charAt(i)>='0' && s.charAt(i)<='6') ) {
          dp[i+1]=dp[i]+dp[i-1];
        }else
          dp[i+1]=dp[i];
      }
      //System.out.println(Arrays.toString(dp));
      return dp[n];
    }
}
```

Saved to local

Testcase Test Result

Description Editorial Solutions Submissions

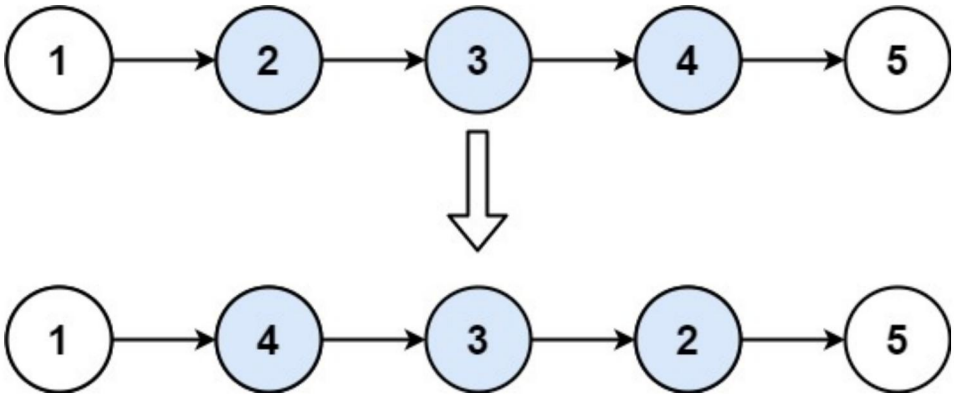
92. Reverse Linked List II

So

Medium Topics Companies

Given the `head` of a singly linked list and two integers `left` and `right` where `left <= right`, reverse the nodes of the list from po `left` to position `right`, and return *the reversed list*.

Example 1:




```
Input: head = [1,2,3,4,5], left = 2, right = 4
Output: [1,4,3,2,5]
```

Example 2:

```
Input: head = [5], left = 1, right = 1
Output: [5]
```

Constraints:

- The number of nodes in the list is `n`.
- `1 <= n <= 500`
- `-500 <= Node.val <= 500`
- `1 <= left <= right <= n`

Follow up: Could you do it in one pass?

11K 80

Code

Java Auto

```java
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode(int x) { val = x; }
 * }
 */
class Solution {
    public ListNode reverseBetween(ListNode head, int m, int n) {
      ListNode tmp=new ListNode(0);
      tmp.next=head;
      ListNode pre=tmp;
      for(int i=0; i<m-1; i++) pre=pre.next;
      ListNode cur=pre.next;
      ListNode nodem=cur;
      ListNode next=cur.next;
      for(int i=m; i<n; i++) {
        pre.next=next;
        next=next.next;
        pre.next.next=cur;
        cur=pre.next;
      }
      nodem.next=next;
      return tmp.next;

    }
}
```

Saved to local

Testcase Test Result

93. Restore IP Addresses

So

Medium Topics Companies

A **valid IP address** consists of exactly four integers separated by single dots. Each integer is between `0` and `255` (**inclusive**) and canr have leading zeros.

- For example, `"0.1.2.201"` and `"192.168.1.1"` are **valid** IP addresses, but `"0.011.255.245"`, `"192.168.1.312"` and `"192.168@1.1"` are **invalid** IP addresses.

Given a string `s` containing only digits, return *all possible valid IP addresses that can be formed by inserting dots into* `s`. You are **not** al to reorder or remove any digits in `s`. You may return the valid IP addresses in **any** order.

Example 1:

```
Input: s = "25525511135"
Output: ["255.255.11.135","255.255.111.35"]
```

Example 2:

```
Input: s = "0000"
Output: ["0.0.0.0"]
```

Example 3:

```
Input: s = "101023"
Output: ["1.0.10.23","1.0.102.3","10.1.0.23","10.10.2.3","101.0.2.3"]
```

Constraints:

- `1 <= s.length <= 20`
- `s` consists of digits only.

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

Yes No

Accepted 428.7K Submissions 878.4K Acceptance Rate 48.8%

5K 59

Code

Java Auto

