递归的一些例子

> QQ：475679136编写
>
> 微店：一条黄学长

基本概念

递归组成部分

1. 边界条件 —— 对于边界情况应该如何操作
2. 一般操作 —— 对于一般情况应该如何操作
3. 返回部分 —— 具体需要返回哪些内容

谨记上面的内容
分析递归的过程需要我们**先明确要做什么**，下面的例子可能有点抽象

> 假如我们有一个递归函数f，它的功能是x，那么我们先**假设**存在一个f的实现func，func能够完成功能x
> 先假设存在这么一个函数很重要
> 然后我们再依次按照上面的1、2、3部分来填充func
> 最后我们就能在仅有假设的情况下，通过分析1、2、3来真正的得到一个f的实现func

题目

206.反转链表

206. 反转链表

难度 简单 1319

反转一个单链表。

示例:

```
输入: 1->2->3->4->5->NULL
输出: 5->4->3->2->1->NULL
```


分析:

假设存在一个函数func(head)，它接收头结点为head的单链表，然后返回翻转后的链表

1. 边界条件

   当链表为空，或者链表只有一个结点时，返回head

```
if not head or not head.next:
    return head
```

2. 一般操作

   我们目前有一个指针head指向当前结点，则

   我们将head之后的链表翻转——func(head.next)

   然后将head添加到翻转后的链表——尾部插入，同时更新head的next为None

   然后返回之前翻转后的链表首结点

   我们目前有一个翻转后的链表t和一个需要插入的结点head，想要在链表最后插入结点head，然后返回t

```
    # func返回以head.next结点为起点的单链表，返回反转后的链表的首结点
    # t是翻转链表后的首结点
    t = func(head.next)
    # 取得翻转链表后的最后一个结点
    tail = t
    while tail.next:
        tail = tail.next
    tail.next = head    # 完成插入操作
```

3. 返回部分

```
    t = func(head.next)
    do something......
    return t
```

完整代码

```
def func(head):
    if not head or not head.next:
        return head
    # 翻转head.next之后的链表
    t = func(head.next)
    # 取得翻转链表后的最后一个结点
    tail = t
    while tail.next:
        tail = tail.next
    head.next = None # 保证链表最后指向None
    tail.next = head # 完成插入操作
    return t
```

另一种解法不太贴切今天想讲的内容，但还是贴一下吧 python递归逆置一条单链表详解

226.翻转二叉树

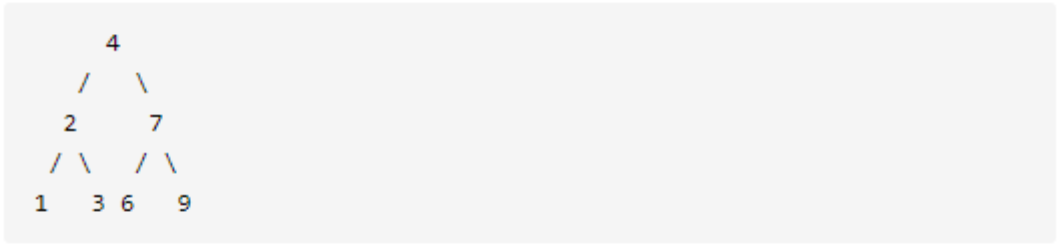
226. 翻转二叉树

难度 简单 675

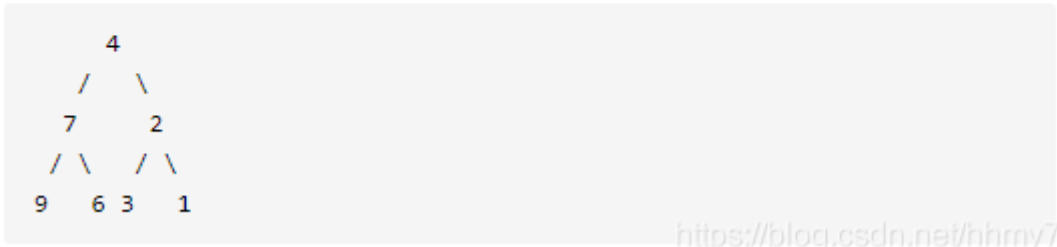
翻转一棵二叉树。

示例:

