简单1_50

1-10

7、整数反转

给你一个 32 位的有符号整数 x ，返回将 x 中的数字部分反转后的结果。

如果反转后整数超过 32 位的有符号整数的范围 [−231，231 − 1] ，就返回 0。

假设环境不允许存储 64 位整数（有符号或无符号）。

> 输入：x = 123
> 输出：321

> 输入：x = -123
> 输出：-321

> 输入：x = 120
> 输出：21

题解

1、直接来

数字转换成字符，然后反转

```
def reverse_force(self, x: int) -> int:
        if -10 < x < 10:
            return x
        str_x = str(x)
        if str_x[0] != "-":
            str_x = str_x[::-1]
            x = int(str_x)
        else:
            str_x = str_x[:0:-1]
            x = int(str_x)
            x = -x
        return x if -2147483648 < x < 2147483647 else 0
```

2、优化解

1. 我们可以一次构建反转整数的一位数字。在这样做的时候，我们可以预先检查向原整数附加另一位数字是否会导致溢出。
2. 反转整数的方法可以与反转字符串进行类比。
3. 我们想重复 “弹出” x 的最后一位数字，并将它 “推入” 到 res 的后面。最后，res 将与 x 相反。

   优化解：

```
def reverse_better(self, x: int) -> int:
        y, res = abs(x), 0
        # 则其数值范围为 [-2^31,  2^31 - 1]
        boundry = (1<<31) -1 if x>0 else 1<<31
        while y != 0:
            res = res*10 +y%10
            if res > boundry :
                return 0
            y //=10
        return res if x >0 else -res
```

9、回文数

给你一个整数 x ，如果 x 是一个回文整数，返回 true ； 否则，返回 false 。

回文数是指正序（从左向右）和倒序（从右向左）读都是一样的整数。例如，121 是回文，而 123 不是。

> 输入：x = 121
> 输出：true

> 输入：x = -121
> 输出：false
> 解释：从左向右读，为 -121 。 从右向左读，为 121- 。因此它不是一个回文数。

题解

1、转换成字符，直接判断

```
class Solution:
    def isPalindrome(self, x: int) -> bool:
        s=str(x)
        return s==s[::-1]
```

2、双向队列

```
    def isPalindrome(x: int) -> bool:
        lst = list(str(x))
        while len(lst) > 1:
            if lst.pop(0) != lst.pop():
                return  False
        return True
```

3、双指针

```python
def isPalindrome(x: int) -> bool:
        lst = list(str(x))
        L, R = 0, len(lst)-1
        while L <= R:
            if lst[L] != lst[R]:
                return  False
            L += 1
            R -= 1
        return True
```

4、利用数学思想

1. 首先判断特殊情况：小于0的数，和，能被10整除且不为0的数，一定不是回文数
2. 新建变量rem，用于保存由x计算的倒序数
3. 循环，当x>rem时，取下x的最后一位，并添加到rem上。此过程是利用数学公式，整除和求余。
4. 循环结束后的判断条件：当`x==rem`或`x==rem//10`，则是回文数；否则不是回文数。

```python
class Solution:
    def isPalindrome(self, x: int) -> bool:
        # 不用字符串，则用数学思想解
        if x < 0 or (x % 10 == 0 and x !=0 ):
            return False
        rem = 0
        while x > rem:
            rem = rem*10 + x%10
            x = x//10
        return x == rem or x == rem//10
```

13、罗马数字转整数

罗马数字包含以下七种字符: `I`， `V`， `X`， `L`，`C`，`D` 和 `M`

字符	数值
I	1
V	5
X	10
L	50
C	100
D	500
M	1000

例如， 罗马数字 2 写做 II ，即为两个并列的 1。12 写做 XII ，即为 X + II 。 27 写做 XXVII, 即为 XX + V + II 。

通常情况下，罗马数字中小的数字在大的数字的右边。但也存在特例，例如 4 不写做 IIII，而是 IV。数字 1 在数字 5 的左边，所表示的数等于大数 5 减小数 1 得到的数值 4 。同样地，数字 9 表示为 IX。这个特殊的规则只适用于以下六种情况：

- I 可以放在 V (5) 和 X (10) 的左边，来表示 4 和 9。
- X 可以放在 L (50) 和 C (100) 的左边，来表示 40 和 90。
- C 可以放在 D (500) 和 M (1000) 的左边，来表示 400 和 900。

给定一个罗马数字，将其转换成整数。输入确保在 1 到 3999 的范围内。

```
输入: "III"
输出: 3
```

```
输入: "IV"
输出: 4
```

```
输入: "LVIII"
输出: 58
解释: L = 50, V= 5, III = 3.
```

```
输入: "MCMXCIV"
输出: 1994
解释: M = 1000, CM = 900, XC = 90, IV = 4.
```

题解

1、保存字典，遍历一遍，查找特殊情况，没有则按字典相加

```
class Solution:
    def romanToInt(self, s: str) -> int:
        d = {'I': 1, 'IV': 4, 'V': 5, 'IX': 9, 'X': 10, 'XL': 40, 'L': 50, 'XC':
90,  'C': 100, 'CD': 400, 'D': 500, 'CM': 900, 'M': 1000}
        result = 0
        i = 0
        while i < len(s):
            #查看当前位和下一位的字符
            str1 = s[i:i+2]
            #如果当前位置是特殊情况，那么返回其在字典中对应值，并且下一次从特殊字符之后一位开
始索引
            if str1 in d:
                result += d.get(str1)
                i += 2
            #如果当前位不是特殊情况，那么只返回当前位的数值
            else:
                result += d[s[i]]
                i += 1
        return result
```

2、字典，for循环，判断情况，

```
class Solution:
    def romanToInt(self, s: str) -> int:
        tmp = {"I": 1,
               "V": 5,
               "X": 10,
               "L": 50,
               "C": 100,
               "D": 500,
               "M": 1000}
        total = 0
        for i in range(len(s) - 1):     #循环长度-1，最后加上tmp[s[-1]]
            if tmp[s[i]] < tmp[s[i + 1]]:
                total -= tmp[s[i]]
```

```
            else:
                total += tmp[s[i]]
        total += tmp[s[-1]]
        return total
```

14、最长公共长缀

编写一个函数来查找字符串数组中的最长公共前缀。

如果不存在公共前缀，返回空字符串 `""`。

> 输入：strs = ["flower","flow","flight"]
> 输出："fl"

> 输入：strs = ["dog","racecar","car"]
> 输出：""
> 解释：输入不存在公共前缀。

题解

1、使用zip函数

```
class Solution:
    def longestCommonPrefix(self, strs):
        ret = ''
        for i in zip(*strs):
            if len(set(i)) == 1:
                ret += i[0]
            else:
                break
        return ret
```

2、使用ASCII码排序

python中可以按照ASCII码进行max和min排序，a->z的码由小到大

```
class Solution:
    def longestCommonPrefix(self, strs: List[str]) -> str:
            if not strs: return ""
            s1 = min(strs)
            s2 = max(strs)
            for i,x in enumerate(s1):
                if x != s2[i]:
                    return s2[:i]
            return s1
```

3、纵向匹配

```
def longestCommonPrefix(strs):
    if not strs: return ''
    # 按照字符串长度从小到大排序
```

```
    strs.sort(key = lambda x:len(x))
    s_min = strs[0]
    i = 0
    # 纵向扫描
    for i in range(len(s_min)):
        # 有一个不以 s_min[0:i+1] 起始 就break
        if any([not s.startswith(s_min[0:i+1]) for s in strs]):
            break
        # 如果走到了最后 且都是匹配的  因为最后一个 i=len-1 注意要+1
        if i==len(s_min)-1:
            i = len(s_min)
    return s_min[0:i]
```

26、删除有序数组中的重复项

给你一个有序数组 nums ，请你 原地 删除重复出现的元素，使每个元素 只出现一次 ，返回删除后数组的新长度。

不要使用额外的数组空间，你必须在 原地 修改输入数组 并在使用 O(1) 额外空间的条件下完成。

说明:

为什么返回数值是整数，但输出的答案是数组呢?

请注意，输入数组是以「引用」方式传递的，这意味着在函数里修改输入数组对于调用者是可见的。

你可以想象内部操作如下:

```
// nums 是以“引用”方式传递的。也就是说，不对实参做任何拷贝
int len = removeDuplicates(nums);

// 在函数里修改输入数组对于调用者是可见的。
// 根据你的函数返回的长度, 它会打印出数组中 该长度范围内 的所有元素。
for (int i = 0; i < len; i++) {
    print(nums[i]);
}
```

> 输入：nums = [1,1,2]
> 输出：2, nums = [1,2]
> 解释：函数应该返回新的长度 2 ，并且原数组 nums 的前两个元素被修改为 1, 2 。不需要考虑数组中超出新长度后面的元素。
>
> 来源：力扣（LeetCode）
> 链接：https://leetcode-cn.com/problems/remove-duplicates-from-sorted-array
>

> 输入：nums = [0,0,1,1,1,2,2,3,3,4]
> 输出：5, nums = [0,1,2,3,4]
> 解释：函数应该返回新的长度 5 ， 并且原数组 nums 的前五个元素被修改为 0, 1, 2, 3, 4 。不需要考虑数组中超出新长度后面的元素。
>
> 来源：力扣（LeetCode）
> 链接：https://leetcode-cn.com/problems/remove-duplicates-from-sorted-array
> 著作权归领扣网络所有。商业转载请联系官方授权，非商业转载请注明出处。

题解

1、双指针

题目需要我们把**去重后的结果**保存到原本的数组中，所以想到必须有一个指针指向当前需要把结果放在哪个位置。还要一个指针指向当前应该放到哪个元素。

- 慢指针作为基准，快指针用于寻找与慢指针不同的元素。
- 如果快指针和慢指针指向的元素不等，则把快指针指向的元素放到慢指针的下一个位置。
- 慢指针右移，把新的元素作为基准。

```python
class Solution(object):
    def removeDuplicates(self, nums):
        N = len(nums)
        left = 0
        for right in range(1, N):
            if nums[right] != nums[left]:
                left += 1
                nums[left] = nums[right]
        return left + 1
```

27、移除元素

给你一个数组 nums 和一个值 val，你需要 原地 移除所有数值等于 val 的元素，并返回移除后数组的新长度。

不要使用额外的数组空间，你必须仅使用 O(1) 额外空间并 原地 修改输入数组。

元素的顺序可以改变。你不需要考虑数组中超出新长度后面的元素。

> 输入：nums = [3,2,2,3], val = 3
> 输出：2, nums = [2,2]
> 解释：函数应该返回新的长度 2, 并且 nums 中的前两个元素均为 2。你不需要考虑数组中超出新长度后面的元素。例如，函数返回的新长度为 2 ，而 nums = [2,2,3,3] 或 nums = [2,2,0,0]，也会被视作正确答案。

> 输入：nums = [0,1,2,2,3,0,4,2], val = 2
> 输出：5, nums = [0,1,4,0,3]
> 解释：函数应该返回新的长度 5, 并且 nums 中的前五个元素为 0, 1, 3, 0, 4。注意这五个元素可为任意顺序。你不需要考虑数组中超出新长度后面的元素。

题解

1、双指针

同26题，只不过是找到不等于val的数，先插入进来，left再+1

```
class Solution:
    def removeElement(self, nums: List[int], val: int) -> int:
        n=len(nums)
        left=0
        for right in range(n):
            if nums[right]!=val:
                nums[left]=nums[right]
                left+=1

        return left
```

2、使用pop函数

遍历数组，遇到目标数。即通过使用POP函数去除，此时由于取出后数组长度变化，故不增加下标(相对增加坐标)

```
class Solution:
    def removeElement(self, nums: List[int], val: int) -> int:
        i,count=0,0
        while(i<len(nums)):
            if nums[i]==val:
                nums.pop(i)
                continue
            i=i+1
        return len(nums)
```

28、实现strStr()

给你两个字符串 haystack 和 needle ，请你在 haystack 字符串中找出 needle 字符串出现的第一个位置（下标从 0 开始）。如果不存在，则返回 -1 。

> 输入：haystack = "hello", needle = "ll"
> 输出：2
>
> 输入：haystack = "aaaaa", needle = "bba"
> 输出：-1
>
> 输入：haystack = "", needle = ""
> 输出：0

题解

1、滑动窗口

窗口长度为：len(needle)，然后遍历haystack，匹配是否相同

```python
class Solution:
    def strStr(self, haystack: str, needle: str) -> int:
        n=len(needle)
        for i in range(len(haystack)-n+1):
            if haystack[i:i+n]==needle:
                return i

        return -1
```

2、双指针——指针回退

```python
class Solution:
    def strStr(self, haystack: str, needle: str) -> int:
        s = haystack        #纯粹为了好写
        t = needle
        sn, tn = len(s), len(t)
        if tn == 0:
            return 0
        i, j = 0, 0
        while i < sn and j < tn:
            if s[i] == t[j]:
                i += 1
                j += 1
            else:
                i = i - j + 1   #回退
                j = 0
        ## j指针遍历了t
        if j == tn:
            return i - j
        return -1
```

3、find函数

```python
class Solution:
    def strStr(self, haystack: str, needle: str) -> int:
        if not needle:
            return 0
        return haystack.find(needle)
```

4、KMP

```python
class Solution:
    def strStr(self, haystack: str, needle: str) -> int:
        s = haystack        #纯粹为了好写
        t = needle
        sn, tn = len(s), len(t)
        if tn == 0:
            return 0
        next = self.get_next(t)
        i = 0
        j = 0
        while i < sn and j < tn:
            if j == -1 or  s[i] == t[j]:
                i += 1
                j += 1
```

```
            else:
                j = next[j]
        ## j指针遍历了t
        if j == tn:
            return i - j
        return -1

    ######## 计算next数组（本质是match_数组）########
    def get_next(self, t: str) -> List[int]:
        n = len(t)
        next = [0 for _ in range(n + 1)]
        next[0] = -1

        L = -1
        R = 0
        while R < n:
            if L == -1 or t[L] == t[R]:
                L += 1
                R += 1
                next[R] = L
            else:
                L = next[L]
        return next
```