```java
class Solution {
    public List<String> restoreIpAddresses(String s) {
      List<String> ret=new ArrayList<>();
      for(int i=1;i<=3;i++)
        for(int j=1;j<=3;j++)
          for(int k=1;k<=3;k++) {
            int l=s.length()-i-j-k;
            if(l<=0 || l>3) continue;
            int n1=Integer.parseInt(s.substring(0,i));
            if(n1>255) continue;
            int n2=Integer.parseInt(s.substring(i,i+j));
            if(n2>255) continue;
            int n3=Integer.parseInt(s.substring(i+j,i+j+k));
            if(n3>255) continue;
            int n4=Integer.parseInt(s.substring(i+j+k));
            if(n4>255) continue;

            StringBuilder sb=new StringBuilder();
            sb.append(n1); sb.append('.');
            sb.append(n2); sb.append('.');
            sb.append(n3); sb.append('.');
            sb.append(n4);
            if(sb.length()==s.length()+3)
              ret.add(sb.toString());
          }
      return ret;
    }
}
```

Saved to local

Testcase Test Result

Description Editorial Solutions Submissions

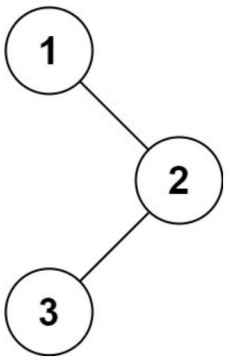
94. Binary Tree Inorder Traversal

So

Easy Topics Companies

Given the `root` of a binary tree, return *the inorder traversal of its nodes' values*.

Example 1:




```
Input: root = [1,null,2,3]
Output: [1,3,2]
```

Example 2:

```
Input: root = []
Output: []
```

Example 3:

```
Input: root = [1]
Output: [1]
```

Constraints:

- The number of nodes in the tree is in the range `[0, 100]`.
- `-100 <= Node.val <= 100`

Follow up: Recursive solution is trivial, could you do it iteratively?

12.6K 72

Code

Java Auto

```java
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
class Solution {
    public void recur(List<Integer> ret, TreeNode root) {
      if(root==null) return;
      recur(ret, root.left);
      ret.add(root.val);
      recur(ret, root.right);
    }

    public List<Integer> recursol(TreeNode root) {
      List<Integer> ret = new ArrayList<>();
      recur(ret, root);
      return ret;

    }

    public List<Integer> loopsol(TreeNode root) {
      List<Integer> ret = new ArrayList<>();
      Stack<TreeNode> stack=new Stack<>();
      while(!stack.empty() || root!=null) {
        if(root!=null) {
          stack.push(root);
          root=root.left;
        }else {
          root = stack.pop();
          ret.add(root.val);
          root=root.right;
        }
      }

      return ret;
    }


// 1. Initialize current as root
// 2. While current is not NULL
//    If the current does not have left child
//       a) Print current's data
//       b) Go to the right, i.e., current = current->right
//    Else
//       a) Find rightmost node in current left subtree OR
//              node whose right child == current.
//          If we found right child == current
```

Saved to local

Testcase Test Result

Code

Java Auto

```
38
39        return ret;
40      }
41
42
43  // 1. Initialize current as root
44  // 2. While current is not NULL
45  //    If the current does not have left child
46  //       a) Print current's data
47  //       b) Go to the right, i.e., current = current->right
48  //    Else
49  //       a) Find rightmost node in current left subtree OR
50  //              node whose right child == current.
51  //          If we found right child == current
52  //              a) Update the right child as NULL of that node whose right child is current
53  //              b) Print current's data
54  //              c) Go to the right, i.e. current = current->right
55  //          Else
56  //              a) Make current as the right child of that rightmost
57  //                 node we found; and
58  //              b) Go to this left child, i.e., current = current->left
59
60      public List<Integer> morrissol(TreeNode root) {
61        List<Integer> ret=new ArrayList<Integer>();
62        while(root!=null) {
63          if(root.left!=null) {
64            TreeNode tmp=root.left;
65            while(tmp.right!=null && tmp.right!=root) tmp=tmp.right;
66            if(tmp.right!=root) {
67              //System.out.println("++ "+tmp.val +" "+root.val);
68              tmp.right=root;
69              root=root.left;
70            }else {
71              //System.out.println("-- "+tmp.val +" "+root.val);
72              ret.add(root.val);
73              root=root.right;
74              tmp.right=null;
75            }
76          }else {
77            ret.add(root.val);
78            root=root.right;
79          }
80        }
81        return ret;
82      }
83
84      public List<Integer> inorderTraversal(TreeNode root) {
85        //return recursol(root);
86        //return loopsol(root);
87        return morrissol(root);
88      }
```

Saved to local

Testcase Test Result

Description Editorial Solutions Submissions

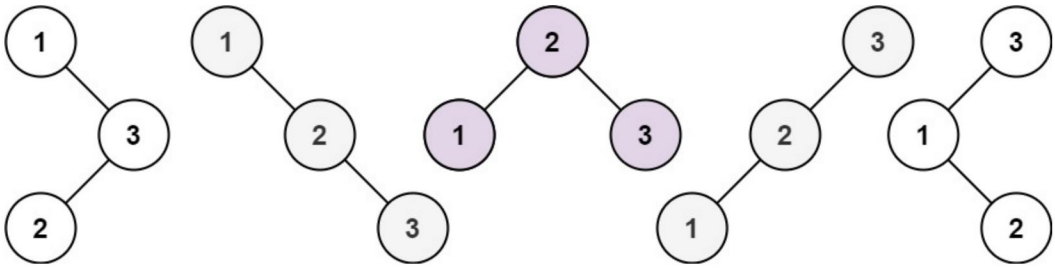
95. Unique Binary Search Trees II

So

Medium Topics Companies

Given an integer `n`, return *all the structurally unique **BST**'s (binary search trees), which has exactly* `n` *nodes of unique values from* `1` *to* Return the answer in **any order**.

Example 1:




```
Input: n = 3
Output: [[1,null,2,null,3],[1,null,3,2],[2,1,3],[3,1,null,null,2],[3,2,null,1]]
```

Example 2:

```
Input: n = 1
Output: [[1]]
```

Constraints:

- `1 <= n <= 8`

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

Yes No

Accepted 427.1K Submissions 752.8K Acceptance Rate 56.7%

Topics

Companies

7.3K 51

Code

Java
Auto

```java
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
class Solution {
    public List<TreeNode> gen(int start, int end) {
      List<TreeNode> ret=new ArrayList<>();

      if(start>end) {
        ret.add(null);
        return ret;
      }
      if(start==end) {
        ret.add(new TreeNode(start));
        return ret;
      }

      for(int i=start;i<=end;i++) {
        List<TreeNode> left,right;
        left=gen(start,i-1);
        right=gen(i+1,end);
        for(TreeNode lnode : left) {
          for(TreeNode rnode : right) {
            TreeNode root=new TreeNode(i);
            root.left=lnode;
            root.right=rnode;
            ret.add(root);
          }
        }

      }

      return ret;
    }
    public List<TreeNode> generateTrees(int n) {
      if(n==0) return new ArrayList<TreeNode>();
      return gen(1,n);
    }

}
```

Saved to local

Testcase
Test Result

Description Editorial Solutions Submissions

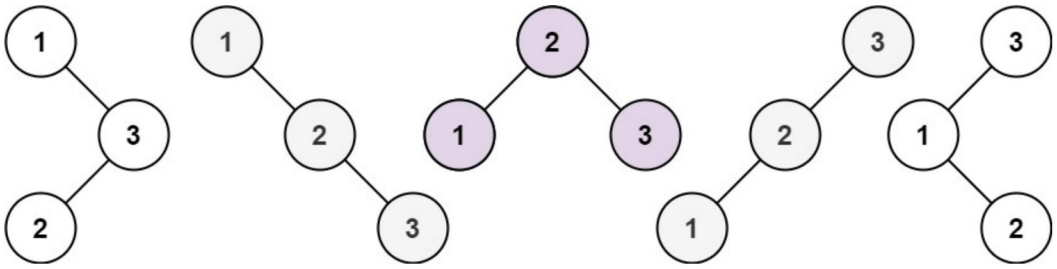
96. Unique Binary Search Trees

So

Medium Topics Companies

Given an integer `n`, return *the number of structurally unique* ***BST****'s (binary search trees) which has exactly* `n` *nodes of unique values fro* *to* `n`.

Example 1:




```
Input: n = 3
Output: 5
```

Example 2:

```
Input: n = 1
Output: 1
```

Constraints:

- `1 <= n <= 19`

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

Yes No

Accepted 622.3K Submissions 1M Acceptance Rate 60.6%

Topics

Companies

10K 28

Problem List Run Submit 0

Code

Java Auto

