


81. 搜索旋转排序数组 II

81. 搜索旋转排序数组 II

难度 中等  131  收藏  分享  切换为英文  关注  反馈

假设按照升序排序的数组在预先未知的某个点上进行了旋转。

(例如，数组 `[0, 0, 1, 2, 2, 5, 6]` 可能变为 `[2, 5, 6, 0, 0, 1, 2]`)。

编写一个函数来判断给定的目标值是否存在于数组中。若存在返回 `true`，否则返回 `false`。

示例 1:

```
输入: nums = [2,5,6,0,0,1,2], target = 0
输出: true
```

示例 2:

```
输入: nums = [2,5,6,0,0,1,2], target = 3
输出: false
```

进阶:

- 这是 [搜索旋转排序数组](#) 的延伸题目，本题中的 `nums` 可能包含重复元素。
- 这会影响到程序的时间复杂度吗？会有怎样的影响，为什么？

```
class Solution {
public:
    bool search(vector<int>& nums, int target) {
        for (auto &v : nums)
            if (v == target)
                return true;
        return false;
    }
};
```

82. 删除排序链表中的重复元素 II

82. 删除排序链表中的重复元素 II

难度 中等

👍 256

📖 收藏

🔗 分享

🌐 切换为英文

🔔 关注

💡 反馈

给定一个排序链表，删除所有含有重复数字的节点，只保留原始链表中 *没有重复出现* 的数字。

示例 1:

输入: 1->2->3->3->4->4->5
输出: 1->2->5

示例 2:

输入: 1->1->1->2->3
输出: 2->3

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode* deleteDuplicates(ListNode* head) {
        ListNode* dummy = new ListNode(0);
        dummy->next = head;
        ListNode* p = dummy;
        while (p->next)
        {
            ListNode* q = p->next;
            while (q && q->val == p->next->val)
                q = q->next;
            if (p->next->next == q) p = p->next;
            else p->next = q;
        }
        return dummy->next;
    }
};
```

83. 删除排序链表中的重复元素

83. 删除排序链表中的重复元素

难度 简单

👍 289

📖 收藏

🔗 分享

🌐 切换为英文

🔔 关注

📝 反馈

给定一个排序链表，删除所有重复的元素，使得每个元素只出现一次。

示例 1:

输入: 1->1->2

输出: 1->2

示例 2:

输入: 1->1->2->3->3

输出: 1->2->3

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode* deleteDuplicates(ListNode* head) {
        if(!head) return NULL;
        ListNode *first,*second;
        first = second = head;
        while(first)
        {
            if(first->val != second->val)
            {
                second->next = first;
                second = first;
            }
            first = first->next;
        }
        second->next = NULL;
        return head;
    }
};
```