35、搜索插入位置

给定一个排序数组和一个目标值，在数组中找到目标值，并返回其索引。如果目标值不存在于数组中，返回它将会被按顺序插入的位置。

你可以假设数组中无重复元素。

> 输入: [1,3,5,6], 5
> 输出: 2

> 输入: [1,3,5,6], 2
> 输出: 1

> 输入: [1,3,5,6], 7
> 输出: 4

题解

1、遍历一次，判断target的位置

```
class Solution:
    def searchInsert(self, nums: List[int], target: int) -> int:
        if target<=nums[0]:
            return 0
        elif target > nums[-1]:
            return len(nums)
        elif target==nums[-1]:
```

```
            return len(nums)-1

        for i in range(len(nums)-1):
            if nums[i]<target<nums[i+1]:
                return i+1
            elif nums[i]==target:
                return i
```

2、target添加到数组，数组排序，返回索引

```
class Solution:
    def searchInsert(self, nums: List[int], target: int) -> int:
        nums.append(target)
        nums.sort()
        return nums.index(target)
```

3、二分查找——（左闭右闭）0

1. 左指针 left = 0，右指针 right = n - 1
2. 中间值 mid = (left + right) // 2，循环条件 left <= right
3. 当 nums[mid] = target 时，找到目标，返回mid
4. 当 nums[mid] > target 时，说明当前值偏大，右指针左移至 mid - 1
5. 当 nums[mid] < target 时，说明当前值偏小，左指针右移至 mid + 1
6. 跳出循环时，left 指向 target 插入位置，返回 left

```
class Solution:
    def searchInsert(self, nums: List[int], target: int) -> int:
        left = 0
        right = len(nums) - 1
        while left <= right:
            mid = (left + right) // 2
            if nums[mid] == target:
                return mid
            elif nums[mid] > target:
                right = mid - 1
            else:
                left = mid + 1
        return left
```

38、外观数列

给定一个正整数 n ，输出外观数列的第 n 项。

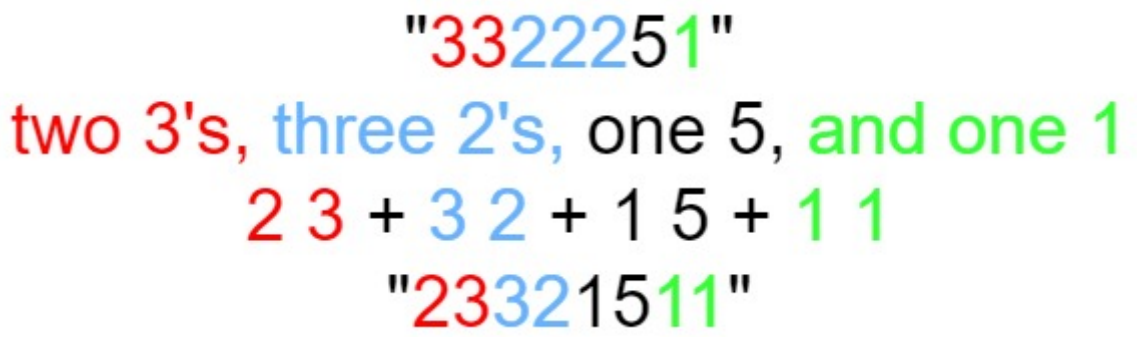
「外观数列」是一个整数序列，从数字 1 开始，序列中的每一项都是对前一项的描述。

你可以将其视作是由递归公式定义的数字字符串序列：

- countAndSay(1) = "1"
- countAndSay(n) 是对 countAndSay(n-1) 的描述，然后转换成另一个数字字符串。

```
1.     1
2.     11
3.     21
4.     1211
5.     111221
第一项是数字 1
描述前一项，这个数是 1 即 “ 一 个 1 ”，记作 "11"
描述前一项，这个数是 11 即 “ 二 个 1 ” ，记作 "21"
描述前一项，这个数是 21 即 “ 一 个 2 + 一 个 1 ” ，记作 "1211"
描述前一项，这个数是 1211 即 “ 一 个 1 + 一 个 2 + 二 个 1 ” ，记作 "111221"
```

要 描述 一个数字字符串，首先要将字符串分割为 最小 数量的组，每个组都由连续的最多 相同字符 组成。然后对于每个组，先描述字符的数量，然后描述字符，形成一个描述组。要将描述转换为数字字符串，先将每组中的字符数量用数字替换，再将所有描述组连接起来。




```
输入：n = 1
输出："1"
解释：这是一个基本样例。
```

```
输入：n = 4
输出："1211"
解释：
countAndSay(1) = "1"
countAndSay(2) = 读 "1" = 一 个 1 = "11"
countAndSay(3) = 读 "11" = 二 个 1 = "21"
countAndSay(4) = 读 "21" = 一 个 2 + 一 个 1 = "12" + "11" = "1211"
```

题解

1、双指针

1. 首先定义变量 pre 记录前一项，初始化为空字符串；定义变量 cur 记录当前项，初始化为 '1'（第一项为 1）；
2. 定义双指针 start， end 均指向序列项的头部，这里用于统计元素出现的次数；
3. 由于给定的n≥1，这里由第 2 项开始逐项对前一项进行描述（注意，要将 cur 赋值给 pre，并初始化 cur 为空字符串，重新拼接得到当前项）：
    - 从左往右遍历 pre，当元素相同时，移动 end 指针，直至元素不相同时，那么此时 end-start 就是相同元素的个数，而 start 指针指向的元素就是重复的元素，进行拼接，cur += str(end-start)+pre[start]。
    - 此时要让 start 指向 end 所在的位置，开始记录下个元素出现的次数；
    - 重复上面的步骤，直至 end 指针到达序列项尾部，便可得到当前项。

4. 逐项对前面一项描述开始时，都应该重置 start、end 指针指向序列项头部，同时应将 cur 赋值给 pre，初始化 cur，也就是前面注意部分所说的内容（可结合代码理解）。然后，再次重复第三个步骤。

```
class Solution:
    def countAndSay(self, n: int) -> str:
        pre = ''
        cur = '1'

        # 从第 2 项开始
        for _ in range(1, n):
            # 这里注意要将 cur 赋值给 pre
            # 因为当前项，就是下一项的前一项。有点绕，尝试理解下
            pre = cur
            # 这里 cur 初始化为空，重新拼接
            cur = ''
            # 定义双指针 start，end
            start = 0
            end = 0
            # 开始遍历前一项，开始描述
            while end < len(pre):
                # 统计重复元素的次数，出现不同元素时，停止
                # 记录出现的次数，
                while end < len(pre) and pre[start] == pre[end]:
                    end += 1
                # 元素出现次数与元素进行拼接
                cur += str(end-start) + pre[start]
                # 这里更新 start，开始记录下一个元素
                start = end

        return cur
```

2、递归

由于每次得到的数据都是来源于上一次的结果，所以我们可以假设得到了上次的结果，继而往后运算。这就运用到了递归。

```
def countAndSay(self, n: int) -> str:
    if n == 1:
        return '1'
    s = self.countAndSay(n-1)

    i, res = 0, ''
    for j, c in enumerate(s):
        if c != s[i]:
            res += str(j-i) + s[i]
            i = j
    res += str(len(s) - i) + s[-1]  # 最后一个元素莫忘统计
    return res
```

3、正则表达式——提取元素

字符串 (\d)\1* 可以用来匹配结果。这里用来提取连在一块的元素， 如 '111221'，提取出的元素是 res = ['111', '22', '1']。
然后再返回我们要组装的字符串。

```python
def countAndSay(self, n: int) -> str:
    if n == 1:
        return '1'
    s = self.countAndSay(n-1)

    p = r'(\d)\1*'
    pattern = re.compile(p)
    res = [_.group() for _ in pattern.finditer(s)]  # 提取结果
    return ''.join(str(len(c)) + c[0] for c in res)  # join 内部的 str(len(c)) +
c[0] for c in res 是生成器类型
```

4、正则表达式——元素替换

对于该句：res = pattern.sub(lambda x: str(len(x.group())) + x.group(1), res)，还是拿上面的举例，对于 111221 的第一个匹配项 111，其中 group() 匹配的是 111（全局匹配），而 group(1) 匹配到的是 1 (第一个捕获组，三个 1 中的第一个)。

```python
def countAndSay(self, n: int) -> str:
    res = '1'
    p = r'(\d)\1*'
    pattern = re.compile(p)
    for _ in range(n-1):
        res = pattern.sub(lambda x: str(len(x.group())) + x.group(1), res)  # 替
换
    return res
```

58、最后一个单词的长度

给你一个字符串 s，由若干单词组成，单词之间用空格隔开。返回字符串中最后一个单词的长度。如果不存在最后一个单词，请返回 0 。

单词 是指仅由字母组成、不包含任何空格字符的最大子字符串。

> 输入：s = "Hello World"
> 输出：5

> 输入：s = " "
> 输出：0

题解

1、使用split()函数，分开列表后输出最后一个单词长度

```python
class Solution:
    def lengthOfLastWord(self, s: str) -> int:
        s_list=s.split(" ")
        #print(s_list)
        for i in range(len(s_list)-1,-1,-1):
            if s_list[i] != '':
                return len(s_list[i])
            else:
                continue

        return 0
```

2、从后往前遍历一次

```python
class Solution:
    def lengthOfLastWord(self, s: str) -> int:
        #计算字符串的长度
        n = len(s)
        #计算最后单词的长度，初值为0
        nums = 0
        #从右向左遍历字符串
        for i in range(n-1,-1,-1):
            #当遍历到的值不为空格时，单词长度+1
            if s[i] != " ":
                nums += 1
            #当遍历到空格时，判断nums是否为0
            else:
                #nums不为0，说明已经遍历完了最后一个单词
                if nums != 0:
                    return nums
                #为0说明还没有遍历到字母
                else:
                    pass
        #如果没有单词返回初值
        return nums
```

11-20

66、加一

给定一个由 整数 组成的 非空 数组所表示的非负整数，在该数的基础上加一。

最高位数字存放在数组的首位， 数组中每个元素只存储单个数字。

你可以假设除了整数 0 之外，这个整数不会以零开头。

> 输入：digits = [1,2,3]
> 输出：[1,2,4]
> 解释：输入数组表示数字 123。

> 输入：digits = [4,3,2,1]
> 输出：[4,3,2,2]
> 解释：输入数组表示数字 4321。

> 输入：digits = [0]
> 输出：[1]

题解

1、从后遍历，判断后位的数字，有两种情况

- 0-8：直接加1，return就可以
- 9：变0，进位

```
class Solution:
    def plusOne(self, digits: List[int]) -> List[int]:
        for i in range(len(digits)-1,-1,-1):
            if digits[i]!=9:
                digits[i]+=1
                return digits
            else:
                digits[i]=0
                continue
        if digits[0]==0:
            digits.insert(0,1)
            return digits
```

2、变成数字，加一，再转换成列表

```
def plusOne(digits: List[int]) -> List[int]:
    num = int(''.join([str(s) for s in digits]))+1
    return [int(s) for s in str(num)]
```

67、二进制求和

给你两个二进制字符串，返回它们的和（用二进制表示）。

输入为 **非空** 字符串且只包含数字 `1` 和 `0`。

> 输入: a = "11", b = "1"
> 输出: "100"

> 输入: a = "1010", b = "1011"
> 输出: "10101"

题解

1、先转十进制，再转二进制

```python
class Solution:
    def addBinary(self, a: str, b: str) -> str:
        ia = int(a, 2)
        ib = int(b, 2)
        isum = ia + ib
        return "{0:b}".format(isum)
```

2、模拟加法

```python
class Solution:
    def addBinary(self, a: str, b: str) -> str:
        r, p = '', 0
        d = len(b) - len(a)
        a = '0' * d + a
        b = '0' * -d + b
        for i, j in zip(a[::-1], b[::-1]):
            s = int(i) + int(j) + p
            r = str(s % 2) + r
            p = s // 2
        return '1' + r if p else r
```

3、递归

- 都是0，不进位
- 有一个是1，不进位
- 要进位

a,b = '1100','110'
addBinary('1100' , '110') -> addBinary('110','11') + '0'
->addBinary('11','1')+'10' -> addBinary(addBinary('1',''),'1') + '010'
->addBinary('1','1')+'010' -> addBinary(addBinary('',''),'1') + '0010'
->'10010'

```python
class Solution:
    def addBinary(self, a: str, b: str) -> str:
        if a == '': return b
        if b == '': return a
        if a[-1] == '1' and b[-1] == '1':
            return self.addBinary(self.addBinary(a[:-1],b[:-1]),'1')+'0'
        elif a[-1] == '0' and b[-1] == '0':
            return self.addBinary(a[:-1],b[:-1])+'0'
        else:
            return self.addBinary(a[:-1],b[:-1])+'1'
```

4、python内置函数

```python
class Solution:
    def addBinary(self, a: str, b: str) -> str:
        return bin(int(a,2)+int(b,2))[2:]
```

69、x的平方根

实现 int sqrt(int x) 函数。

计算并返回 x 的平方根，其中 x 是非负整数。

由于返回类型是整数，结果只保留整数的部分，小数部分将被舍去。

> ```
> 输入：4
> 输出：2
> ```
>
> ```
> 输入：8
> 输出：2
> 说明：8 的平方根是 2.82842...,
>   由于返回类型是整数，小数部分将被舍去。
> ```

题解

1、内置函数

```
class Solution:
    def mySqrt(self, x: int) -> int:
        return int(sqrt(x))
```

2、 循环一次，找到 i*i*<= *x* < *(i+1)*(i+1)

```
class Solution:
    def mySqrt(self, x: int) -> int:
        if x==0 or x==1 :return x
        for i in range(x):
            if i*i <=x < (i+1)*(i+1):
                return i
```

