LeetCode 题解 (Python 版本)

胡欣毅 (icedomain_hu@qq.com)

https://github.com/Icedomain/LeetCode

最后更新 March 12, 2020

本文档一共统计了 258 道题

```python
1  #
2  # @lc app=leetcode.cn id=1 lang=python3
3  #
4  # [1] 两数之和
5  #
6  class Solution:
7      def twoSum(self, nums: List[int], target: int) -> List[int]:
8          dic = {}
9          for i in range(len(nums)):
10             if target - nums[i] in dic:
11                 return [dic[target-nums[i]], i]
12             dic[nums[i]] = i
```

```python
1  #
2  # @lc app=leetcode.cn id=2 lang=python3
3  #
4  # [2] 两数相加
5  #
6  # Definition for singly-linked list.
7  # class ListNode:
8  #     def __init__(self, x):
9  #         self.val = x
10 #         self.next = None
11
12 class Solution:
13     def addTwoNumbers(self, l1: ListNode, l2: ListNode) -> ListNode:
14         jingwei = 0
15         # 两个空指针 n后面要被覆盖的
16         head = n = ListNode(0)
17         while l1 or l2 or jingwei:
18             v1 = v2 = 0
19             if l1:
20                 v1 = l1.val
21                 l1 = l1.next
22             if l2:
```

```python
                v2 = l2.val
                l2 = l2.next
            # 除数、余数
            val = (v1+v2+jingwei) % 10
            jingwei = (v1+v2+jingwei) // 10
            n.next = ListNode(val)
            # 指向下一个
            n = n.next
        return head.next # 记得把第一个0去掉
```

```python
#
# @lc app=leetcode.cn id=3 lang=python3
#
# [3] 无重复字符的最长子串
#
class Solution:
    def lengthOfLongestSubstring(self, s: str) -> int:
        # 记录表 256个字符
        charmap = [-1 for _ in range(256)]

        start = maxlen = 0
        # 遍历 滑动窗 [start,j] j往右边移动 若遇到重复的 start又移一位
        for j in range(len(s)):
            # 如果这个字符出现过了，又移动 最左边那个踢出滑动窗
            if charmap[ord(s[j])] >= start:
                start = charmap[ord(s[j])] + 1
            # 如果这个字符在滑动窗中没出现过，位置给它(出现过也要给它)
            charmap[ord(s[j])] = j
            maxlen = max(maxlen , j-start +1)
        return maxlen
```

```python
#
# @lc app=leetcode.cn id=4 lang=python3
#
# [4] 寻找两个有序数组的中位数
#
class Solution:
    def findMedianSortedArrays(self, nums1: List[int], nums2: List[int]) -> float:
        leng = len(nums1)+len(nums2)
        if leng%2 == 1:#奇数
            return self.findk(nums1,nums2,leng//2)
        else:
            return ( self.findk(nums1,nums2,leng//2-1)+self.findk(nums1,nums2,leng//2))/2.0
    # 找k大的数
    def findk( self ,nums1,nums2,k):
        if not nums1:
```

```
16            return nums2[k]
17        if not nums2:
18            return nums1[k]
19        l1 , l2 = len(nums1)//2,len(nums2)//2
20        val1 ,val2 = nums1[l1],nums2[l2]
21
22        if l1+l2<k:# 往右找
23            if val1 > val2:
24                return self.findk(nums1, nums2[l2 + 1:], k − l2 − 1)
25            else :
26                return self.findk(nums1[l1 + 1:],nums2, k − l1 − 1)
27        else : # 往左找
28            if val1 > val2:
29                return self.findk(nums1[:l1],nums2, k)
30
31            else :
32                return self.findk(nums1, nums2[:l2], k)
```

```
1  #
2  # @lc app=leetcode.cn id=5 lang=python3
3  #
4  # [5] 最长回文子串
5  #
6  class Solution:
7      def longestPalindrome(self, s: str) −> str:
8          if s is None:
9              return None
10
11         # 动态规划
12         dp = [[0 for _ in range(len(s))] for _ in range(len(s))]
13         left , right , max_len = 0, 0, 0
14
15         for j in range(len(s)):
16             # 对角线置1
17             dp[j][j] = 1
18             for i in range(j):
19                 if s[i] == s[j] and (j−i < 2 or dp[i+1][j−1]):
20                     dp[i][j] = 1
21                 if dp[i][j] and max_len < j−i+1:
22                     max_len = j − i + 1
23                     left , right = i, j
24         return s[ left :right+1]
```

```
1  #
2  # @lc app=leetcode.cn id=6 lang=python3
3  #
```

```python
# [6] Z 字形变换
#
class Solution:
    def convert(self, s: str, numRows: int) -> str:
        if numRows == 1 or numRows >= len(s):
            return s
        # z前半个(|/)个数两行减2
        p = 2 * (numRows - 1)

        result = [""] * numRows
        for i in range(len(s)):
            floor = i % p # 一个形状轮回的位置
            if floor >= p//2: # 在/上
                floor = p - floor
            result[floor] += s[i]
        return "".join(result)
```

```python
#
# @lc app=leetcode.cn id=7 lang=python3
#
# [7] 整数反转
#
class Solution:
    def reverse(self, x: int) -> int:
        sign = 1 if x > 0 else -1
        res = 0
        x = abs(x)
        while x:
            res = res*10 + x%10
            if res > 2**31 - 1:
                return 0
            x = x//10

        return sign * res
```

```python
#
# @lc app=leetcode.cn id=8 lang=python3
#
# [8] 字符串转换整数 (atoi)
#
class Solution:
    def myAtoi(self, str: str) -> int:
        # 去空格
        str = str.strip()
        if len(str) == 0:
            return 0
```

```python
12          sign = 1
13          if str[0] == '+' or str[0] == '-':
14              if str[0] == '-':
15                  sign = -1
16              str = str[1:]
17          res = 0
18          for char in str:
19              if char >= '0' and char <= '9':
20                  res = res * 10 + ord(char) - ord('0')
21              if char < '0' or char > '9':
22                  break
23          return max(-2**31, min(sign * res,2**31-1))
```

```python
1  #
2  # @lc app=leetcode.cn id=9 lang=python3
3  #
4  # [9] 回文数
5  #
6  class Solution:
7      def isPalindrome(self, x: int) -> bool:
8          if x < 0:
9              return False
10         # 最高位的位数
11         d = 1
12         while x // d >= 10:
13             d *= 10
14         while x > 0:
15             # p q 对应最高位和最低位
16             p = x //d
17             q = x % 10
18             if p != q:
19                 return False
20             # x 去掉最高位,去掉最低位
21             x = x % d // 10
22             # x 去掉了两位,d也减两位
23             d //= 100
24         return True
```

```python
1  #
2  # @lc app=leetcode.cn id=10 lang=python3
3  #
4  # [10] 正则表达式匹配
5  #
6  class Solution:
7      def isMatch(self, s: str, p: str) -> bool:
8          '''
```

```python
         # 递归写法
         # s已被匹配且p已耗完
         if not s and not p:
             return True
         # p已耗完但s未被完全匹配
         if len(s) > 0 and len(p) == 0:
             return False

         # 如果模式第二个字符是*
         if len(p) > 1 and p[1] == '*':
             if len(s) > 0 and (s[0] == p[0] or p[0] == '.'):  # ax a* or ax .*
                 # 如果第一个字符匹配，三种可能1、p后移两位；2、字符串移1位
                 return self.isMatch(s, p[2:]) or self.isMatch(s[1:], p)
             else:
                 # 如果第一个字符不匹配，p往后移2位，相当于忽略x*
                 return self.isMatch(s, p[2:])
         # 如果模式第二个字符不是*
         if len(s) > 0 and (s[0] == p[0] or p[0] == '.'):
             return self.isMatch(s[1:], p[1:])
         else:
             return False
         '''
         # 动态规划
         # 初始化dp表，初始化表的第一列和第一行
         # p对应列 s对应行
         dp = [[False for j in range(len(p) + 1)] for i in range(len(s) + 1)]
         dp[0][0] = True  # s 和 p 都为空时
         # 若 s 为空时
         # 处理第一行
         # p 与 dp 有一位的错位(多了一个空导致的)
         for j in range(1, len(p) + 1):
             # dp[0][j]= (p[j-1]=="*")and(j>=2)and(dp[0][j-2])
             # 等同于下列语句
             if p[j - 1] == '*':
                 if j >= 2:
                     dp[0][j] = dp[0][j - 2]
         # 第一列就第一个是 True,下面都是 False
         # 不用处理 pass

         for i in range(1, len(s) + 1):
             for j in range(1, len(p) + 1):
                 # j-1才为正常字符串中的索引
                 # p当前位置为"*"时
                 # 代表空串--dp[i][j-2]
                 # 一个或者多个前一个字符--( dp[i-1][j] and (p[j-2]==s[i-1] or p[j-2]=='.')
                 if p[j - 1] == '*':
```

6

```python
                        dp[i][j] = dp[i][j - 2] or (
                                dp[i - 1][j] and (p[j - 2] == s[i - 1] or p[j - 2] == '.')