输入:



输出:



分析:

假设存在一个函数func(root)，它接收根结点为root的二叉树，然后返回翻转后的二叉树

1. 边界条件
   为空树、或者只有一个根节点的树，直接返回

```
if not root or (not root.left and not root.right):
    return root
```

2. 一般情况
   我们当前有root、root.left构成的左子树、root.right构成的右子树，我们需要翻转以root为根节点的二叉树
   一个想法就是，我们交换root.left和root.right，然后对root.left和root.right分别调用func即可，**因为func会翻转给定的二叉树**

```
交换root.left和root.right
func(root.left)
func(root.right)
```

3. 返回部分
   给定一颗二叉树，返回翻转后的二叉树，所以我们需要返回的是root

```
    return root
```

完整代码

```
def func(root):
    if not root or (not root.left and not root.right):
        return root
    交换root.left和root.right
    func(root.left)
    func(root.right)
    return root
```

提交的代码

```
class Solution:
    def invertTree(self, root: TreeNode) -> TreeNode:
        if not root:
            return None

        root.left,root.right = root.right,root.left
        self.invertTree(root.left)
        self.invertTree(root.right)

        return root
```

100.相同的树

100. 相同的树

难度 简单 496

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
```


分析：

假设存在函数func(p,q)，可以判断p树和q树是否相同

1. 边界条件

   分析可知，当只有一个结点为空另一个结点不为空的时候，我们直接返回False，另一个情况是两个结点都为空，这一步我们返回True

   ```
   if 只有一边不为空:
       return False
   if 两边都为空:
       return True
   ```

2. 一般操作

   对于p、q两个结点，当它们的值不相同的时候，返回False，值相同的时候，判断左右子树的情况

```
if p、q值不相同:
    return False
```

3. 返回条件
   继续判断p、q左右子树的情况

```
return p的左子树和q的左子树相同 并且 p的右子树和q的右子树相同
```

完整代码

```
def func(p,q):
    if 只有一边不为空:
        return False
    if 两边都为空:
        return True
    if p、q值不相同:
        return False
    return p的左子树和q的左子树相同 并且 p的右子树和q的右子树相同
```

提交代码

```
class Solution:
    def isSameTree(self, p: TreeNode, q: TreeNode) -> bool:
        if (not p and q) or ( p and not q):
            return False
        if not p and not q:
            return True
        if p.val != q.val:
            return False
        return self.isSameTree(p.left,q.left) and self.isSameTree(p.right,q.right)
```

35. 二叉搜索树的最近公共祖先

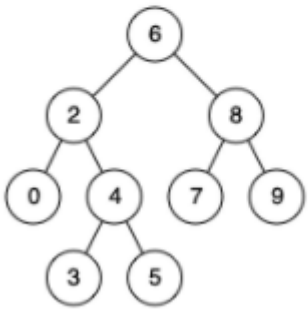
235. 二叉搜索树的最近公共祖先

难度 简单 483 收藏 分享 切换为英文 接收动态 反馈

给定一个二叉搜索树, 找到该树中两个指定节点的最近公共祖先。

百度百科中最近公共祖先的定义为：“对于有根树 T 的两个结点 p、q，最近公共祖先表示为一个结点 x，满足 x 是 p、q 的祖先且 x 的深度尽可能大（**一个节点也可以是它自己的祖先**）。”

例如，给定如下二叉搜索树: root = [6,2,8,0,4,7,9,null,null,3,5]



示例 1:

```
输入: root = [6,2,8,0,4,7,9,null,null,3,5], p = 2, q = 8
输出: 6
解释: 节点 2 和节点 8 的最近公共祖先是 6。
```

示例 2:

```
输入: root = [6,2,8,0,4,7,9,null,null,3,5], p = 2, q = 4
输出: 2
解释: 节点 2 和节点 4 的最近公共祖先是 2，因为根据定义最近公共祖先节点可以为节点本身。
```

说明:

- 所有节点的值都是唯一的。
- p、q 为不同节点且均存在于给定的二叉搜索树中。

分析：

假设存在函数func(root,p,q)，可以在root中找到p、q结点的最近公共祖先

1. 边界情况

   考虑最极端的情况，也就是初始传入的时候，p和q中某个结点等于root，则我们可以直接返回root，因为此时root一定是最近的公共祖先结点

```
    if q或p是根节点:
        return root
```

2. 一般操作

   除了上述边界情况以外，我们具体分析在一颗二叉搜索树里面root、p、q结点的存在可能性

   1. p和q都在root的一边，即`p.val < root.val and q.val < root.val` 或者 `p.val > root.val and q.val > root.val`
   2. p和q各自在root的一边，即`p.val < root.val and q.val > root.val` 或者 `p.val > root.val and q.val < root.val`

观察情况2可以发现当p和q在root的左右两边的时候，root一定是p和q的最近公共祖先，想象一下左右两棵树加上根节点合并成一颗更大的树
而情况1时，我们要继续往下递归，找到它们的最近公共祖先

```
if p和q各自在root的一边:
    return root
else if p和q都在root的左边:
    在root的左子树中找到p和q的最近公共祖先
else if p和q都在root的右边:
    在root的右子树中找到p和q的最近公共祖先
```

3. 返回部分
我们返回祖先即可

```
    if p和q各自在root的一边:
        return root
    else if p和q都在root的左边:
        return 在root的左子树中找到p和q的最近公共祖先
    else if p和q都在root的右边:
        return 在root的右子树中找到p和q的最近公共祖先
```

完整代码

```
def func(root,p,q):
    if q或p是根节点:
        return root
    if p和q各自在root的一边:
        return root
    else if p和q都在root的左边:
        return 在root的左子树中找到p和q的最近公共祖先
    else if p和q都在root的右边:
        return 在root的右子树中找到p和q的最近公共祖先
```

提交代码

```
class Solution:
    def lowestCommonAncestor(self, root: 'TreeNode', p: 'TreeNode', q: 'TreeNode') ->
'TreeNode':
        if root == p or root == q:
            return root
        if (p.val<root.val and q.val>root.val ) or (p.val>root.val  and q.val<root.val):
            return root
        elif p.val<root.val and q.val<root.val:
            return self.lowestCommonAncestor(root.left,p,q)
        else:
            return self.lowestCommonAncestor(root.right,p,q)
```

判断二叉树中是否有结点值等于当前层次

四、(15 分)使用递归算法，判断二叉链表存储的二叉树中是否存在结点，它的元素值（皆为整数）和层次一样（树根为 1）。

分析：

假设存在func(root,layer = 1)，它可以判断树中是否有满足条件的结点

1. 边界情况
   显然树空是边界情况，此时返回False

```
if not root:
    return False
```

2. 一般操作
   这里思考一下，如果当前结点值等于layer值，那么就返回True，如果当前结点值不等于layer值，我们还需要判断左右子树里面是否存在着满足条件的结点

```
if root.val == layer:
    return True
else:
    return 判断左右子树里面是否存在满足条件的结点
```

3. 返回部分
   返回部分里面我们继续完善else条件里面的操作
   我们一开始假设了func的功能，因此我们可以做如下操作

```
if root.val == layer:
    return True
else:
    return func(root.left,layer+1) or func(root.left,layer+1)
```

完整代码

```
def func(root,layer = 1):
    if not root:
        return False
    if root.val == layer:
        return True
    else:
        return func(root.left,layer+1) or func(root.left,layer+1)
```

最后我们可以优化返回部分

```
def func(root,layer = 1):
    if not root:
        return False
    return root.val == layer || func(root.left,layer+1) || func(root.right,layer+1)
```

最后一行表达式只要有一个True成立，就会直接返回True