3、 二分查找

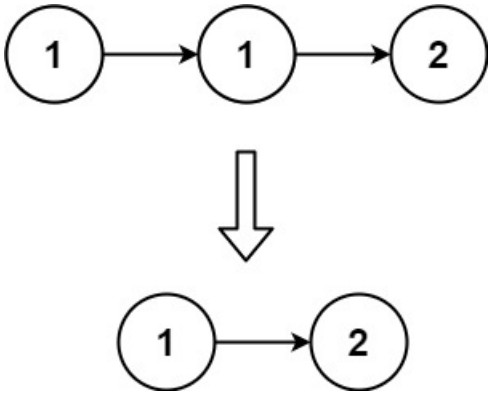
1. 因为x的平方根一定不会大于x的一半（除了1以外，注意本题不算小数部分），因此可令右指针 right = x // 2 + 1，加1即考虑了x为1的情况
2. 当 mid * mid > x时，要减小mid值，令 right = mid - 1
3. 当 mid * mid <= x时，因为mid有可能就是我们想要的结果，因此 left = mid
4. mid = (left + right + 1) // 2，多加了1是为了避免死循环，比如x = 9时，如果没有多加1，会在[3,4]区间内一直出不来
5. 循环条件是left < right，没有 “=” 号，因为两者相等的时候就已经找到了我们想要的结果，不需要再循环了。如果有 “=” 号的话会陷入死循环的，比如x = 1时，最后left = right = 1，有 “=” 号的话就出不了循环了

```python
class Solution:
    def mySqrt(self, x: int) -> int:
        left = 0
        right = x // 2 + 1
        while left < right:
            mid = (left + right + 1) // 2
            if mid * mid > x:
                right = mid - 1
            else:
                left = mid
        return left
```

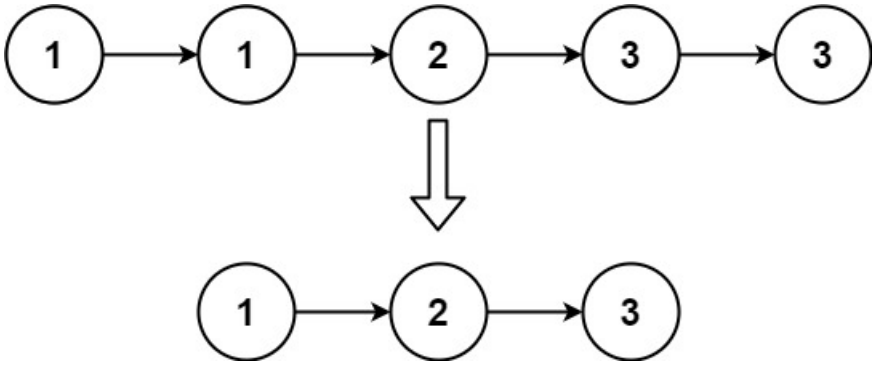
83、删除排序链表中的重复元素

存在一个按升序排列的链表，给你这个链表的头节点 `head` ，请你删除所有重复的元素，使每个元素 **只出现一次** 。

返回同样按升序排列的结果链表




```
输入：head = [1,1,2]
输出：[1,2]
```




```
输入：head = [1,1,2,3,3]
输出：[1,2,3]
```

题解

1、递归，跳过连续相等的元素

- 如果 head.val != head.next.val ，说明头节点的值不等于下一个节点的值，所以当前的 head 节点必须保留；但是 head.next 节点要不要保留呢？我们还不知道，需要对 head.next 进行递归，即对 head.next 作为头节点的链表做处理，使值相等的节点仅保留一个。然后我们看到 self.deleteDuplicates(head) 函数就是做这个事情的！所以 head.next = self.deleteDuplicates(head.next)，这就是递归调用的由来。
- 如果 head.val == head.next.val ，说明头节点的值 等于 下一个节点的值，所以当前的 head 节点必须删除，删除到哪个节点为止呢？按照函数定义，我们保留值相等的各个节点中最后一个节点，所以 head 到与 head.val 相等的最后一个节点之间的节点也都需要删除；需要用 move 指针一直向后遍历寻找到最后一个与 head.val 相等的节点。此时 move 之前的节点都不保留了，因此返回 deleteDuplicates(move);

```
class Solution(object):
    def deleteDuplicates(self, head):
        if not head or not head.next:
            return head
        if head.val != head.next.val:
            head.next = self.deleteDuplicates(head.next)
        else:
            move = head.next
            while move.next and head.val == move.next.val:
                move = move.next
            return self.deleteDuplicates(move)
        return head
```

2、递归，删除下一个相等的元素

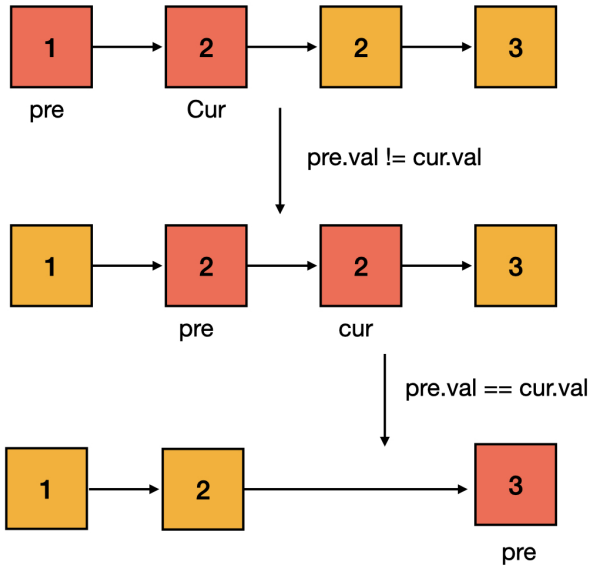
- 无论 head.val 和 head.next.val 是否相等，head.next 一定等于后续链表的去重，即 self.deleteDuplicates(head.next) 。原因是：
    - 当 head.val != head.next.val 时，head 节点要保留，所以 head.next = self.deleteDuplicates(head.next) 比较好理解，因为要把后面去重的链表拼接到当前 head 节点之后；
    - 当 head.val == head.next.val 时， head 节点要删除，所以 return head.next ，如果我们不把 head.next = self.deleteDuplicates(head.next)， 那么 return 的结果是原始 head.next ，所以仍然是没去重。

```
class Solution(object):
    def deleteDuplicates(self, head):
        if not head or not head.next: return head
        head.next = self.deleteDuplicates(head.next)
        return head if head.val != head.next.val else head.next
```

3、一次遍历

- 一次遍历的方法比较好想，有两种方法：一种是遇到值相等的节点保留第一个节点；另一种是遇到值相等的节点保留最后一个节点。保留第一个节点的方法更简单，我这里使用保留第一个节点的做法。
- 使用两个指针 pre 和 cur：pre 节点表示固定一个节点，cur 用于寻找所有跟 pre 值相等的节点。如果 cur.val 等于 pre.val，则删除 cur。在找到不等的 val 之前，pre 不走，只移动 cur。

制图：负雪明烛




```
class Solution(object):
    def deleteDuplicates(self, head):
        if not head: return None
        prev, cur = head, head.next
        while cur:
            if cur.val == prev.val:
                prev.next = cur.next
            else:
                prev = cur
            cur = cur.next
        return head
```

4、利用 set 保存出现过的元素

使用了两次遍历，第一次遍历统计每个节点的值出现的次数，第二次遍历的时候，如果发现 head.next 的 val 出现次数不是 1 次，则需要删除 head.next。

```
class Solution:
    def deleteDuplicates(self, head):
        if not head or not head.next: return head
        val_set = set()
        val_set.add(head.val)
        root = ListNode(0)
        root.next = head
        while head and head.next:
            if head.next.val in val_set:
                head.next = head.next.next
            else:
                head = head.next
                val_set.add(head.val)
        return root.next
```

88、合并两个有序数组

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。

初始化 nums1 和 nums2 的元素数量分别为 m 和 n 。你可以假设 nums1 的空间大小等于 m + n，这样它就有足够的空间保存来自 nums2 的元素。

> 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]

> 输入：nums1 = [1], m = 1, nums2 = [], n = 0
> 输出：[1]

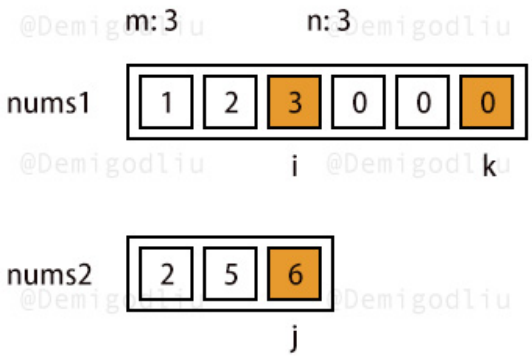
题解

1、 删除nums1后面0，拼接nums1和nums2，sort()函数排序就好

```
class Solution:
    def merge(self, nums1: List[int], m: int, nums2: List[int], n: int) -> None:
        """
        Do not return anything, modify nums1 in-place instead.
        """
        for i in range(n):
            nums1.pop()
        nums1.extend(nums2)
        nums1.sort()

        #更简单的方法
        nums1[m:] = nums2
        nums1.sort()
```

2、逆向双指针动画演示

总共需要创建三个指针，两个指针用于指向 ums1ums1 和 nums2nums2 的初始化元素数量的末位，也就是分别指向 m-1m−1 和 n-1n−1 的位置，还有一个指针，我们指向 nums1nums1 数组末位即可。



创建 i 指针，j 指针，k 指针，放置对应位置

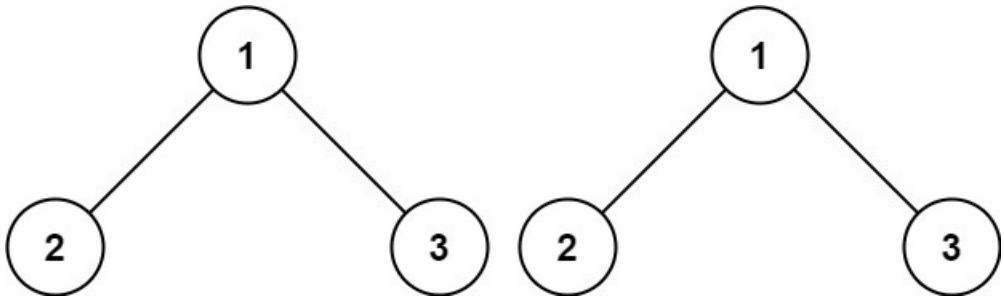
```python
class Solution:
    def merge(self, nums1: List[int], m: int, nums2: List[int], n: int) -> None:
        """
        Do not return anything, modify nums1 in-place instead.
        """
        p1, p2 = m - 1, n - 1
        tail = m + n - 1
        while p1 >= 0 or p2 >= 0:
            if p1 == -1:
                nums1[tail] = nums2[p2]
                p2 -= 1
            elif p2 == -1:
                nums1[tail] = nums1[p1]
                p1 -= 1
            elif nums1[p1] > nums2[p2]:
                nums1[tail] = nums1[p1]
                p1 -= 1
            else:
                nums1[tail] = nums2[p2]
                p2 -= 1
            tail -= 1
```

```python
class Solution(object):
    def merge(self, nums1, m, nums2, n):
        k = m + n - 1
        while m > 0 and n > 0:
            if nums1[m - 1] > nums2[n - 1]:
                nums1[k] = nums1[m - 1]
                m -= 1
            else:
                nums1[k] = nums2[n - 1]
                n -= 1
            k -= 1
        nums1[:n] = nums2[:n]
```

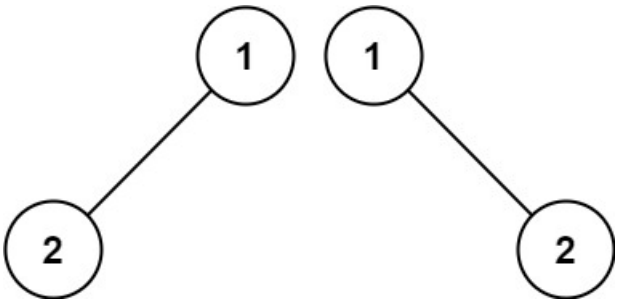
100、相同的树

给你两棵二叉树的根节点 `p` 和 `q` ，编写一个函数来检验这两棵树是否相同。

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。




```
输入：p = [1,2,3], q = [1,2,3]
输出：true
```




```
输入：p = [1,2], q = [1,null,2]
输出：false
```

题解

1、DFS

- 如果两节点都为空，返回true；
- 如果两节点一个为空一个不为空，返回false；
- 如果两节点值不相同，返回false
- 如果两个节点值相同，比较左子树和右子树是否相同，这就进入了递归

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

2、BFS

- 特例处理：如果两根节点都为空，返回true；如果两根节点一个为空一个不为空，返回false

- 用两个队列分别存储p树和q树的节点，只要两个队列都非空就进入循环
- 循环中，先弹出两个队列的节点，如果值不同，直接返回false
- 接下来比较俩节点的子节点情况，如果俩节点的左子节点和右子节点没有分别都存在或都不存在，返回false
- 存在的子节点们分别入队
- 循环结束后，只有当两个队列都为空时才会返回true

```
class Solution:
    def isSameTree(self, p: TreeNode, q: TreeNode) -> bool:
        if not p and not q:
            return True
        if not p or not q:
            return False

        queueP = [p]
        queueQ = [q]
        while queueP and queueQ:
            nodeP = queueP.pop(0)
            nodeQ = queueQ.pop(0)
            if nodeP.val != nodeQ.val:
                return False

            leftP, rightP = nodeP.left, nodeP.right
            leftQ, rightQ = nodeQ.left, nodeQ.right
            if (not leftP) ^ (not leftQ):
                return False
            if (not rightP) ^ (not rightQ):
                return False
            if leftP:
                queueP.append(leftP)
            if leftQ:
                queueQ.append(leftQ)
            if rightP:
                queueP.append(rightP)
            if rightQ:
                queueQ.append(rightQ)

        return not queueP and not queueQ
```

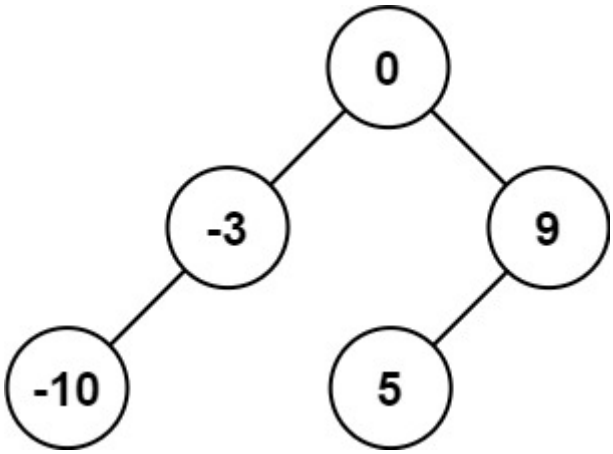
3、各种遍历算法，对比数组

```
class Solution:
    def isSameTree(self, p: TreeNode, q: TreeNode) -> bool:
        def preorder(root):
            if not root:
                return [None]
            else:
                return [root.val] +preorder(root.left)+preorder(root.right)
        return preorder(p)==preorder(q)
```