                                )
                    # p当前位置为"."时或者与s相同时，传递dp[i-1][j-1]的真值
                    else:
                        dp[i][j] = (p[j - 1] == '.' or p[j - 1] == s[i - 1]) and dp[i - 1][j - 1]
        return dp[-1][-1]
```

```python
#
# @lc app=leetcode.cn id=11 lang=python3
#
# [11] 盛最多水的容器
#
class Solution:
    def maxArea(self, height: List[int]) -> int:
        max_area = 0
        left, right = 0, len(height) - 1
        while left < right:
            # 高取左边和右边的高当中的最小值，下标right-left为宽，两者相乘为面积
            temp = min(height[left], height[right]) * (right - left)
            max_area = max(max_area, temp)
            # 判断哪条高小，小的那边下标进行操作
            if height[right] > height[left]:
                left += 1
            else:
                right -= 1
        return max_area
```

```python
#
# @lc app=leetcode.cn id=12 lang=python3
#
# [12] 整数转罗马数字
#
class Solution:
    def intToRoman(self, num: int) -> str:
        # 贪心算法
        dic = {
            'M': 1000,
            'CM': 900, 'D': 500, 'CD': 400, 'C': 100,
            'XC': 90, 'L': 50, 'XL': 40, 'X': 10,
            'IX': 9, 'V': 5, 'IV': 4, 'I': 1,
        }
        result = ""
        for letter, number in dic.items():
            if num >= number:
                result += letter*(num//number)
```

```
19                num %= number
20            return result
```

```
1   #
2   # @lc app=leetcode.cn id=13 lang=python3
3   #
4   # [13] 罗马数字转整数
5   #
6   class Solution:
7       def romanToInt(self, s: str) -> int:
8           dicts = {
9               "I": 1,
10              "V": 5,
11              "X": 10,
12              "L": 50,
13              "C": 100,
14              "D": 500,
15              "M": 1000
16          }
17          s = s.replace("IV", "IIII").replace("IX", "VIIII")
18          s = s.replace("XL", "XXXX").replace("XC", "LXXXX")
19          s = s.replace("CD", "CCCC").replace("CM", "DCCCC")
20          data = 0
21          for item in s:
22              data += dicts[item]
23          return data
```

```
1   #
2   # @lc app=leetcode.cn id=14 lang=python3
3   #
4   # [14] 最长公共前缀
5   #
6   class Solution:
7       def longestCommonPrefix(self, strs: List[str]) -> str:
8           '''
9           sz = zip(*strs)
10          ret = ""
11          for char in sz:
12              if len(set(char)) > 1:
13                  break
14              ret +=char[0]
15          return ret
16          '''
17          if len(strs) == 0:
18              return ''
19          strs.sort(key = lambda x : len(x))
```

```
20          for idx in range(len(strs[0])):
21              # 最大的可能长度就是第一个的长度
22              for i in range(1,len(strs)):
23                  # 对每个元素都要遍历
24                  if strs[i][idx] != strs[0][idx]:
25                      return strs[0][:idx]
26          return strs[0]
```

```
1  #
2  # @lc app=leetcode.cn id=15 lang=python3
3  #
4  # [15] 三数之和
5  #
6  class Solution:
7      def threeSum(self, nums: List[int]) -> List[List[int]]:
8          nums.sort()
9          res = []
10         for i in range(len(nums)-2):
11             if i >0 and nums[i] == nums[i-1]:
12                 continue
13             l,r = i+1,len(nums) - 1
14             while l < r:
15                 s = nums[i]+nums[l]+nums[r]
16                 if s < 0:
17                     l+=1
18                 elif s > 0:
19                     r -= 1
20                 else:
21                     res.append((nums[i], nums[l], nums[r]))
22                     # 避免一样的加进去
23                     while l < r and nums[l] == nums[l+1]:
24                         l += 1
25                     while l < r and nums[r] == nums[r-1]:
26                         r -= 1
27                     l += 1
28                     r -= 1
29         return res
```

```
1  #
2  # @lc app=leetcode.cn id=16 lang=python3
3  #
4  # [16] 最接近的三数之和
5  #
6  class Solution:
7      def threeSumClosest(self, nums: List[int], target: int) -> int:
8          nums.sort()
```

```
9         res = sum(nums[0:3])
10
11         for i in range(len(nums)−2):
12             l,r = i+1,len(nums)−1
13             while l < r:
14                 sum_val = nums[i]+nums[l]+nums[r]
15                 if sum_val == target:
16                     return sum_val
17                 if abs(res−target)>abs(sum_val−target):
18                     res = sum_val
19                 if sum_val < target:
20                     l+=1
21                 else:
22                     r −= 1
23         return res
```

```
1  #
2  # @lc app=leetcode.cn id=17 lang=python3
3  #
4  # [17] 电话号码的字母组合
5  #
6  class Solution:
7      def letterCombinations(self, digits: str) −> List[str]:
8          dmap = {
9          '2': 'abc',
10         '3': 'def',
11         '4': 'ghi',
12         '5': 'jkl',
13         '6': 'mno',
14         '7': 'pqrs',
15         '8': 'tuv',
16         '9': 'wxyz'
17         }
18         if len(digits) == 0:
19             return []
20         if len(digits) == 1:
21             return list(dmap[digits])
22         prev = self.letterCombinations(digits[:−1])
23         additional = dmap[digits[−1]]
24         return [s + c for s in prev for c in additional]
```

```
1  #
2  # @lc app=leetcode.cn id=18 lang=python3
3  #
4  # [18] 四数之和
5  #
```

10

```python
class Solution:
    def fourSum(self, nums: List[int], target: int) -> List[List[int]]:
        res = []
        # 去除异常
        if not nums or len(nums) < 4:
            return res
        nums.sort()
        # 第一个数遍历
        for i in range(len(nums) - 3):
            if i > 0 and nums[i] == nums[i - 1]:
                continue
            # 第二个数遍历
            for j in range(i + 1, len(nums) - 2):
                if j > i + 1 and nums[j] == nums[j - 1]:
                    continue
                # 双指针
                L, R = j + 1, len(nums) - 1
                while L < R:
                    if nums[i] + nums[j] + nums[L] + nums[R] == target:
                        res.append([nums[i], nums[j], nums[L], nums[R]])
                        while L < R and nums[L] == nums[L + 1]:
                            L += 1
                        while L < R and nums[R] == nums[R - 1]:
                            R -= 1
                        L += 1
                        R -= 1
                    elif nums[i] + nums[j] + nums[L] + nums[R] < target:
                        L += 1
                    else:
                        R -= 1
        return res
```

```python
#
# @lc app=leetcode.cn id=19 lang=python3
#
# [19] 删除链表的倒数第N个节点
#
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution:
    def removeNthFromEnd(self, head: ListNode, n: int) -> ListNode:
        if head is None:
```

```python
15            return None
16        dummy = ListNode(-1)
17        dummy.next = head
18        slow = fast = dummy
19        # 先走n步
20        for i in range(n):
21            fast = fast.next
22
23        # slow 少走n步
24        while fast.next :
25            fast = fast.next
26            slow = slow.next
27        # 删除
28        slow.next = slow.next.next
29        return dummy.next
```

```python
1  #
2  # @lc app=leetcode.cn id=20 lang=python3
3  #
4  # [20] 有效的括号
5  #
6  class Solution:
7      def isValid(self, s: str) -> bool:
8          # 判断是否是奇数或空字符
9          if s=='':
10             return True
11         if len(s) %2 != 0:
12             return False
13         count = 0
14         leng = len(s)
15         # 将其中的(){}[] 都换掉，然后判断是否有剩余
16         while(count < leng/2 ):
17             s = s.replace("{}",""). replace("[]",""). replace("()","")
18             count+=1
19
20         if len(s)>0:
21             return False
22         else :
23             return True
```

```python
1  #
2  # @lc app=leetcode.cn id=21 lang=python3
3  #
4  # [21] 合并两个有序链表
5  #
6  # Definition for singly-linked list .