84. 柱状图中最大的矩形

84. 柱状图中最大的矩形

难度 **困难**

👍 530

❤ 收藏

🔗 分享

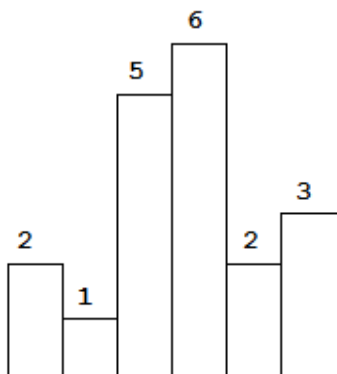
🌐 切换为英文

🔔 关注

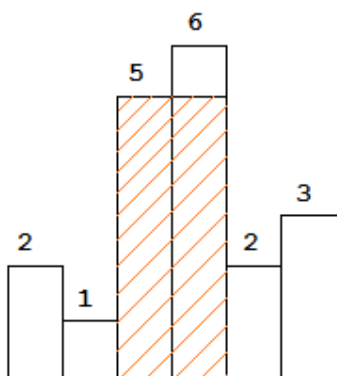
📝 反馈

给定 n 个非负整数，用来表示柱状图中各个柱子的高度。每个柱子彼此相邻，且宽度为 1。

求在该柱状图中，能够勾勒出来的矩形的最大面积。



以上是柱状图的示例，其中每个柱子的宽度为 1，给定的高度为 `[2, 1, 5, 6, 2, 3]`。



图中阴影部分为所能勾勒出的最大矩形面积，其面积为 10 个单位。

```
class Solution {
public:
    int largestRectangleArea(vector<int>& heights) {
        int n = heights.size();
        vector<int> left(n), right(n); //左右边界

        stack<int> stk;
        //左边
        for(int i = 0; i < n; i++)
        {
            while(stk.size() && heights[stk.top()] >= heights[i]) stk.pop();
            if(stk.empty()) left[i] = -1;
            else left[i] = stk.top();
            stk.push(i);
        }
        while(stk.size()) stk.pop();
        //右边
        for(int i = n - 1; i >= 0; i--)
        {
            while(stk.size() && heights[stk.top()] >= heights[i]) stk.pop();
```

```


        if(stk.empty()) rihgt[i] = n;
        else rihgt[i] = stk.top();
        stk.push(i);
    }

    int res = 0;
    //枚举每个边界，取最大值
    for(int i = 0; i < n; i++) res = max(res, heights[i] * (rihgt[i] - left[i]
- 1));
    return res;
}
};

```

85. 最大矩形

85. 最大矩形

难度 困难  380  收藏  分享  切换为英文  关注  反馈

给定一个仅包含 0 和 1 的二维二进制矩阵，找出只包含 1 的最大矩形，并返回其面积。

示例:

输入:

```

[
  ["1","0","1","0","0"],
  ["1","0","1","1","1"],
  ["1","1","1","1","1"],
  ["1","0","0","1","0"]
]
输出: 6

```

```

class Solution {
public:
    int maximalRectangle(vector<vector<char>>& matrix) {
        int n = matrix.size(), m, ans = 0;
        if (n == 0)
            return 0;
        m = matrix[0].size();
        vector<int> heights(m + 1, 0);
        heights[m] = -1;
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++)
                if (matrix[i][j] == '0')
                    heights[j] = 0;
                else
                    heights[j]++;

            stack<int> st;
            for (int j = 0; j <= m; j++) {
                while (!st.empty() && heights[j] < heights[st.top()]) {
                    int cur = st.top();
                    st.pop();
                    if (st.empty())
                        ans = max(ans, heights[cur] * j);
                }
            }
        }
        return ans;
    }
};

```

```

        else
            ans = max(ans, heights[cur] * (j - st.top() - 1));
        }
        st.push(j);
    }
}
return ans;
}
};

```

86. 分隔链表

86. 分隔链表

难度 中等  184  收藏  分享  切换为英文  关注  反馈

给定一个链表和一个特定值 x ，对链表进行分隔，使得所有小于 x 的节点都在大于或等于 x 的节点之前。

你应当保留两个分区中每个节点的初始相对位置。

示例:

输入: head = 1->4->3->2->5->2, $x = 3$
 输出: 1->2->2->4->3->5

```

/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode* partition(ListNode* head, int x) {
        ListNode *before = new ListNode(0);
        ListNode *after = new ListNode(0);
        ListNode *pb = before, *pa = after;

        for (ListNode *p = head; p; p = p->next)
            if (p->val < x)
            {
                pb->next = p;
                pb = p;
            }
            else
            {
                pa->next = p;
                pa = p;
            }

        pb->next = after->next;
        pa->next = 0;

        return before->next;
    }
};

```

```
}  
};
```

87. 扰乱字符串

87. 扰乱字符串

难度 **困难**

👍 112

📖 收藏

🔗 分享

🌐 切换为英文

🔔 关注

💡 反馈

给定一个字符串 $s1$ ，我们可以把它递归地分割成两个非空子字符串，从而将其表示为二叉树。

下图是字符串 $s1 = \text{"great"}$ 的一种可能的表示形式。



在扰乱这个字符串的过程中，我们可以挑选任何一个非叶节点，然后交换它的两个子节点。

例如，如果我们挑选非叶节点 `"gr"`，交换它的两个子节点，将会产生扰乱字符串 `"rgeat"`。



我们将 `"rgeat"` 称作 `"great"` 的一个扰乱字符串。

同样地，如果我们继续交换节点 `"eat"` 和 `"at"` 的子节点，将会产生另一个新的扰乱字符串 `"rgtae"`。

```

    r g t a e
   /   \
  rg    tae
 / \   / \
r  g ta e
    / \
   t  a