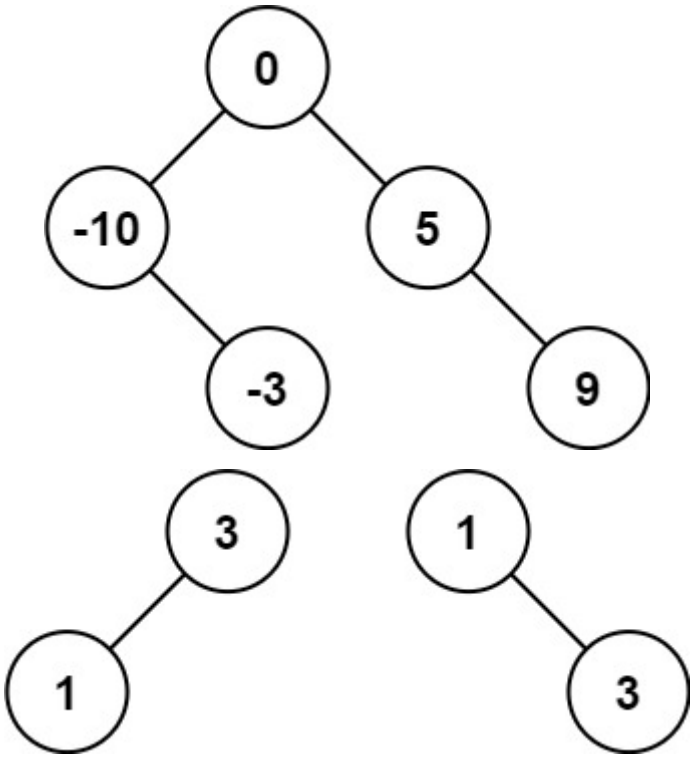
108、将有序数组转换为二叉搜索树

给你一个整数数组 nums ，其中元素已经按 升序 排列，请你将其转换为一棵 高度平衡 二叉搜索树。

高度平衡 二叉树是一棵满足「每个节点的左右两个子树的高度差的绝对值不超过 1 」的二叉树。



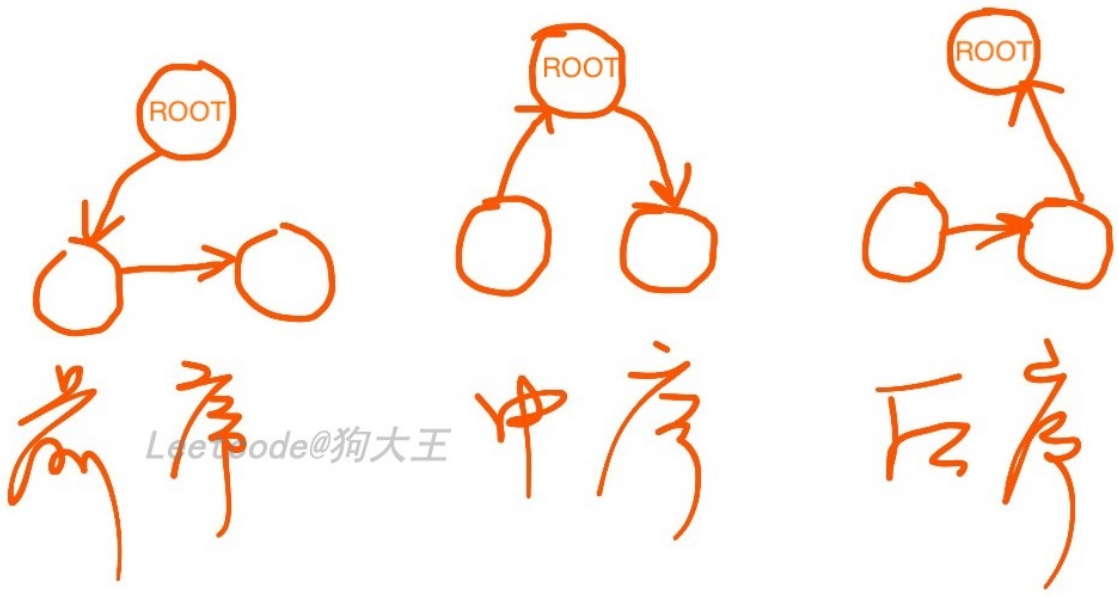

```
输入：nums = [-10,-3,0,5,9]
输出：[0,-3,9,-10,null,5]
解释：[0,-10,5,null,-3,null,9] 也将被视为正确答案：
```




```
输入：nums = [1,3]
输出：[3,1]
解释：[1,3] 和 [3,1] 都是高度平衡二叉搜索树。
```

题解

1、 中续遍历



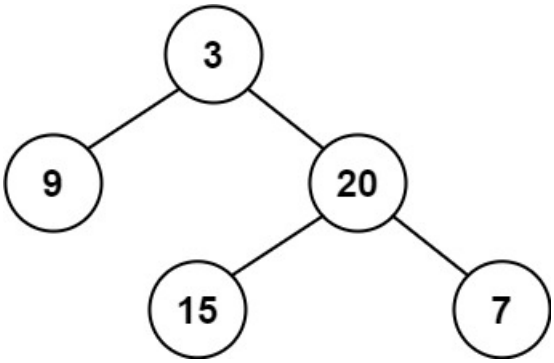

```python
class Solution:
    def sortedArrayToBST(self, nums: List[int]) -> TreeNode:

        def make_tree(start_index, end_index): #只和长度有关
            #首先判定我们的区间是否合理，即left_index要<=right_index
            #当相等时，只有root会产生，不会产生左右小树
            if start_index > end_index:
                return None

            #我这里变量名都写得比较长，目的是方便理解
            mid_index = (start_index + end_index)//2
            this_tree_root = TreeNode(nums[mid_index]) #做一个小树的root

            this_tree_root.left = make_tree(start_index,mid_index-1)
            this_tree_root.right = make_tree(mid_index+1, end_index)

            return this_tree_root #做好的小树

        return make_tree(0,len(nums)-1)
        #可以看到整个题解只和index有关，和数组里的具体数字无关，
        #因为题目给出的“有序数列”帮助我们满足了“二叉搜索树”的条件。
```

110、平衡二叉树

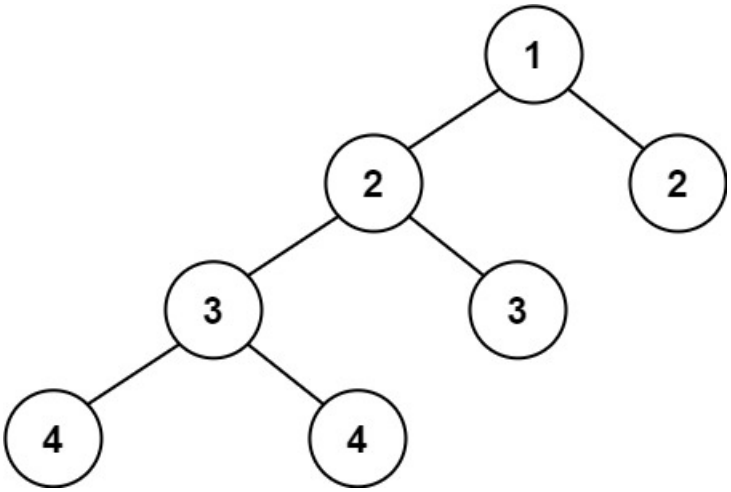
给定一个二叉树，判断它是否是高度平衡的二叉树。

本题中，一棵高度平衡二叉树定义为：

一个二叉树*每个节点*的左右两个子树的高度差的绝对值不超过 1 。




```
输入：root = [3,9,20,null,null,15,7]
输出：true
```




```
输入：root = [1,2,2,3,3,null,null,4,4]
输出：false
```

题解

1、 自底至顶

`recur(root):`

- 递归返回值：
  1. 当节点root 左 / 右子树的高度差 <2 ： 则返回以节点root为根节点的子树的最大高度，即节点 root 的左右子树中最大高度加 1 （ max(left, right) + 1 ）；
  2. 当节点root 左 / 右子树的高度差 ≥2 ： 则返回 -1−1 ，代表 此子树不是平衡树 。
- 递归终止条件：
  1. 当越过叶子节点时，返回高度 0；
  2. 当左（右）子树高度 left== -1 时，代表此子树的 左（右）子树 不是平衡树，因此直接返回 -1 ；

`isBalanced(root)` ：

- 返回值： 若 recur(root) != 1 ，则说明此树平衡，返回 true； 否则返回 false 。

```python
class Solution:
    def isBalanced(self, root: TreeNode) -> bool:
        return self.recur(root) != -1

    def recur(self, root):
        if not root: return 0
        left = self.recur(root.left)
        if left == -1: return -1
        right = self.recur(root.right)
        if right == -1: return -1
        return max(left, right) + 1 if abs(left - right) < 2 else -1
```

2、 自顶至底

构造一个获取当前节点最大深度的方法 depth(root) ，通过比较此子树的左右子树的最大高度差 abs(depth(root.left) - depth(root.right))，来判断此子树是否是二叉平衡树。若树的所有子树都平衡时，此树才平衡。

`isBalanced(root)` ： 判断树 root 是否平衡

- 特例处理： 若树根节点 root 为空，则直接返回 truetrue ；
- 返回值： 所有子树都需要满足平衡树性质，因此以下三者使用与逻辑 &&&& 连接；
    1. abs(self.depth(root.left) - self.depth(root.right)) <= 1 ： 判断 当前子树 是否是平衡树；
    2. self.isBalanced(root.left)： 先序遍历递归，判断 当前子树的左子树 是否是平衡树；
    3. self.isBalanced(root.right)： 先序遍历递归，判断 当前子树的右子树 是否是平衡树；

`depth(root)` ： 计算树 root 的最大高度

- 终止条件： 当 root 为空，即越过叶子节点，则返回高度 00 ；
- 返回值： 返回左 / 右子树的最大高度加 11 。

```python
class Solution:
    def isBalanced(self, root: TreeNode) -> bool:
        if not root: return True
        return abs(self.depth(root.left) - self.depth(root.right)) <= 1 and \
            self.isBalanced(root.left) and self.isBalanced(root.right)

    def depth(self, root):
        if not root: return 0
        return max(self.depth(root.left), self.depth(root.right)) + 1
```

3、DFS，同二

- 从第一层开始，如果下一层为None 为 0
- 对于某一层的高度为max(left, right)
- 对于左子树和右子树的高度差大于1，说明非高度平衡的二叉树，否者是高度平衡的二叉树

```python
class Solution:
    def isBalanced(self, root: TreeNode) -> bool:
        self.res = True
        def helper(root):
            if not root:
                return 0

            left = helper(root.left) + 1
            right = helper(root.right) + 1
```

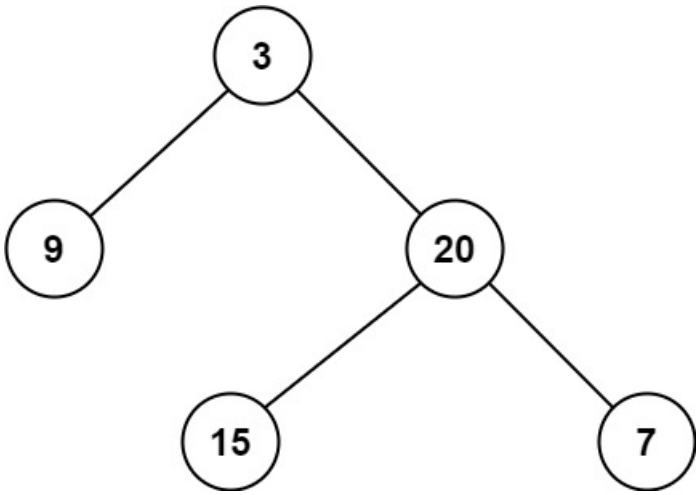
```
            if abs(right - left) > 1:
                self.res = False
            return max(left, right)
        helper(root)
        return self.res
```

111、二叉树的最小深度

给定一个二叉树，找出其最小深度。

最小深度是从根节点到最近叶子节点的最短路径上的节点数量。

说明：叶子节点是指没有子节点的节点。




```
输入：root = [3,9,20,null,null,15,7]
输出：2
```

```
输入：root = [2,null,3,null,4,null,5,null,6]
输出：5
```

题解

1、DFS

```
#开始还是DFS的迭代方法，就当左右节点都是NONE的时候就是一个子叶结点，这个时候就往上推，
#如果左右节点不全为一，就选择其中较大的，如果都是0或者都不为零，就去其中较小的哪一个，直达推到根节点。
class Solution:
    def minDepth(self, root: TreeNode):
        if not root: return 0
        left = self.minDepth(root.left)
        right = self.minDepth(root.right)
        if left ==0 and right==0:
            return min(left,right)+1
        elif (left == 0 and right!=0) or (left!=0 and right==0):
            return max(left,right)+1
        else:
            return min(left,right)+1
```