```java
class Solution {
    public int numTrees(int n) {
      int[] dp = new int[n+1];
      dp[0]=1;
      dp[1]=1;
      for(int i=2;i<=n;i++){
        for(int j=0; j<i; j++)
          dp[i]+=dp[j]*dp[i-j-1];
      }
      return dp[n];
    }
}
```

Saved to local

Testcase Test Result

97. Interleaving String

So

Medium Topics Companies

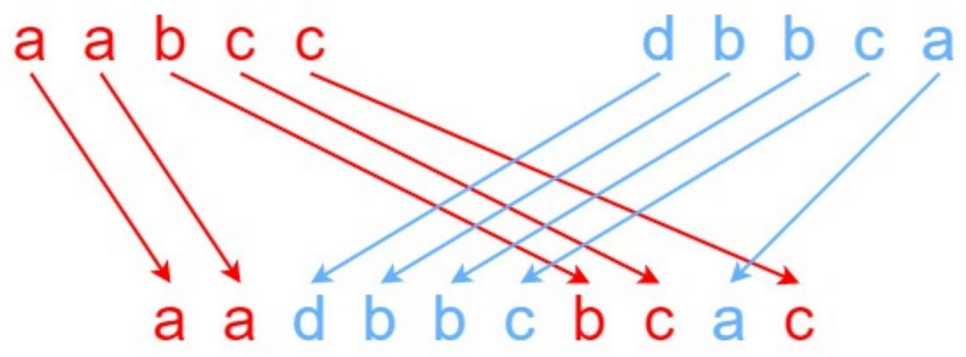
Given strings `s1`, `s2`, and `s3`, find whether `s3` is formed by an **interleaving** of `s1` and `s2`.

An **interleaving** of two strings `s` and `t` is a configuration where `s` and `t` are divided into `n` and `m` substrings respectively, such tha

- $s = s_1 + s_2 + ... + s_n$
- $t = t_1 + t_2 + ... + t_m$
- $|n - m| <= 1$
- The **interleaving** is $s_1 + t_1 + s_2 + t_2 + s_3 + t_3 + ...$ or $t_1 + s_1 + t_2 + s_2 + t_3 + s_3 + ...$

Note: `a + b` is the concatenation of strings `a` and `b`.

Example 1:




```
Input: s1 = "aabcc", s2 = "dbbca", s3 = "aadbbcbcac"
Output: true
Explanation: One way to obtain s3 is:
Split s1 into s1 = "aa" + "bc" + "c", and s2 into s2 = "dbbc" + "a".
Interleaving the two splits, we get "aa" + "dbbc" + "bc" + "a" + "c" = "aadbbcbcac".
Since s3 can be obtained by interleaving s1 and s2, we return true.
```

Example 2:

```
Input: s1 = "aabcc", s2 = "dbbca", s3 = "aadbbbaccc"
Output: false
Explanation: Notice how it is impossible to interleave s2 with any other string to obtain s3.
```

Code

Java Auto

```java
class Solution {
    public boolean isInterleave(String s1, String s2, String s3) {
      int n1=s1.length(); int n2=s2.length(); int n3=s3.length();
      if(n3!=n1+n2) return false;
      boolean[][] dp=new boolean[n1+1][n2+1];
      dp[0][0]=true;
      for(int i=1;i<n2+1;i++) {
        if(s2.charAt(i-1)==s3.charAt(i-1))
          dp[0][i]=true;
        else
          break;
      }
      for(int i=1;i<n1+1;i++) {
        if(s1.charAt(i-1)==s3.charAt(i-1))
          dp[i][0]=true;
        else break;
      }
      for(int i=1;i<n1+1;i++) {
        for(int j=1;j<n2+1;j++){
          if(dp[i-1][j] && s1.charAt(i-1)==s3.charAt(i-1+j)  ||
             dp[i][j-1] && s2.charAt(j-1)==s3.charAt(i-1+j) ) {
            dp[i][j]=true;
          }
        }
      }
      return dp[n1][n2];
    }
}
```

Saved to local

Testcase Test Result

98. Validate Binary Search Tree

So

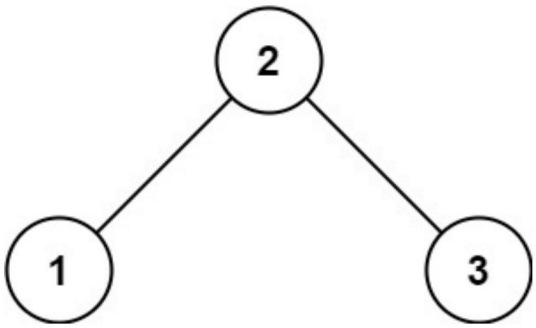
Medium Topics Companies

Given the `root` of a binary tree, *determine if it is a valid binary search tree (BST)*.

A **valid BST** is defined as follows:

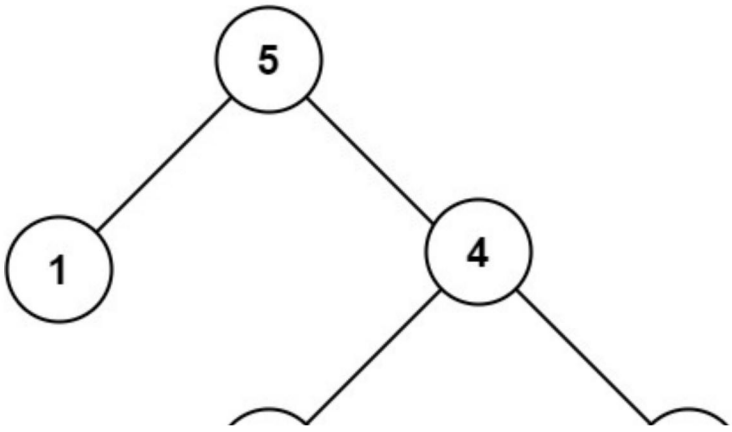
- The left subtree of a node contains only nodes with keys **less than** the node's key.
- The right subtree of a node contains only nodes with keys **greater than** the node's key.
- Both the left and right subtrees must also be binary search trees.

Example 1:




```
Input: root = [2,1,3]
Output: true
```

Example 2:



Code

Java Auto

```java
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
class Solution {
    public boolean isValidBST(TreeNode root) {
      return isValid(root, Long.MIN_VALUE, Long.MAX_VALUE)    ;
    }
    public boolean isValid(TreeNode root, long min, long max) {
      if(root==null) return true;

      if(root.val<= min || root.val>=max) return false;
      return isValid(root.left, min, root.val) && isValid(root.right, root.val, max);

    }

}
```

Saved to local

Testcase Test Result

Description Editorial Solutions Submissions

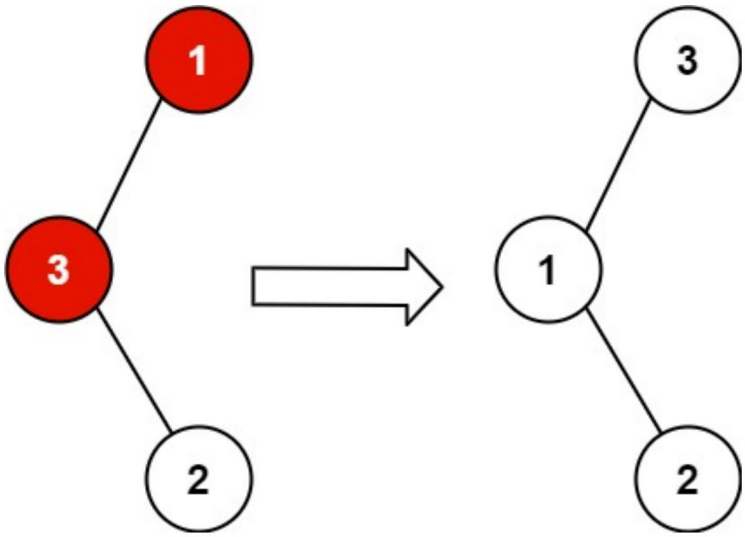
99. Recover Binary Search Tree

So

Medium Topics Companies

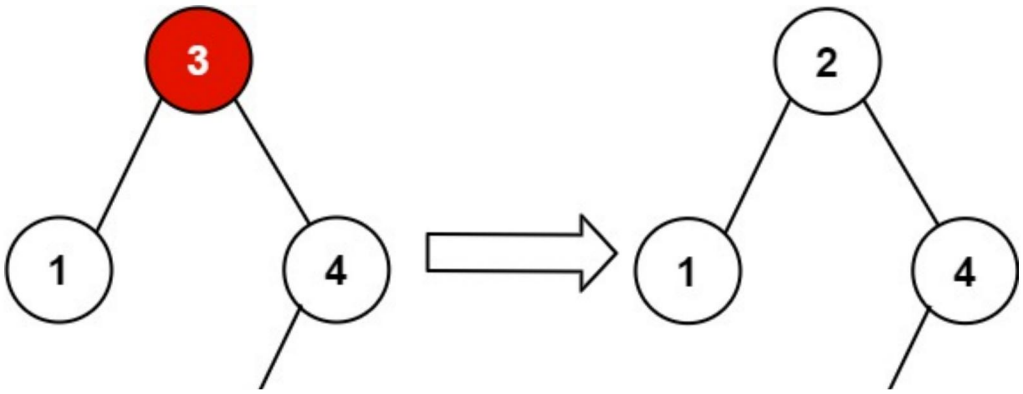
You are given the `root` of a binary search tree (BST), where the values of **exactly** two nodes of the tree were swapped by mistake. *Rec*
the tree without changing its structure.

Example 1:




```
Input: root = [1,3,null,null,2]
Output: [3,1,null,null,2]
Explanation: 3 cannot be a left child of 1 because 3 > 1. Swapping 1 and 3 makes the BST valid.
```

Example 2:



Code

Java Auto

```java
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode(int x) { val = x; }
 * }
 */
class Solution {

    TreeNode firstElement = null;
    TreeNode secondElement = null;
    // The reason for this initialization is to avoid null pointer exception in the first comparison when prevElement h
been initialized
    TreeNode prevElement = null;

    public void recoverTree(TreeNode root) {

        // In order traversal to find the two elements
        traverse(root);

        // Swap the values of the two nodes
        int temp = firstElement.val;
        firstElement.val = secondElement.val;
        secondElement.val = temp;
    }

    private void traverse(TreeNode root) {

        if (root == null)
            return;

        traverse(root.left);

        if(prevElement == null) {
          prevElement = root;
        } else {

            if (prevElement.val > root.val) {
              if (firstElement==null) firstElement=prevElement;
              secondElement=root;


            }
//           // Start of "do some business",
//           // If first element has not been found, assign it to prevElement (refer to 6 in the example above)
//           if (firstElement == null && prevElement.val >= root.val) {
//               firstElement = prevElement;
//           }

```

Saving...

Testcase Test Result

Code

Java Auto

```java
    private void traverse(TreeNode root) {

        if (root == null)
            return;

        traverse(root.left);

        if(prevElement == null) {
          prevElement = root;
        } else {

            if (prevElement.val > root.val) {
              if (firstElement==null) firstElement=prevElement;
              secondElement=root;


            }
//           // Start of "do some business",
//           // If first element has not been found, assign it to prevElement (refer to 6 in the example above)
//           if (firstElement == null && prevElement.val >= root.val) {
//               firstElement = prevElement;
//           }

//           // If first element is found, assign the second element to the root (refer to 2 in the example above)
//           if (firstElement != null && prevElement.val >= root.val) {
//               secondElement = root;
//           }
          prevElement = root;

        }


        // End of "do some business"

        traverse(root.right);

    }
}



    // TreeNode *pre = NULL, *first = NULL, *second = NULL;
    // void recoverTree(TreeNode* root) {
    //     inorder(root);
    //     swap(first->val, second->val);
    // }
    // void inorder(TreeNode* root) {
    //     if (!root) return;
    //     inorder(root->left);
    //     if (!pre) pre = root;
    //     else {
```

Saved to local

Testcase Test Result

Code

Java Auto

```
78      //     else {
79      //         if (pre->val > root->val) {
80      //             if (!first) first = pre;
81      //             second = root;
82      //         }
83      //         pre = root;
84      //     }
85      //     inorder(root->right);
86      // }
87
88
89  //moris traverse
90  //     public void recoverTree(TreeNode root) {
91  //       TreeNode pre=null;
92  //       TreeNode cur=root;
93  //       TreeNode n1=null;
94  //       TreeNode n2=null;
95
96  //       while(cur!=null) {
97  //         if(cur.left!=null)   {
98  //           TreeNode tmp = cur.left;
99  //           while(tmp.right!=null && tmp.right!=cur) tmp=tmp.right;
100 //           if(tmp.right!=null) {
101
102 //             if(pre!=null && pre.val>cur.val){
103 //               System.out.println("!!!"+pre.val+" "+cur.val);
104 //               if(n1==null) n1=pre;
105 //               n2=cur;
106 //             }
107 //             pre=cur;
108 //             tmp.right=null;
109 //             cur=cur.right;
110 //           }else {
111 //             tmp.right=cur;
112 //             cur=cur.left;
113 //           }
114 //         }else {
115 //           if(pre!=null && pre.val>cur.val){
116 //             if(n1==null) n1=pre;
117 //             n2=cur;
118 //           }
119 //           pre=cur;
120 //           cur=cur.right;
121
122 //         }
123 //       }
124 //       if(n1!=null && n2!=null) {
125 //         int tmp=n1.val;
126 //         n1.val=n2.val;
127 //         n2.val=tmp;
128 //       }
```

Saved to local

Testcase Test Result