```

我们将 “rgtae” 称作 “great” 的一个扰乱字符串。

给出两个长度相等的字符串 $s1$ 和 $s2$ ，判断 $s2$ 是否是 $s1$ 的扰乱字符串。

示例 1:

```

输入: s1 = "great", s2 = "rgeat"
输出: true

```

示例 2:

```

输入: s1 = "abcde", s2 = "caebd"
输出: false

```

```

class Solution {
public:
    bool isScramble(string s1, string s2) {
        if (s1 == s2) return true;
        string ss1 = s1, ss2 = s2;
        sort(ss1.begin(), ss1.end()), sort(ss2.begin(), ss2.end());
        if (ss1 != ss2) return false;
        for (int i = 1; i < s1.size(); i++) {
            if (isScramble(s1.substr(0, i), s2.substr(0, i))
                && isScramble(s1.substr(i), s2.substr(i)))
                return true;
            if (isScramble(s1.substr(0, i), s2.substr(s2.size() - i))
                && isScramble(s1.substr(i), s2.substr(0, s2.size() - i)))
                return true;
        }
        return false;
    }
};

```

88. 合并两个有序数组

88. 合并两个有序数组

难度 简单

👍 473

❤ 收藏

🔗 分享

🌐 切换为英文

🔔 关注

📝 反馈

给你两个有序整数数组 *nums1* 和 *nums2*，请你将 *nums2* 合并到 *nums1* 中，使 *nums1* 成为一个有序数组。

说明:

- 初始化 *nums1* 和 *nums2* 的元素数量分别为 *m* 和 *n*。
- 你可以假设 *nums1* 有足够的空间（空间大小大于或等于 $m + n$ ）来保存 *nums2* 中的元素。

示例:

输入:

nums1 = [1,2,3,0,0,0], *m* = 3

nums2 = [2,5,6], *n* = 3

输出: [1,2,2,3,5,6]

```
class Solution {
public:
    void merge(vector<int>& nums1, int m, vector<int>& nums2, int n) {
        int p = m - 1, q = n - 1, cur = m + n - 1;
        while(p >= 0 && q >= 0)
        {
            if(nums1[p] >= nums2[q]) nums1[cur --] = nums1[p --];
            else nums1[cur --] = nums2[q --];
        }
        while(p >= 0) nums1[cur --] = nums1[p --];
        while(q >= 0) nums1[cur --] = nums2[q --];
    }
};
```

89. 格雷编码

89. 格雷编码

难度 中等

👍 150

📖 收藏

🔗 分享

🌐 切换为英文

🔔 关注

📝 反馈

格雷编码是一个二进制数字系统，在该系统中，两个连续的数值仅有一个位数的差异。

给定一个代表编码总位数的非负整数 n ，打印其格雷编码序列。格雷编码序列必须以 0 开头。

示例 1:

```
输入: 2
输出: [0,1,3,2]
解释:
00 - 0
01 - 1
11 - 3
10 - 2
```

对于给定的 n ，其格雷编码序列并不唯一。
例如， $[0,2,3,1]$ 也是一个有效的格雷编码序列。

```
00 - 0
10 - 2
11 - 3
01 - 1
```

示例 2:

```
输入: 0
输出: [0]
解释: 我们定义格雷编码序列必须以 0 开头。
       给定编码总位数为  $n$  的格雷编码序列，其长度为  $2^n$ 。当  $n = 0$  时，长度为  $2^0 = 1$ 。
       因此，当  $n = 0$  时，其格雷编码序列为  $[0]$ 。
```

```
class Solution {
public:
    vector<int> grayCode(int n) {
        vector<int> res;
        res.push_back(0);
        int t = 1;
        while (n -- )
        {
            vector<int> newRes;
            for (int i = 0; i < res.size(); i ++ )
                newRes.push_back(res[i]);
            for (int i = res.size() - 1; i >= 0; i -- )
                newRes.push_back(t + res[i]);
            res = newRes;
            t *= 2;
        }
        return res;
    }
};
```