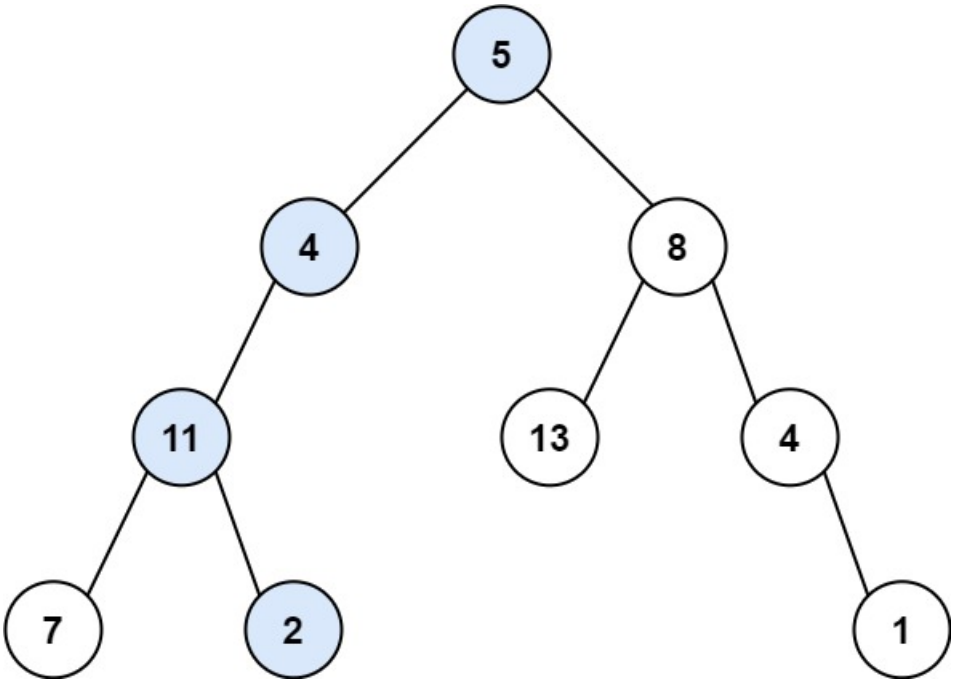
2、BFS

```
#一层一层的遍历，一旦发现了某一层有个节点的没有左节点也没有右节点，
#就停止计数，这一层就是最小的叶子节点的深度
class Solution:
    def minDepth(self, root: TreeNode):
        if not root: return 0
        res = 0
        queue = deque()
        queue.append(root)
        while queue:
            res += 1
            for _ in range(len(queue)):
                cur = queue.popleft()
                if cur.left: queue.append(cur.left)
                if cur.right : queue.append(cur.right)
                if not cur.left and not cur.right: return res
        return res
```

112、路径总和

给你二叉树的根节点 root 和一个表示目标和的整数 targetSum ，判断该树中是否存在 根节点到叶子节点 的路径，这条路径上所有节点值相加等于目标和 targetSum 。

叶子节点 是指没有子节点的节点。




```
输入：root = [5,4,8,11,null,13,4,7,2,null,null,null,1], targetSum = 22
输出：true
```

```
输入：root = [1,2,3], targetSum = 5
输出：false
```

题解

1、 DFS

```
class Solution(object):
    def hasPathSum(self, root, sum):
        """
        :type root: TreeNode
        :type sum: int
        :rtype: bool
        """
        if not root: return False
        if not root.left and not root.right:        #判断是否为叶子节点
            return sum == root.val
        return self.hasPathSum(root.left, sum - root.val) or
self.hasPathSum(root.right, sum - root.val)
```

2、回溯

这里的回溯指 利用 DFS 找出从根节点到叶子节点的所有路径，只要有任意一条路径的 和 等于 sum，就返回 True。

下面的代码并非是严格意义上的回溯法，因为没有重复利用 path 变量。

```
class Solution(object):
    def hasPathSum(self, root, sum):
        """
        :type root: TreeNode
        :type sum: int
```

```
        :rtype: bool
        """
        if not root: return False
        res = []
        return self.dfs(root, sum, res, [root.val])

    def dfs(self, root, target, res, path):
        if not root: return False
        if sum(path) == target and not root.left and not root.right:
            return True
        left_flag, right_flag = False, False
        if root.left:
            left_flag = self.dfs(root.left, target, res, path + [root.left.val])
        if root.right:
            right_flag = self.dfs(root.right, target, res, path +
[root.right.val])
        return left_flag or right_flag
```

3、BFS

使用 **队列** 保存遍历到每个节点时的**路径和**，如果该节点恰好是叶子节点，并且 路径和 正好等于 sum，说明找到了解。

```
class Solution:
    def hasPathSum(self, root: TreeNode, sum: int) -> bool:
        if not root:
            return False
        que = collections.deque()
        que.append((root, root.val))
        while que:
            node, path = que.popleft()
            if not node.left and not node.right and path == sum:
                return True
            if node.left:
                que.append((node.left, path + node.left.val))
            if node.right:
                que.append((node.right, path + node.right.val))
        return False
```

4、栈

同时保存节点和到这个节点的路径和。但是这个解法已经不是 BFS。因为会优先访问 后进来 的节点，导致会把根节点的右子树访问结束之后，才访问左子树。

栈中同时保存了 (节点，路径和)，也就是说只要能把所有的节点访问一遍，那么就一定能找到正确的结果。无论是用 队列 还是 栈，都是一种 树的遍历 方式，只不过访问顺序有所有不同罢了。

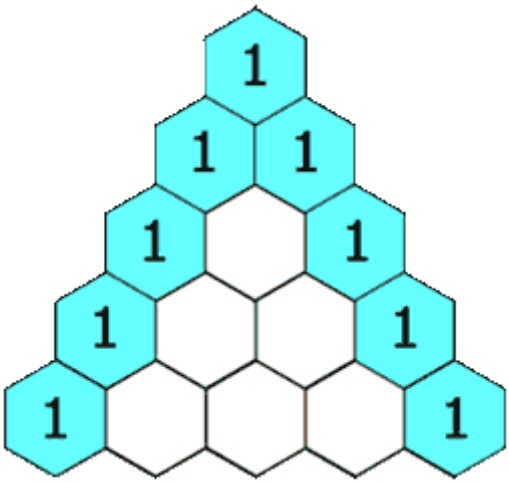
```
class Solution(object):
    def hasPathSum(self, root, sum):
        """
        :type root: TreeNode
        :type sum: int
        :rtype: bool
        """
        if not root:
            return False
        stack = []
```

```
        stack.append((root, root.val))
        while stack:
            node, path = stack.pop()
            if not node.left and not node.right and path == sum:
                return True
            if node.left:
                stack.append((node.left, path + node.left.val))
            if node.right:
                stack.append((node.right, path + node.right.val))
        return False
```

21-30

118、杨辉三角

给定一个非负整数 *numRows*，生成杨辉三角的前 *numRows* 行。




```
输入: 5
输出:
[
  [1],
 [1,1],
[1,2,1],
[1,3,3,1],
[1,4,6,4,1]]
```

题解

1、特殊情况，为1为2时，，然后就是遍历上一行

```
class Solution:
    def generate(self, numRows: int) -> List[List[int]]:
        res=[]
        if numRows==1: return [[1]]
        if numRows==2: return [[1],[1,1]]
        else:
            res=[[1],[1,1]]
            for i in range(3,numRows+1):
```

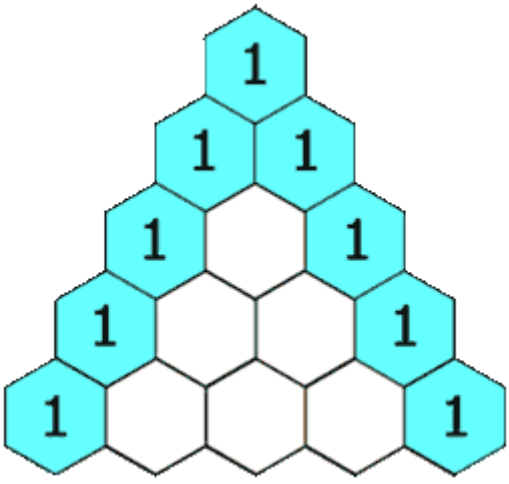
```
                num=[1]*i
                for j in range(1,i-1):
                    num[j]=res[i-2][j-1]+res[i-2][j]
                res.append(num)

        return res
```

2、数学做法

```
class Solution:
    def generate(self, numRows: int) -> List[List[int]]:

        # 运用杨辉三角的公式:第n行第m个数可表示为C(n-1, m-1) = (n - 1)!/(m - 1)!*(n - m - 2)!

        res = [[] for _ in range(numRows)] # 用列表推导式初始化
        if not numRows: return [] # 如果numRows == 0则直接返回[]

        from math import factorial as f # 导入math库中的计算阶乘的方法factorial，并简写为f

        for i in range(numRows): # 遍历每一行
            for j in range(i + 1): # 遍历每一行的数
                res[i].append(f(i)//(f(j)*f(i - j)))
                """
                注意这里直接用i,j就行了,而不是杨辉三角公式所示的n - 1, m -1。
                因为在python语言中数字是从0开始的
                """
        return res
```

119、杨辉三角II

给定一个非负索引 *k*，其中 *k* ≤ 33，返回杨辉三角的第 *k* 行。



> 输入: 3
> 输出: [1,3,3,1]

题解

1、同上

```
class Solution:
    def getRow(self, rowIndex: int) -> List[int]:
        res=[[1],[1,1]]
        if rowIndex==0:return [1]
        if rowIndex==1:return [1,1]
        for i in range(3,rowIndex+2):
            num=[1]*i
            for j in range(1,i-1):
                num[j]=res[i-2][j-1]+res[i-2][j]
            res.append(num)

        return res[-1]
```

2、优化空间

```
class Solution(object):
    def getRow(self, rowIndex):
        """
        :type rowIndex: int
        :rtype: List[int]
        """
        res = [1] * (rowIndex + 1)
        for i in range(2, rowIndex + 1):
            for j in range(i - 1, 0, -1):
                res[j] += res[j - 1]
        return res
```

122、买卖股票的最佳时机II

给定一个数组 prices ，其中 prices[i] 是一支给定股票第 i 天的价格。

设计一个算法来计算你所能获取的最大利润。你可以尽可能地完成更多的交易（多次买卖一支股票）。

注意：你不能同时参与多笔交易（你必须在再次购买前出售掉之前的股票）。

> 输入: prices = [7,1,5,3,6,4]
> 输出: 7
> 解释: 在第 2 天（股票价格 = 1）的时候买入，在第 3 天（股票价格 = 5）的时候卖出，这笔交易所能获得利润 = 5-1 = 4 。
> 随后，在第 4 天（股票价格 = 3）的时候买入，在第 5 天（股票价格 = 6）的时候卖出，这笔交易所能获得利润 = 6-3 = 3 。

> 输入: prices = [1,2,3,4,5]
> 输出: 4
> 解释: 在第 1 天（股票价格 = 1）的时候买入，在第 5 天 （股票价格 = 5）的时候卖出，这笔交易所能获得利润 = 5-1 = 4 。
> 注意你不能在第 1 天和第 2 天接连购买股票，之后再将它们卖出。因为这样属于同时参与了多笔交易，你必须在再次购买前出售掉之前的股票。

题解

1、动态规划

1. 本题在交易股票的过程中，一共会有2种状态：
    - dp0：手里没股票
    - dp1：手里有股票
2. 初始化2种状态：
    - dp0 = 0
    - dp1 = - prices[0]
3. 对2种状态进行状态转移：
    - dp0 = max(dp0, dp1 + prices[i])
      前一天也是dp0状态，或者前一天是dp1状态，今天卖出一笔变成dp0状态
    - dp1 = max(dp1, dp0 - prices[i])
      前一天也是dp1状态，或者前一天是dp0状态，今天买入一笔变成dp1状态
4. 最后一定是手里没有股票赚的钱最多，因此返回的是dp0

```
class Solution:
    def maxProfit(self, prices: List[int]) -> int:
        dp0 = 0             # 手里没股票
        dp1 = - prices[0]   # 手里有股票
        for i in range(1, len(prices)):
            dp0 = max(dp0, dp1 + prices[i])
            dp1 = max(dp1, dp0 - prices[i])
        return dp0
```

2、单调栈

- 计算所有增区间的差值之和，利用递减单调栈寻找拐点，
- 当当前元素大于栈顶时，说明开始递增，增值= 当前元素-栈顶元素，
- 清空单调栈，当前元素入栈

```
class Solution:
    def maxProfit(self, prices: List[int]) -> int:
        if not prices:
            return 0
        s = prices[0]
        count = 0
        for i in range(len(prices)):
            if prices[i] > s:
                count += prices[i] -s
            s = prices[i]
        return count
```

125、验证回文串

给定一个字符串，验证它是否是回文串，只考虑字母和数字字符，可以忽略字母的大小写。

说明：本题中，我们将空字符串定义为有效的回文串。

```
输入: "A man, a plan, a canal: Panama"
输出: true
```

```
输入: "race a car"
输出: false
```

题解

1，删除s中标点符号和空格，再转换成小写

```
class Solution:
    def isPalindrome(self, s: str) -> bool:
        s=s.lower().replace(" ",'')
        import string
        punctuation_string = string.punctuation
        for i in punctuation_string:
            s=s.replace(i,'')

        return s==s[::-1]
```

2、将字母和数字提取出来

`str.isalnum()` 这个方法，检测字符串是否由字母和数字组成。

```
class Solution:
    def isPalindrome(self, s):
        ret = []
        for i in s:
            if i.isalnum():
                ret.append(i.lower())
        return ret == ret[::-1]
```

总结处理字符串的相关函数

1. `string.capitalize()` 把字符串的第一个字符大写
2. `string.count(str, beg=0, end=len(string))` 返回 str 在 string 里面出现的次数，如果 beg 或者 end 指定则返回指定范围内 str 出现的次数
3. `string.endswith(obj, beg=0, end=len(string))` 检查字符串是否以 obj 结束，如果beg 或者 end 指定则检查指定的范围内是否以 obj 结束，如果是，返回 True,否则返回 False.
4. `string.find(str, beg=0, end=len(string))` 检测 str 是否包含在 string 中，如果 beg 和 end 指定范围，则检查是否包含在指定范围内，如果是返回开始的索引值，否则返回-1
5. `string.index(str, beg=0, end=len(string))` 跟find()方法一样，只不过如果str不在 string 中会报一个异常.
6. `string.isalnum()` 如果 string 至少有一个字符并且所有字符都是字母或数字则返回 True,否则返回 False
7. `string.isalpha()` 如果 string 至少有一个字符并且所有字符都是字母则返回 True,否则返回 False
8. `string.isdecimal()` 如果 string 只包含十进制数字则返回 True 否则返回 False.
9. `string.isdigit()` 如果 string 只包含数字则返回 True 否则返回 False.
10. `string.islower()` 如果 string 中包含至少一个区分大小写的字符，并且所有这些(区分大小写的)字符都是小写，则返回 True，否则返回 False
11. `string.isnumeric()` 如果 string 中只包含数字字符，则返回 True，否则返回 False

12. `string.isspace()` 如果 string 中只包含空格，则返回 True，否则返回 False.
13. `string.istitle()` 如果 string 是标题化的(见 title())则返回 True，否则返回 False
14. `string.isupper()` 如果 string 中包含至少一个区分大小写的字符，并且所有这些(区分大小写的)字符都是大写，则返回 True，否则返回 False
15. `string.join(seq)` 以 string 作为分隔符，将 seq 中所有的元素(的字符串表示)合并为一个新的字符串
16. `string.lower()` 转换 string 中所有大写字符为小写.
17. `string.lstrip()` 截掉 string 左边的空格
18. `max(str)` 返回字符串 str 中最大的字母。
19. `min(str)` 返回字符串 str 中最小的字母。
20. `string.replace(str1, str2, num=string.count(str1))` 把 string 中的 str1 替换成 str2,如果 num 指定，则替换不超过 num 次.
21. `string.split(str="", num=string.count(str))` 以 str 为分隔符切片 string，如果 num 有指定值，则仅分隔 num+ 个子字符串
22. `string.startswith(obj, beg=0,end=len(string))` 检查字符串是否是以 obj 开头，是则返回 True，否则返回 False。如果beg 和 end 指定值，则在指定范围内检查.
23. `string.strip([obj])` 在 string 上执行 lstrip()和 rstrip()
24. `string.swapcase()` 翻转 string 中的大小写
25. `string.title()` 返回"标题化"的 string,就是说所有单词都是以大写开始，其余字母均为小写(见 istitle())
26. `string.translate(str, del="")` 根据 str 给出的表(包含 256 个字符)转换 string 的字符,要过滤掉的字符放到 del 参数中
27. `string.upper()` 转换 string 中的小写字母为大写

167、两数之和Ⅱ-输入有序数组

给定一个已按照 升序排列 的整数数组 numbers ，请你从数组中找出两个数满足相加之和等于目标数 target 。

函数应该以长度为 2 的整数数组的形式返回这两个数的下标值。numbers 的下标 从 1 开始计数 ，所以答案数组应当满足 1 <= answer[0] < answer[1] <= numbers.length 。

你可以假设每个输入只对应唯一的答案，而且你不可以重复使用相同的元素。

> ```
> 输入：numbers = [2,7,11,15], target = 9
> 输出：[1,2]
> 解释：2 与 7 之和等于目标数 9 。因此 index1 = 1, index2 = 2 。
> ```
>
> ```
> 输入：numbers = [2,3,4], target = 6
> 输出：[1,3]
> ```

题解

1、 遍历，查找两个元素相加等于target——超时间

```
class Solution:
    def twoSum(self, numbers: List[int], target: int) -> List[int]:
        for i in range(len(numbers)-1):
            for j in range(i+1,len(numbers)):
                if numbers[i]+numbers[j]==target:
                    return [i+1,j+1]
```

2、双指针

左右各一个指针，向中间收缩

- 相等则返回下标
- 相加大于target，右指针-1
- 相加小于target，做指针+1

```
class Solution:
    def twoSum(self, numbers: List[int], target: int) -> List[int]:
        left=0
        right=len(numbers)-1
        while left<right:
            if numbers[left]+numbers[right]==target:
                return [left+1,right+1]
            elif numbers[left]+numbers[right]<target:
                left+=1
            else:
                right-=1
```

3、二分查找

首先找到一个数，对另一个数的查找使用二分查找

```
class Solution:
    def twoSum(self, numbers: List[int], target: int) -> List[int]:
        for i in range(len(numbers)):
            target1 = target - numbers[i]
            left , right = i+1 , len(numbers) - 1
            while left <= right:
                mid = (left+right)//2
                if target1 < numbers[mid]:
                    right = mid - 1
                elif target1 > numbers[mid]:
                    left = mid + 1
                else:
                    return [i+1,mid+1]
```

4、哈希表

- [target-num]作为dict的键，index作为值。
- 这样if判断只需要判断num是否在dict中，
    - 如果在的话，通过dict[num]找到之前的num对应的index，那么就可以返回`[dict[num]+1,index+1],`
    - 不在的话就更新字典，target-num代表着下一个需要的数字，而迭代的num就是下一个数字，以此来判断。

```python
class Solution:
    def twoSum(self, numbers: List[int], target: int) -> List[int]:
        dict = {}
        for index,num in enumerate(numbers):
            if num in dict:
                return [dict[num]+1,index+1]
            dict[target-num] = index
```

168、Excel表列名称

给定一个正整数，返回它在 Excel 表中相对应的列名称。

1 -> A
2 -> B
3 -> C
...
26 -> Z
27 -> AA
28 -> AB

> 输入：1
> 输出："A"
>
> 输入：28
> 输出："AB"
>
> 输入：701
> 输出："ZY"

题解

1、 二十六进制转换

- 注意：1对应A，而不是0对应A
- 所以每次转化前需要：n-=1
- chr() 用一个范围在 range（256）内的（就是0～255）整数作参数，返回一个对应的字符。

```python
class Solution:
    def convertToTitle(self, n: int) -> str:
        s = ''
        while n:
            n -= 1
            #ASCII码转大写字符 并且左加
            s = chr(65 + n % 26) + s
            n //= 26
        return s
```

2、递归写法

```python
class Solution:
    def convertToTitle(self, n: int) -> str:
        return "" if n == 0 else self.convertToTitle((n - 1) // 26) + chr((n - 1) % 26 + 65)
```

171、Excel表列序号

给定一个Excel表格中的列名称，返回其相应的列序号。

A -> 1
B -> 2
C -> 3
...
Z -> 26
AA -> 27
AB -> 28

> ```
> 输入: "A"
> 输出: 1
> ```
>
> ```
> 输入: "AB"
> 输出: 28
> ```

题解

1、二十六进制转十进制，从后往前遍历

ord() 函数：它以一个字符（长度为1的字符串）作为参数，返回对应的 ASCII 数值

```python
class Solution:
    def titleToNumber(self, s: str) -> int:
        return sum(26**i*(ord(j)-64) for i, j in enumerate(s[::-1]))
```

172、阶乘后的零

给定一个整数 *n*，返回 *n*! 结果尾数中零的数量。

> ```
> 输入: 3
> 输出: 0
> 解释: 3! = 6, 尾数中没有零。
> ```
>
> ```
> 输入: 5
> 输出: 1
> 解释: 5! = 120, 尾数中有 1 个零.
> ```

题解

1、判断为0的条件，其实就是5的个数，但同时，25=5*5，25中包含两个5，

可以循环对n取5, 25, 125...的商，将所有的情况都包括，最终将所有的商汇总即0的个数。

```
class Solution:
    def trailingZeroes(self, n: int) -> int:
        p = 0
        while n >= 5:
            n = n // 5
            p += n
        return p
```

2、递归写法

```
class Solution:
    def trailingZeroes(self, n: int) -> int:
        return 0 if not n else self.trailingZeroes(n//5)+n//5
```

176、颠倒二进制位

颠倒给定的 32 位无符号整数的二进制位。

> 输入：00000010100101000001111010011100
> 输出：00111001011110000010100101000000
> 解释：输入的二进制串 00000010100101000001111010011100 表示无符号整数 43261596，
> 因此返回 964176192，其二进制表示形式为 00111001011110000010100101000000。

> 输入：11111111111111111111111111111101
> 输出：10111111111111111111111111111111
> 解释：输入的二进制串 11111111111111111111111111111101 表示无符号整数 4294967293，
> 因此返回 3221225471 其二进制表示形式为 10111111111111111111111111111111

题解

1、循环

每次把 `res` 左移，把 n的二进制末尾数字，拼接到结果 `res` 的末尾。然后把 n 右移。

```
class Solution:
    def reverseBits(self, n):
        res = 0
        for i in range(32):
            res = (res << 1) | (n & 1)
            n >>= 1
        return res
```

2、换成字符，再转回二进制

zfill() 方法返回指定长度的字符串，原字符串右对齐，前面填充0。

```python
class Solution:
    def reverseBits(self, n):
        # 1. 首先我们获取n的二进制
        # '0b10100101000001111010011100'
        bin_n = bin(n)
        print(bin_n)
        # 2. 接下来我们将'0b'替换为完整的全零前缀
        # 3. 然后将tmp_n倒置
        tmp_n = bin_n[2:].zfill(32)[::-1]
        # 4. 最后我们将tmp_n转换为整数返回
        ret = int(tmp_n,2)
        return ret
```

191、位1的个数

编写一个函数，输入是一个无符号整数（以二进制串的形式），返回其二进制表达式中数字位数为 '1' 的个数（也被称为[汉明重量](#)）。

> 输入：00000000000000000000000000001011
> 输出：3
> 解释：输入的二进制串 00000000000000000000000000001011 中，共有三位为 '1'。

> 输入：00000000000000000000000010000000
> 输出：1
> 解释：输入的二进制串 00000000000000000000000010000000 中，共有一位为 '1'。

题解

1、循环，向右移，比较最后一位是否为1

```python
class Solution:
    def hammingWeight(self, n: int) -> int:
        res=0
        for i in range(32):
            res+=n&1
            n>>=1

        return res
```

2、转换成字符，判断1的个数

```python
class Solution:
    def hammingWeight(self, n: int) -> int:
        res=0
        bin_n=bin(n)
        for i in bin_n[2:].zfill(32):
            if i == '1':
                res+=1

        return res
```

3、库函数

```
class Solution(object):
    def hammingWeight(self, n):
        return bin(n).count("1")
```

4、 `n & (n - 1)` ，这个代码可以把 n 的二进制中，最后一个出现的 1 改写成 0。

```
class Solution(object):
    def hammingWeight(self, n):
        res = 0
        while n:
            res += 1
            n &= n - 1
        return res
```

31-40

202、快乐数

编写一个算法来判断一个数 n 是不是快乐数。

「快乐数」定义为：

- 对于一个正整数，每一次将该数替换为它每个位置上的数字的平方和。
- 然后重复这个过程直到这个数变为 1，也可能是 无限循环 但始终变不到 1。
- 如果 可以变为 1，那么这个数就是快乐数。

如果 n 是快乐数就返回 true ； 不是，则返回 false 。

> 输入：19
> 输出：true
> 解释：
> 12 + 92 = 82
> 82 + 22 = 68
> 62 + 82 = 100
> 12 + 02 + 02 = 1

> 输入：n = 2
> 输出：false

题解

1、转成str，平方后再变成int，循环小于10000次

```
class Solution:
    def isHappy(self, n: int) -> bool:
        str_n=str(n)
        num=0
        x=10000
        while x>0:
```

```
            num=0
            for i in str_n:
                num+=int(i)**2
            if num==1:
                return True
            str_n=str(num)
            x-=1
        return False
```

2、快慢指针

重复这个变换一定会出现循环

- 如果是快乐数，那么快指针一定先到达1，原地打圈，然后慢指针也到达1。
- 如果不是快乐数，那么快慢指针相遇的时候一定不是1，
  因为只要过程中变到1就跳不出来了。

```
def isHappy(n: int) -> bool:
    def nxt(n):
        return sum([int(c)**2 for c in str(n)])
    slow, fast = n, nxt(n)
    while slow!=fast:
        slow = nxt(slow)
        fast = nxt(nxt(fast))
        #print(slow, fast)
    return slow==1
```

3、无限循环，判定小于10的情况

```
class Solution:
    def isHappy(self, n: int) -> bool:
        if n == 1:
            return True
        def tonum(num):     #更新n
            sum = 0
            while num:
                sum += (num % 10)**2
                num //= 10
            return sum

        while True:
            n = tonum(n)
            if n < 10:
                if n == 1 or n == 7:#特别注意在
                    return True
                return False
```

4、集和解法

```
class Solution:
    def isHappy(self, n: int) -> bool:
        seen = {1}
        while n not in seen:
            seen.add(n)
            n = sum(int(i) ** 2 for i in str(n))
        return n == 1
```

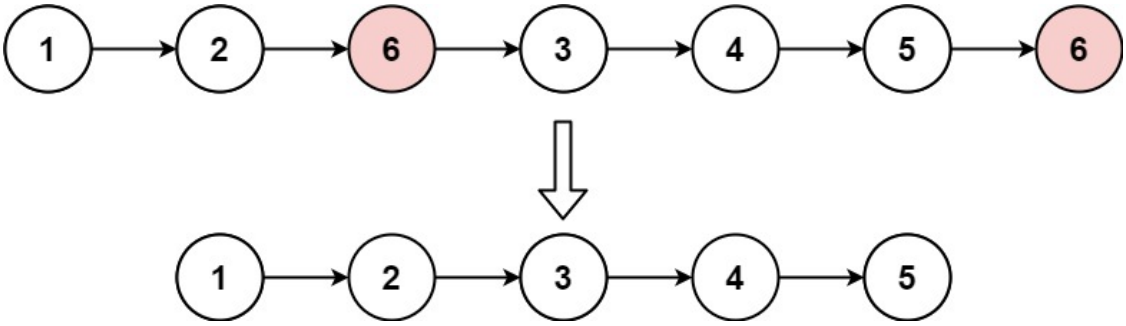
5、递归

- 不是快乐数的数称为不快乐数(unhappy number)，所有不快乐数的数位平方和计算，最后都会进入 4 → 16 → 37 → 58 → 89 → 145 → 42 → 20 → 4 的循环中
- 已知规律： [1 ~ 4] 中只有 1 是快乐数，[5 ~ ∞] 的数字要么回归到 1 要么回归到 4 或 3
- 因此仅需在 n > 4 时调用递归

```
class Solution:
    def isHappy(self, n: int) -> bool:
        return self.isHappy(sum(int(i) ** 2 for i in str(n))) if n > 4 else n ==
1
```

203、移除链表元素

给你一个链表的头节点 `head` 和一个整数 `val` ，请你删除链表中所有满足 `Node.val == val` 的节点，并返回 **新的头节点** 。




```
输入：head = [1,2,6,3,4,5,6], val = 6
输出：[1,2,3,4,5]
```

```
输入：head = [7,7,7,7], val = 7
输出：[]
```