90. 子集 II

90. 子集 II

难度 中等

👍 194

📖 收藏

🔗 分享

🌐 切换为英文

🔔 关注

📝 反馈

给定一个可能包含重复元素的整数数组 *nums*，返回该数组所有可能的子集（幂集）。

说明：解集不能包含重复的子集。

示例：

```
输入：[1,2,2]
输出：
[
  [2],
  [1],
  [1,2,2],
  [2,2],
  [1,2],
  []
]
```

```
class Solution {
public:
    vector<vector<int>> ans;
    vector<int> path;

    vector<vector<int>> subsetsWithDup(vector<int>& nums) {
        sort(nums.begin(),nums.end());
        dfs(nums,0);
        return ans;
    }

    void dfs(vector<int> &nums,int u)
    {
        if(u == nums.size())
        {
            ans.push_back(path);
            return;
        }

        int k = 0;
        while(u + k < nums.size() && nums[u + k] == nums[u]) k ++;

        for(int i = 0; i <= k;i ++){
            dfs(nums,u + k);
            path.push_back(nums[u]);
        }

        for(int i = 0;i <= k;i ++){path.pop_back();}
    }
};
```

91. 解码方法

91. 解码方法

难度 中等

343

收藏

分享

切换为英文

关注

反馈

一条包含字母 A-Z 的消息通过以下方式进行了编码：

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

给定一个只包含数字的非空字符串，请计算解码方法的总数。

示例 1:

```
输入: "12"
输出: 2
解释: 它可以解码为 "AB" (1 2) 或者 "L" (12)。
```

示例 2:

```
输入: "226"
输出: 3
解释: 它可以解码为 "BZ" (2 26), "VF" (22 6), 或者 "BBF" (2 2 6)。
```

```
class Solution {
public:
    int numDecodings(string s) {
        int n = s.size();
        vector<int> f(n + 1);
        f[0] = 1;
        for(int i = 1; i <= n; i++)
        {
            if(s[i - 1] != '0') f[i] += f[i - 1];
            if(i >= 2)
            {
                int t = (s[i - 2] - '0') * 10 + s[i - 1] - '0';
                if(t >= 10 && t <= 26) f[i] += f[i - 2];
            }
        }
        return f[n];
    }
};
```

92. 反转链表 II

92. 反转链表 II

难度 中等

👍 342

📖 收藏

🔗 分享

🌐 切换为英文

🔔 关注

💡 反馈

反转从位置 m 到 n 的链表。请使用一趟扫描完成反转。

说明:

$1 \leq m \leq n \leq$ 链表长度。

示例:

输入: 1->2->3->4->5->NULL, $m = 2$, $n = 4$

输出: 1->4->3->2->5->NULL

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode* reverseBetween(ListNode* head, int m, int n) {
        if(m == n) return head;
        ListNode *dummy = new ListNode(-1);
        dummy->next = head;
        ListNode *b = dummy;
        for(int i = 0; i < m - 1; i++) b = b->next;
        ListNode *a = b;
        b = b->next;
        ListNode *c = b->next;
        for(int i = 0; i < n - m; i++)
        {
            ListNode *t = c->next;
            c->next = b;
            b = c, c = t;
        }
        ListNode *mp = a->next;
        ListNode *np = b;
        a->next = np, mp->next = c;
        return dummy->next;
    }
};
```

93. 复原IP地址

93. 复原IP地址

难度 中等

👍 235

❤ 收藏

🔗 分享

🌐 切换为英文

🔔 关注

📝 反馈

给定一个只包含数字的字符串，复原它并返回所有可能的 IP 地址格式。

示例:

输入: "25525511135"

输出: ["255.255.11.135", "255.255.111.35"]

```
class Solution {
public:
    vector<string> ans;
    vector<int> path;

    vector<string> restoreIpAddresses(string s) {
        dfs(0, 0, s);
        return ans;
    }