题解

1、递归

可以直接将 removeElements 的结果存放在 head->next 中，然后再判断 head->val 是否等于待删除的 val 即可。

```
class Solution:
    def removeElements(self, head: ListNode, val: int) -> ListNode:
        if not head: return
        head.next = self.removeElements(head.next, val)
        return head.next if head.val == val else head
```

2、迭代

1. 遍历链表找到所有值为 val 的节点；
2. 将值为 val 的节点的上一个节点直接指向该节点的下一个节点（ pre->next = pre->next->next）。

```python
class Solution:
    def removeElements(self, head: ListNode, val: int) -> ListNode:
        while head and head.val == val:
            head = head.next
        if not head: return
        pre = head
        while pre.next:
            if pre.next.val == val:
                pre.next = pre.next.next
            else:
                pre = pre.next
        return head
```

3、双指针

1. 设置两个指针分别指向头节点，pre （记录待删除节点的前一节点）和 cur (记录当前节点)；
2. 遍历整个链表，查找节点值为 val 的节点，找到了就删除该节点，否则继续查找。
    1. 找到，将当前节点的前一节点（pre 节点或者说是之前最近一个值不等于 val 的节点）连接到当前节点（cur 节点）的下一个节点（pre->next = cur->next）。
    2. 没找到，更新最近一个值不等于 val 的节点（pre = cur），继续遍历（cur = cur->next）。

```python
class Solution:
    def removeElements(self, head: ListNode, val: int) -> ListNode:
        while head and head.val == val:
            head = head.next
        pre, cur = head, head
        while cur:
            if cur.val == val:
                pre.next = cur.next
            else:
                pre = cur
            cur = cur.next

        return head
```

204、计数质数

统计所有小于非负整数 `n` 的质数的数量。

> ```
> 输入：n = 10
> 输出：4
> 解释：小于 10 的质数一共有 4 个，它们是 2，3，5，7 。
> ```
>
> ```
> 输入：n = 0
> 输出：0
> ```

题解

1、筛质数（埃氏筛）

我们从2向后遍历，每遇到一个数字，将其倍数所对应的 is_prime 设为False，
因此遇到新的数字num,is_prime[num]=True说明它不是任何2..num-1的数字的倍数，即质数。

```
def countPrimes(n):
    is_prime = [True]*(n+1)
    ans = 0
    for num in range(2,n+1):
        if is_prime[num]:
            ans+=1
            for k in range(1,n//num+1):
                is_prime[num*k]=False
    return ans
```

2、优化

```
class Solution:
    def countPrimes(self, n: int) -> int:
        if n < 2: return 0
        isPrimes = [1] * n
        isPrimes[0] = isPrimes[1] = 0
        for i in range(2, int(n ** 0.5) + 1):
            if isPrimes[i] == 1:
                isPrimes[i * i: n: i] = [0] * len(isPrimes[i * i: n: i])
        return sum(isPrimes)
```

205、同构字符串

给定两个字符串 s 和 t，判断它们是否是同构的。

如果 s 中的字符可以按某种映射关系替换得到 t ，那么这两个字符串是同构的。

每个出现的字符都应当映射到另一个字符，同时不改变字符的顺序。不同字符不能映射到同一个字符上，相同字符只能映射到同一个字符上，字符可以映射到自己本身。

> 输入: s = "egg", t = "add"
> 输出: true

> 输入: s = "foo", t = "bar"
> 输出: false

题解

1、字符串index对比

```
class Solution:
    def isIsomorphic(self, s, t):
        for i in range(len(s)):
            if s.index(s[i]) != t.index(t[i]):
                return False
        return True
```

2、hash表

```
class Solution:
    def isIsomorphic(self, s: str, t: str) -> bool:
        dic1=dict()
        dic2=dict()
        for i in range(len(s)):
            if (s[i] in dic1 and dic1[s[i]]!=t[i]) or (t[i] in dic2 and
dic2[t[i]]!=s[i]):
                return False
            dic1[s[i]]=t[i]
            dic2[t[i]]=s[i]
        return True
```

3、zip

```
class Solution:
    def isIsomorphic1(self, s: str, t: str) -> bool:
        word = zip(*set(zip(s, t)))     #横向对比   s = "egg", t = "add"
                                        #   w对应：('e','g'),('a','d')
        for w in word:
            if len(w) != len(set(w)):
                return False
        return True
```

4、map

```
class Solution:
    def isIsomorphic(self, s: str, t: str) -> bool:
        return [*map(s.index, s)] == [*map(t.index, t)]
```

217、存在重复元素

给定一个整数数组，判断是否存在重复元素。

如果存在一值在数组中出现至少两次，函数返回 `true` 。如果数组中每个元素都不相同，则返回 `false` 。

> v输入: [1,2,3,1]
> 输出: true
>
> 输入: [1,2,3,1]
> 输出: true

题解

1、set()函数

```python
class Solution:
    def containsDuplicate(self, nums: List[int]) -> bool:
        return len(nums)!=len(set(nums))
```

2、哈希字典

```python
class Solution:
    def containsDuplicate(self, nums: List[int]) -> bool:
        dic={}
        for i in nums:
            if i not in dic:
                dic[i]=0
            else:
                return True

        return False
```

3、排序+相邻元素重复相等

```python
class Solution:
    def containsDuplicate(self, nums: List[int]) -> bool:
        nums.sort()
        for i in range(len(nums)-1):
            if nums[i] == nums[i+1]:
                 return True
        return False
```

219、存在重复元素II

给定一个整数数组和一个整数 k，判断数组中是否存在两个不同的索引 i 和 j，使得 nums [i] = nums [j]，并且 i 和 j 的差的 绝对值 至多为 k。

> ```
> 输入: nums = [1,2,3,1], k = 3
> 输出: true
> ```
>
> ```
> 输入: nums = [1,0,1,1], k = 1
> 输出: true
> ```
>
> ```
> 输入: nums = [1,2,3,1,2,3], k = 2
> 输出: false
> ```

题解

1、哈希表保存元素及其索引

```python
class Solution:
    def containsNearbyDuplicate(self, nums: List[int], k: int) -> bool:
        hash={}
        for i in range(len(nums)):
            if(nums[i] not in hash):
                hash[nums[i]]=i
            else:
                if(i-hash[nums[i]]<=k):
                    return True
                else:
                    hash[nums[i]]=i
        return False
```

2、哈希表做滑动数组+遍历

1. 我们构建一个哈希表window（之所以使用哈希表是因为我们需要对其进行很多的【 查询】、【插入】和【删除】操作，哈希表的这三个操作用时都是O(1)）
2. window的额定宽度是K+1，当 I>K时我们就需要将nusm[i-k-1]从哈希表中删除。
3. 查询当前值nums[i]是否存在哈希表中，若存在，则说明在k+1的范围内存在重复元素，return True
4. 将当前值nums[i]插入哈希表
5. 重复234操作直到遍历完整个nums
6. 若遍历完仍未找到，说明不存在这样的 [I,j]对，return False

```python
class Solution:
    def containsNearbyDuplicate(self, nums: List[int], k: int) -> bool:
        window={}
        for i in range(len(nums)):
            if i > k :
                window.pop(nums[i-k-1])
            if window and nums[i] in window:
                return True
            window[nums[i]]=1
        return False
```

228、汇总区间

给定一个无重复元素的有序整数数组 nums 。

返回 恰好覆盖数组中所有数字 的 最小有序 区间范围列表。也就是说，nums 的每个元素都恰好被某个区间范围所覆盖，并且不存在属于某个范围但不属于 nums 的数字 x 。

列表中的每个区间范围 [a,b] 应该按如下格式输出：

- "a->b" ，如果 a != b
- "a" ，如果 a == b

> 输入：nums = [0,1,2,4,5,7]
> 输出：["0->2","4->5","7"]
> 解释：区间范围是：
> [0,2] --> "0->2"
> [4,5] --> "4->5"
> [7,7] --> "7"

```
输入：nums = [0,2,3,4,6,8,9]
输出：["0","2->4","6","8->9"]
解释：区间范围是：
[0,0] --> "0"
[2,4] --> "2->4"
[6,6] --> "6"
[8,9] --> "8->9"
```

题解

1、双指针

1. 左右指针首先都指向nums[0]
2. 循环（1，len(nums)），判断新的nums[i]是否等于right+1，
    - 如果等于，则更新right，
    - 否则，输出区间，并重置left和right

```python
class Solution:
    def summaryRanges(self, nums: List[int]) -> List[str]:
        if not nums:
            return []
        left,right=nums[0],nums[0]
        res=[]

        for i in range(1,len(nums)):
            if nums[i]==(right+1):
                right=nums[i]
                continue
            else:
                if left==right:
                    res.append(str(left))
                else:
                    res.append(str(left)+'->'+str(right))
                left=nums[i]
                right=nums[i]

        if left==right:
            res.append(str(left))
        else:
            res.append(str(left)+'->'+str(right))

        return res
```

优化：末尾哨兵，

```python
class Solution:
    def summaryRanges(self, nums: List[int]) -> List[str]:
        res=[]
        if len(nums)==0: return res
        # 末尾添加一个不连续的数，这样每个原数组元素都能“结算”
        nums.append(nums[0]-1)
        start=end=nums[0]
        for i in range(1, len(nums)):
            if nums[i]-1 == nums[i-1]:
```

```
                end = nums[i]
            else:
                if start==end:
                    res.append(str(start))
                else:
                    res.append(str(start)+"->"+str(end))
                start=end=nums[i]
        return res
```

2、双循环（类似快慢指针），寻找一个区间并保存

```
class Solution:
    def summaryRanges(self, nums: List[int]) -> List[str]:
        n = 0
        res = []
        while n < len(nums):
            if n+1 < len(nums) and nums[n]+1 == nums[n+1]:
                m = n
                while n+1 < len(nums) and nums[n]+1 == nums[n+1]:
                    n += 1
                res.append("{}->{}".format(nums[m],nums[n]))
            else:
                res.append(str(nums[n]))
            n +=1
        return res
```

231、2的幂

给定一个整数，编写一个函数来判断它是否是 2 的幂次方。

> ```
> 输入: 1
> 输出: true
> 解释: 20 = 1
> ```
>
> ```
> 输入: 16
> 输出: true
> 解释: 24 = 16
> ```
>
> ```
> 输入: 218
> 输出: false
> ```

题解

1、比较判断，从0次幂开始，直到等于n—返回True，大于n—返回False

```
class Solution:
    def isPowerOfTwo(self, n: int) -> bool:
        i=0
        while 1:
            if 2**i==n:
                return True
            elif 2**i<n:
                i+=1
            else:
                return False
```

2、除2判断

- 余数不等于0——输出False
- 余数等于0，n=n/2，n=1的话，输出True

```
class Solution:
    def isPowerOfTwo(self, n: int) -> bool:
        if n==1:return True
        if n==0:return False
        while 1:
            if n%2==0:
                n/=2
                if n==1:
                    return True
            else:
                return False
```

优化：

```
class Solution:
    def isPowerOfTwo(self, n: int) -> bool:
        while n > 1:
            n = n / 2       #注意，这里不是//  而是/   返回的是float类型
        return n == 1
```

3、位运算

- 若 n = 2^x，且 x为自然数（即 n为 2的幂），则一定满足以下条件：
    1. 恒有 n & (n - 1) == 0，这是因为：
        - n 二进制最高位为 1，其余所有位为 0；
        - n - 1 二进制最高位为 0，其余所有位为 1；
    2. 一定满足 n > 0。

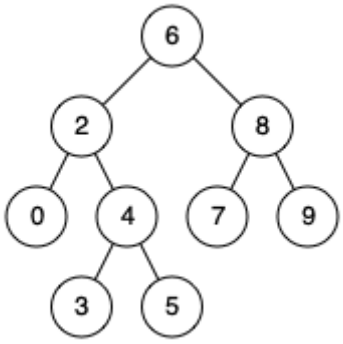
```
class Solution:
    def isPowerOfTwo(self, n: int) -> bool:
        return n > 0 and n & (n - 1) == 0
```

```
class Solution:
    def isPowerOfTwo(self, n: int) -> bool:
        return bin(n).count('1') == 1 if n > 0 else False
```

235、二叉搜索树的最近公共祖先

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）。”




```
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。
```

```
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。
```

题解

1、最近公共祖先的值一定介于p、q值之间(闭区间)

```
class Solution:
    def lowestCommonAncestor(self, root, p, q):
        while (root.val - p.val) * (root.val - q.val) > 0:
            root = (root.left, root.right)[p.val > root.val]
        return root
```

2、递归

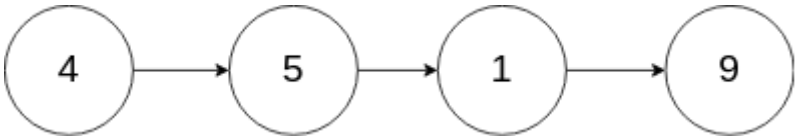
1. 题目要求为最近的公共祖先，首先想到的是普通二叉树的做法，分别记录从root遍历到 p q节点的祖先，对比得到最近公共祖先
2. 但是这道题为BST，分析BST特征，左树比当前节点小，右树比当前节点大。可以想到，我们可以容易的判断p,q在当前节点的哪一侧。
3. 如果再同侧，则再往下层递归，如在两侧，则返回当前节点

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

237、删除链表中的节点

请编写一个函数，使其可以删除某个链表中给定的（非末尾）节点。传入函数的唯一参数为 **要被删除的节点** 。

现有一个链表 -- head = [4,5,1,9]，它可以表示为:

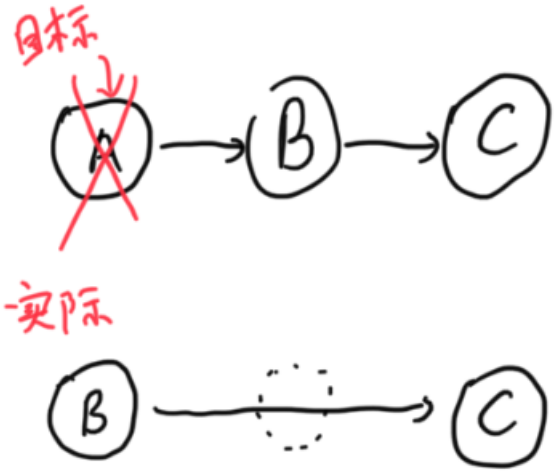


> 输入：head = [4,5,1,9], node = 5
> 输出：[4,1,9]
> 解释：给定你链表中值为 5 的第二个节点，那么在调用了你的函数之后，该链表应变为 4 -> 1 -> 9.

> 输入：head = [4,5,1,9], node = 1
> 输出：[4,5,9]
> 解释：给定你链表中值为 1 的第三个节点，那么在调用了你的函数之后，该链表应变为 4 -> 5 -> 9.

题解

1、node就是要删除的节点，我们任务是要把他删掉




```
class Solution:
    def deleteNode(self, node):
        """
        :type node: ListNode
        :rtype: void Do not return anything, modify node in-place instead.
        """
        node.val = node.next.val
        node.next = node.next.next
```

41-50

242、有效的字母异位词

给定两个字符串 *s* 和 *t* ，编写一个函数来判断 *t* 是否是 *s* 的字母异位词。

> 输入: s = "anagram", t = "nagaram"
> 输出: true

> 输入: s = "rat", t = "car"
> 输出: false

题解

1、sorted()函数

```
class Solution:
    def isAnagram(self, s: str, t: str) -> bool:
        return sorted(s)==sorted(t)
```

2、循环s，删除t中元素，使用try

```
class Solution:
    def isAnagram(self, s: str, t: str) -> bool:
        if len(s)!=len(t):return False
        t=list(t)
        for i in s:
            try:
                t.remove(i)
            except:
                return False

        return True
```

3、字母计数法

```
from collections import Counter
class Solution:
    def isAnagram(self, s: str, t: str) -> bool:
        if len(s) != len(t):
            return False
        s_count = Counter(s)
        t_count = Counter(t)
        for key, value in s_count.items():
            t_value = t_count.get(key, 0)
            if value != t_value:
                return False
        return True
```

优化:

```python
class Solution:
    def isAnagram(self, s: str, t: str) -> bool:
        return collections.Counter(s) == collections.Counter(t)
```

257、二叉树的所有路径

给定一个二叉树，返回所有从根节点到叶子节点的路径。

说明: 叶子节点是指没有子节点的节点。

> 输入:
>
> ```
> 1
> /   \
> 2     3
> \
> 5
> ```
>
> 输出: ["1->2->5", "1->3"]
>
> 解释: 所有根节点到叶子节点的路径为: 1->2->5, 1->3

题解

1、显示回溯

```python
class Solution:
    def binaryTreePaths(self, root: TreeNode) -> List[str]:
        if root is None: return []
        result = []
        subset = [str(root.val)]
        def DFS(root, subset, result):
            if root.left is None and root.right is None:
                result.append(''.join(subset))
                return

            if root.left is not None:
                subset.append('->' + str(root.left.val))
                DFS(root.left, subset, result)
                subset.pop()

            if root.right is not None:
                subset.append('->' + str(root.right.val))
                DFS(root.right, subset, result)
                subset.pop()

        DFS(root, subset, result)

        return result
```

2、隐式回溯

```python
class Solution:
    def binaryTreePaths(self, root: TreeNode) -> List[str]:
        def DFS(root, subset):
            if root:
                subset += str(root.val)
                if not root.left and not root.right:
                    result.append(subset)
                else:
                    subset += '->'
                    DFS(root.left, subset)
                    DFS(root.right, subset)

        result = []
        DFS(root, '')

        return result
```

3、BFS

```python
class Solution:
    def binaryTreePaths(self, root: TreeNode) -> List[str]:
        from collections import deque
        if not root: return []
        res = []
        queue = deque()
        queue.appendleft([root, []])
        while queue:
            node, tmp = queue.pop()
            if not node.left and not node.right:
                res.append("->".join(tmp + [str(node.val)]))
            if node.left:
                queue.appendleft([node.left, tmp + [str(node.val)]])
            if node.right:
                queue.appendleft([node.right, tmp + [str(node.val)]])
        return res
```

258、各位相加

给定一个非负整数 `num`，反复将各个位上的数字相加，直到结果为一位数。

> 输入: 38
> 输出: 2
> 解释: 各位相加的过程为：3 + 8 = 11, 1 + 1 = 2。 由于 2 是一位数，所以返回 2。

题解

1、按照流程，各位相加

```python
class Solution:
    def addDigits(self, num: int) -> int:
        if num<10:return num
        s=str(num)
        while int(s)>9:
            s=str(sum(int(i) for i in s))

        return int(s)
```

2、数学法

任意num为9的倍数时，其位数最终和必为9

1. xyz = 100*x* + *10*y + z = 99*x* + *99*y + (x+y+z)

   x，y，z的最大值可能都是9，所以（x+y+z）之和可能是两位数也可能是一位数。如果是一位数正好就是我们模9的余数，如果是两位数则又变成了

2. mn = 10*m* + *n* = *9*m + (m+n)

   m最大为1 n最大为8 所以（m+n）是一位数，综上xyz按位相加的值就是模9取余的值，

```python
class Solution:
    def addDigits(self, num: int) -> int:
        if num %9 != 0 and num != 0:
            return num%9
        elif num == 0:
            return 0
        else:
            return 9
```

263、丑数

给你一个整数 n ，请你判断 n 是否为 丑数 。如果是，返回 true ；否则，返回 false 。

丑数 就是只包含质因数 2、3 和/或 5 的正整数。

> 输入：n = 6
> 输出：true
> 解释：6 = 2 × 3

> 输入：n = 14
> 输出：false
> 解释：14 不是丑数，因为它包含了另外一个质因数 7 。

> 输入：n = 1
> 输出：true
> 解释：1 通常被视为丑数。

题解

1、if判断是否可以整除2，3，5

```python
class Solution:
    def isUgly(self, n: int) -> bool:
        if n==0:return False
        while n!=1:
            if n%2==0:
                n//=2
            elif n%3==0:
                n//=3
            elif n%5==0:
                n//=5
            else:
                return False

        return True
```

另一种方法

```python
class Solution(object):
    def isUgly(self, n):
        arr = [2,3,5]
        if n ==0:return False
        for i in arr:
            while n%i ==0:
                n/=i
        return n==1
```

268、丢失的数字

给定一个包含 `[0, n]` 中 `n` 个数的数组 `nums` ，找出 `[0, n]` 这个范围内没有出现在数组中的那个数。

> 输入：nums = [3,0,1]
> 输出：2
> 解释：n = 3，因为有 3 个数字，所以所有的数字都在范围 [0,3] 内。2 是丢失的数字，因为它没有出现在 nums 中。
>
> 输入：nums = [0,1]
> 输出：2
> 解释：n = 2，因为有 2 个数字，所以所有的数字都在范围 [0,2] 内。2 是丢失的数字，因为它没有出现在 nums 中。
>
> 输入：nums = [9,6,4,2,3,5,7,0,1]
> 输出：8
> 解释：n = 9，因为有 9 个数字，所以所有的数字都在范围 [0,9] 内。8 是丢失的数字，因为它没有出现在 nums 中。

题解

1、循环len(nums)，查找没有出现再nums中的数——少内存，多时间

```python
class Solution:
    def missingNumber(self, nums: List[int]) -> int:
        for i in range(len(nums)+1):
            if i not in nums:
                return i
```

2、排序，查找索引与对应数字是否相等——少时间，

```python
class Solution:
    def missingNumber(self, nums: List[int]) -> int:
        nums=sorted(nums)
        for i in range(len(nums)):
            if i != nums[i]:
                return i

        return len(nums)
```

3、创建一个不缺数字的数组，求和之后减去原数组求和，

```python
class Solution:
    def missingNumber(self, nums: List[int]) -> int:
        x=sum(i for i in range(len(nums)+1))

        return x-sum(nums)
```

4、位运算——使用异或符号查找缺失数字

由于异或运算（XOR）满足结合律，并且对一个数进行两次完全相同的异或运算会得到原来的数，因此我们可以通过异或运算找到缺失的数字。

```python
class Solution:
    def missingNumber(self, nums):
        missing = len(nums)
        for i, num in enumerate(nums):
            missing ^= i ^ num
        return missing
```

290、单词规律

给定一种规律 pattern 和一个字符串 str ，判断 str 是否遵循相同的规律。

这里的 遵循 指完全匹配，例如， pattern 里的每个字母和字符串 str 中的每个非空单词之间存在着双向连接的对应规律。

> ```
> 输入: pattern = "abba", str = "dog cat cat dog"
> 输出: true
> ```
>
> ```
> 输入:pattern = "abba", str = "dog cat cat fish"
> 输出: false
> ```

```
输入: pattern = "aaaa", str = "dog cat cat dog"
输出: false
```

题解

1、创建字典

```
class Solution:
    def wordPattern(self, pattern: str, s: str) -> bool:
        dic={}
        s=s.split()
        if len(pattern)!=len(s):return False
        if len(set(pattern))!=len(set(s)):return False
        for i in range(len(pattern)):
            if pattern[i] not in dic:
                dic[pattern[i]]=s[i]
            else:
                if dic[pattern[i]]!=s[i]:
                    return False

        return True
```

2、类似205、同构字符串：题解4，使用索引判断

- **map()** 会根据提供的函数对指定序列做映射。

  第一个参数 function 以参数序列中的每一个元素调用 function 函数，返回包含每次 function 函数返回值的新列表。

```
def wordPattern(self, pattern: str, str: str) -> bool:
    res=str.split()
    return list(map(pattern.index, pattern))==list(map(res.index,res))
            #pattern.index  作为函数，返回pattern中字母首次出现的位置
```

3、使用zip函数，类似205、同构字符串：题解3

```
class Solution:
    def wordPattern(self, pattern: str, str: str) -> bool:
        if len(str.split(" ")) != len(list(pattern)):
            return False
        for l in zip(*set(zip(list(pattern), str.split(" ")))):
            if len(l) != len(set(l)):
                return False
        return True
```

292、Nim游戏

你和你的朋友，两个人一起玩 Nim 游戏：

- 桌子上有一堆石头。
- 你们轮流进行自己的回合，你作为先手。
- 每一回合，轮到的人拿掉 1 - 3 块石头。

- 拿掉最后一块石头的人就是获胜者。

假设你们每一步都是最优解。请编写一个函数，来判断你是否可以在给定石头数量为 n 的情况下赢得游戏。如果可以赢，返回 true；否则，返回 false 。

> 输入：n = 4
> 输出：false
> 解释：如果堆中有 4 块石头，那么你永远不会赢得比赛；
> 因为无论你拿走 1 块、2 块 还是 3 块石头，最后一块石头总是会被你的朋友拿走。

> 输入：n = 1
> 输出：true

> 输入：n = 2
> 输出：true

题解

1、找4，只要是4的倍数，就一定赢不了

```
class Solution:
    def canWinNim(self, n: int) -> bool:
        if n%4==0:
            return False
        else:
            return True
```

326、3的幂

给定一个整数，写一个函数来判断它是否是 3 的幂次方。如果是，返回 true ；否则，返回 false 。

整数 n 是 3 的幂次方需满足：存在整数 x 使得 n == 3x

> 输入：n = 27
> 输出：true

> 输入：n = 0
> 输出：false

题解

1、同2的幂，题解1

```
class Solution:
    def isPowerOfThree(self, n: int) -> bool:
        i=0
        while 1:
            if 3**i==n:
                return True
            elif 3**i<n:
                i+=1
            else:
                return False
```

2、同2的幂，题解2

```
class Solution:
    def isPowerOfThree(self, n: int) -> bool:
        while n>1:
            n/=3

        return n==1
```

3、正则化

```
class Solution:
    def isPowerOfThree1(self, n: int) -> bool:
        return n > 0 and 1162261467 % n == 0;
```

4、递归

```
    def isPowerOfThree2(self, n: int) -> bool:
        if n==1:return True
        if (n == 0 or n % 3 != 0): return False
        return self.isPowerOfThree2(n/3)
```

5、对数

```
    def isPowerOfThree4(self, n: int) -> bool:
        if n <= 0: return False
        e=log10(n) / log10(3)
        return  e== floor(e)
```

242、4的幂

给定一个整数，写一个函数来判断它是否是 4 的幂次方。如果是，返回 true ；否则，返回 false 。

整数 n 是 4 的幂次方需满足：存在整数 x 使得 n == 4^x

> 输入：n = 16
> 输出：true

```
输入：n = 5
输出：false
```

题解

1、同上题解1

```python
class Solution:
    def isPowerOfFour(self, n: int) -> bool:
        i=0
        while 1:
            if 4**i==n:
                return True
            elif 4**i<n:
                i+=1
            else:
                return False
```

2、同上题解2

```python
class Solution:
    def isPowerOfFour(self, n: int) -> bool:
        while n>1:
            n/=4

        return n==1
```

344、反转字符串

编写一个函数，其作用是将输入的字符串反转过来。输入字符串以字符数组 char[] 的形式给出。

不要给另外的数组分配额外的空间，你必须原地修改输入数组、使用 O(1) 的额外空间解决这一问题。

你可以假设数组中的所有字符都是 ASCII 码表中的可打印字符。

```
输入：["h","e","l","l","o"]
输出：["o","l","l","e","h"]
```

```
输入：["H","a","n","n","a","h"]
输出：["h","a","n","n","a","H"]
```

题解

1、对半反转，遍历一半长度

```
class Solution:
    def reverseString(self, s: List[str]) -> None:
        n=len(s)//2
        for i in range(n):
            x=s[i]
            s[i]=s[-i-1]
            s[-i-1]=x
```

2、双指针

```
class Solution:
    def reverseString(self, s: List[str]) -> None:
        """
        Do not return anything, modify s in-place instead.
        """
        i = 0
        j = len(s)-1
        while i <j :
            s[i],s[j] =s[j],s[i]
            i+=1
            j-=1
        return s
```

3、reverse()函数

```
class Solution:
    def reverseString(self, s: List[str]) -> None:
        s.reverse()
        #或者
        s[:] = s[::-1]
```