    // u表示枚举到的字符串下标, k表示当前截断的IP个数, s表示原字符串
    void dfs(int u, int k, string &s)
    {
        if (u == s.size())
        {
            if (k == 4)
            {
                string ip = to_string(path[0]);
                for (int i = 1; i < 4; i++)
                    ip += '.' + to_string(path[i]);
                ans.push_back(ip);
            }
            return;
        }
        if (k > 4) return;

        unsigned t = 0;
        for (int i = u; i < s.size(); i++)
        {
            t = t * 10 + s[i] - '0';
            if (t >= 0 && t < 256)
            {
                path.push_back(t);
                dfs(i + 1, k + 1, s);
                path.pop_back();
            }
            if (!t) break;
        }
    }
};
```

94. 二叉树的中序遍历

94. 二叉树的中序遍历

难度 中等

👍 465

❤ 收藏

🔗 分享

🌐 切换为英文

🔔 关注

📝 反馈

给定一个二叉树，返回它的中序遍历。

示例:

输入: [1,null,2,3]

```
  1
   \
    2
   /
  3
```

输出: [1,3,2]

进阶: 递归算法很简单，你可以通过迭代算法完成吗？

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    vector<int> inorderTraversal(TreeNode* root) {
        vector<int> res;
        stack<TreeNode*> stk;

        auto p = root;
        while(p || stk.size())
        {
            while(p)
            {
                stk.push(p);
                p = p->left;
            }

            p = stk.top();
            stk.pop();

            res.push_back(p->val);
            p = p->right;
        }

        return res;
    }
};
```

95. 不同的二叉搜索树 II

95. 不同的二叉搜索树 II

难度 中等

👍 355

❤ 收藏

🔗 分享

🌐 切换为英文

🔔 关注

📝 反馈

给定一个整数 n ，生成所有由 $1 \dots n$ 为节点所组成的二叉搜索树。

示例:

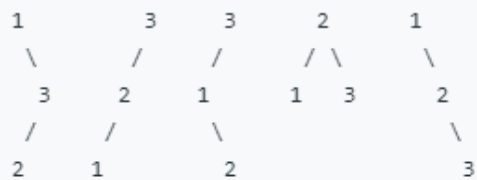
输入: 3

输出:

```
[
  [1,null,3,2],
  [3,2,null,1],
  [3,1,null,null,2],
  [2,1,3],
  [1,null,2,null,3]
]
```

解释:

以上的输出对应以下 5 种不同结构的二叉搜索树:



```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    vector<TreeNode*> generateTrees(int n) {
        if (!n) return vector<TreeNode*>();
        return dfs(1, n);
    }

    vector<TreeNode*> dfs(int l, int r)
    {
        vector<TreeNode*> res;
        if (l > r)
        {
            res.push_back(NULL);
            return res;
        }
        for (int i = l; i <= r; i++)
        {
            vector<TreeNode*> left = dfs(l, i - 1)
                , right = dfs(i + 1, r);
            for (auto &lc : left)
                for (auto &rc : right)
                {
```



```

        TreeNode *root = new TreeNode(i);
        root->left = lc;
        root->right = rc;
        res.push_back(root);
    }
}
return res;
}
};

```

96. 不同的二叉搜索树

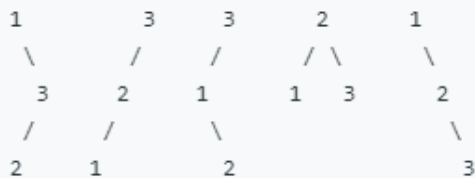
96. 不同的二叉搜索树

难度 中等  469  收藏  分享  切换为英文  关注  反馈

给定一个整数 n ，求以 $1 \dots n$ 为节点组成的二叉搜索树有多少种？

示例:

输入: 3
 输出: 5
 解释:
 给定 $n = 3$ ，一共有 5 种不同结构的二叉搜索树：



```

class Solution {
public:
    int numTrees(int n) {
        vector<int> f(n + 1);
        f[0] = 1;
        for (int i = 1; i <= n; i++) {
            f[i] = 0;
            for (int j = 1; j <= i; j++) {
                f[i] += f[j - 1] * f[i - j];
            }
        }
        return f[n];
    }
};

```

97. 交错字符串

97. 交错字符串

难度 困难

👍 156

📖 收藏

🔗 分享

🌐 切换为英文

🔔 关注

📝 反馈

给定三个字符串 $s1$, $s2$, $s3$, 验证 $s3$ 是否是由 $s1$ 和 $s2$ 交错组成的。

示例 1:

输入: $s1 = "aabcc"$, $s2 = "dbbca"$, $s3 = "aadbcbcbac"$
输出: true

示例 2:

输入: $s1 = "aabcc"$, $s2 = "dbbca"$, $s3 = "aadbcbaccc"$
输出: false

```
class Solution {
public:
    bool isInterleave(string s1, string s2, string s3) {
        int n = s1.size(), m = s2.size(), k = s3.size();
        if (k != n + m) return false;
        vector<vector<int>> f =
            vector<vector<int>>(n + 1, vector<int>(m + 1));
        f[0][0] = 1;
        for (int i = 1; i <= n; i++)
            f[i][0] = f[i - 1][0] && s1[i - 1] == s3[i - 1];
        for (int i = 1; i <= m; i++)
            f[0][i] = f[0][i - 1] && s2[i - 1] == s3[i - 1];
        for (int i = 1; i <= n; i++)
            for (int j = 1; j <= m; j++)
            {
                f[i][j] = 0;
                if (s1[i - 1] == s3[i + j - 1])
                    f[i][j] |= f[i - 1][j];
                if (s2[j - 1] == s3[i + j - 1])
                    f[i][j] |= f[i][j - 1];
            }
        return f[n][m];
    }
};
```

98. 验证二叉搜索树

98. 验证二叉搜索树

难度 中等

👍 506

📖 收藏

🔗 分享

🌐 切换为英文

🔔 关注

📝 反馈

给定一个二叉树，判断其是否是一个有效的二叉搜索树。

假设一个二叉搜索树具有如下特征：

- 节点的左子树只包含小于当前节点的数。
- 节点的右子树只包含大于当前节点的数。
- 所有左子树和右子树自身必须也是二叉搜索树。

示例 1:

```
输入：
    2
   / \
  1   3
输出：true
```

示例 2:

```
输入：
    5
   / \
  1   4
   / \
  3   6
输出：false
解释：输入为：[5,1,4,null,null,3,6]。
      根节点的值为 5 ，但是其右子节点值为 4 。
```

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    bool isValidBST(TreeNode* root) {
        return dfs(root, INT_MIN, INT_MAX);
    }
    bool dfs(TreeNode *root, long long minv, long long maxv)
    {
        if(!root) return true;
        if(root->val < minv || root->val > maxv) return false;

        return dfs(root->left, minv, root->val - 111) && dfs(root->right, root->val
+ 111, maxv);
    }
};
```

99. 恢复二叉搜索树

99. 恢复二叉搜索树

难度 **困难**

183

收藏

分享

切换为英文

关注

反馈

二叉搜索树中的两个节点被错误地交换。

请在不改变其结构的情况下，恢复这棵树。

示例 1:

输入: [1,3,null,null,2]

```
  1
 /
3
 \
  2
```

输出: [3,1,null,null,2]

```
  3
 /
1
 \
  2
```

/*

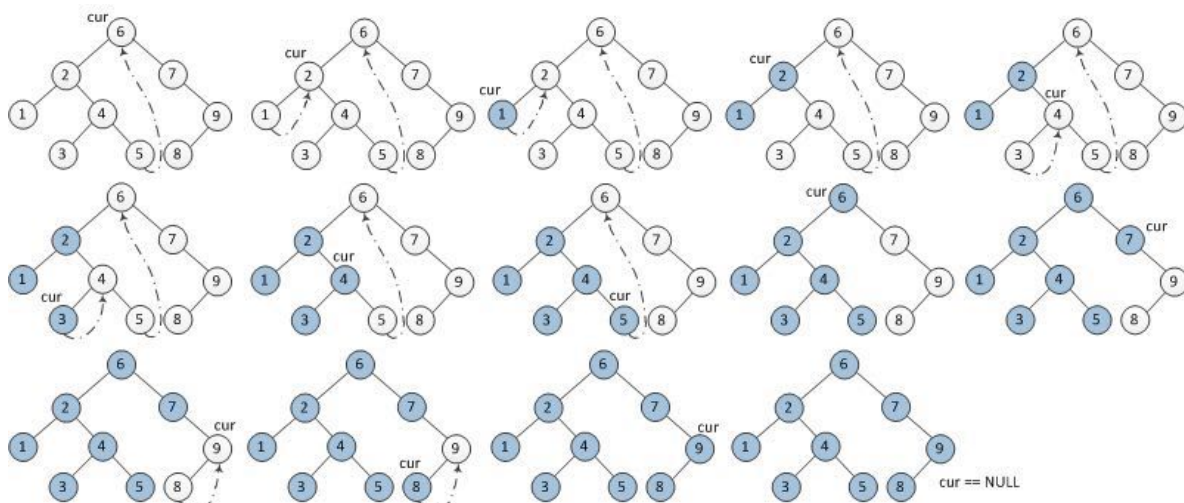
这道题目如果用递归做，递归的层数最坏是 $O(n)O(n)$ 级别的，所以系统栈的空间复杂度是 $O(n)O(n)$ ，与题目要求的 $O(1)O(1)$ 额外空间不符。

同理用栈模拟递归的迭代方式的空间复杂度也是 $O(n)O(n)$ ，不符合题目要求。

这道题目可以用Morris-traversal算法，该算法可以用额外 $O(1)O(1)$ 的空间，以及 $O(n)O(n)$ 的时间复杂度，中序遍历一棵二叉树。

Morris-traversal 算法流程：

下图给了一个具体示例：



/*从根节点开始遍历，直至当前节点为空为止：

如果当前节点没有左儿子，则打印当前节点的值，然后进入右子树；

如果当前节点有左儿子，则找当前节点的前驱。

(1) 如果前驱节点的右儿子为空，说明左子树没遍历过，则进入左子树遍历，并将前驱节点的右儿子置成当前节点，方便回溯；

(2) 如果前驱节点的右儿子为当前节点，说明左子树已被遍历过，则将前驱节点的右儿子恢复为空，然后打印当前节点的值，然后进入右子树继续遍历；

中序遍历的结果就是二叉树搜索树所表示的有序数列。有序数列从小到大排序，但有两个数被交换了位置。共有两种情况：

交换的是相邻两个数，例如 1 3 2 4 5 6，则第一个逆序对，就是被交换的两个数，这里是3和2；

交换的是不相邻的数，例如 1 5 3 4 2 6，则第一个逆序对的第一个数，和第二个逆序对的第二个数，就是被交换的两个数，这里是5和2；

找到被交换的数后，我们将它们换回来即可。

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    void recoverTree(TreeNode* root) {
        TreeNode *first = NULL, *second, *prep = NULL;
        while (root)
        {
            if (!root->left)
            {
                if (prep && prep->val > root->val)
                {
                    if (!first) first = prep, second = root;
                    else second = root;
                }
                prep = root;
                root = root->right;
            }
            else
            {
                TreeNode *p = root->left;
                while (p->right && p->right != root) p = p->right;
                if (!p->right)
                {
                    p->right = root;
                    root = root->left;
                }
                else
                {
                    p->right = NULL;
                    if (prep && prep->val > root->val)
                    {
                        if (!first) first = prep, second = root;
                        else second = root;
                    }
                }
                prep = root;
                root = root->right;
            }
        }
        if (first && second)
            swap(first->val, second->val);
    }
};
```







```

    }
    }
}
swap(first->val, second->val);
}
};

```

100. 相同的树

100. 相同的树

难度 **简单**  343  收藏  分享  切换为英文  关注  反馈

给定两个二叉树，编写一个函数来检验它们是否相同。

如果两个树在结构上相同，并且节点具有相同的值，则认为它们是相同的。

示例 1:

输入:

```

      1      1
     / \    / \
    2   3  2   3

[1,2,3],  [1,2,3]

```

输出: true

示例 2:

输入:

```

      1      1
     /      \
    2         2

[1,2],     [1,null,2]

```

输出: false

示例 3:

输入:

```

      1      1
     / \    / \
    2   1  1   2

[1,2,1],  [1,1,2]

```

输出: false

```

/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };

```

```
*/  
class Solution {  
public:  
    bool isSameTree(TreeNode* p, TreeNode* q) {  
        if(!p || !q) return !p && !q;  
        return p->val == q->val && isSameTree(p->left,q->left) && isSameTree(p->right,q->right);  
    }  
};
```