```

```python
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution:
    def mergeTwoLists(self, l1: ListNode, l2: ListNode) -> ListNode:
        dummy = now = ListNode(-1)
        while l1 and l2:
            if l1.val <= l2.val:
                now.next = l1
                l1 = l1.next
            else:
                now.next = l2
                l2 = l2.next
            now = now.next
        now.next = l1 or l2
        return dummy.next
```

```python
#
# @lc app=leetcode.cn id=22 lang=python3
#
# [22] 括号生成
#
class Solution:
    def generateParenthesis(self, n: int) -> List[str]:
        res = []
        if n > 0:
            self.dfs(n, '', res, 0, 0)
        return res

    def dfs(self, n, path, res, left, right):
        # 终止条件
        if len(path) == 2 * n:
            res.append(path)
            return
        # 左括号(够了没
        if left < n:
            self.dfs(n, path+'(', res, left+1, right)
        # 右括号补成和左括号一样多
        if left > right:
            self.dfs(n, path+')', res, left, right+1)
```

```python
#
# @lc app=leetcode.cn id=23 lang=python3
#
```

```python
# [23] 合并K个排序链表
#
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution:
    def mergeKLists(self, lists: List[ListNode]) -> ListNode:
        if not lists:
            return None
        return self.mergeK(lists, 0, len(lists) - 1)

    def mergeK(self, lists, low, high):
        if low == high:
            return lists[low]
        elif low + 1 == high:
            return self.mergeTwolists(lists[low], lists[high])
        mid = (low + high) // 2
        return self.mergeTwolists(self.mergeK(lists, low, mid), self.mergeK(lists, mid + 1, high))

    def mergeTwolists(self, l1, l2):
        if l1 is None:
            return l2
        if l2 is None:
            return l1
        head = curr = ListNode(-1)
        while l1 and l2:
            if l1.val <= l2.val:
                curr.next = l1
                l1 = l1.next
            else:
                curr.next = l2
                l2 = l2.next
            curr = curr.next
        curr.next = l1 or l2
        return head.next
```

```python
#
# @lc app=leetcode.cn id=24 lang=python3
#
# [24] 两两交换链表中的节点
#
# Definition for singly-linked list.
# class ListNode:
```

```python
 8  #     def __init__(self, x):
 9  #         self.val = x
10  #         self.next = None
11
12  class Solution:
13      def swapPairs(self, head: ListNode) -> ListNode:
14          prev = dummy = ListNode(-1)
15          dummy.next = head
16          while prev.next and prev.next.next:
17              # prev a b -> prev b a (交换a,b)
18              a = prev.next
19              b = prev.next.next
20              prev.next, b.next, a.next = b, a, b.next
21              prev = a
22          return dummy.next
```

```python
 1  #
 2  # @lc app=leetcode.cn id=25 lang=python3
 3  #
 4  # [25] K 个一组翻转链表
 5  #
 6  # Definition for singly-linked list.
 7  # class ListNode:
 8  #     def __init__(self, x):
 9  #         self.val = x
10  #         self.next = None
11
12  class Solution:
13      def reverseKGroup(self, head: ListNode, k: int) -> ListNode:
14          if head is None or k < 2:
15              return head
16          dummy = ListNode(0)
17          dummy.next = head
18          start = dummy
19          end = start.next
20
21          count = 0
22          while end:
23              count += 1
24              if count % k == 0:
25                  # 返回为新一轮的头
26                  start = self.reverse(start, end.next)
27                  end = start.next
28              else:
29                  end = end.next
30          return dummy.next
```

```python
31
32    def reverse(self, start, end):
33        prev, curr = start, start.next
34        first = curr
35        while curr != end:
36            temp = curr.next
37            curr.next = prev
38            prev = curr
39            curr = temp
40        start.next = prev
41        first.next = end
42        return first
```

```python
1  #
2  # @lc app=leetcode.cn id=26 lang=python3
3  #
4  # [26] 删除排序数组中的重复项
5  #
6  class Solution:
7      def removeDuplicates(self, nums: List[int]) -> int:
8          idx = 0
9          while idx < len(nums) -1 :
10             if nums[idx] == nums[idx+1]:
11                 nums.pop(idx)
12                 idx -= 1
13             idx += 1
14         return len(nums)
```

```python
1  #
2  # @lc app=leetcode.cn id=27 lang=python3
3  #
4  # [27] 移除元素
5  #
6  class Solution:
7      def removeElement(self, nums: List[int], val: int) -> int:
8          left = 0
9          right = len(nums) - 1
10         while left <= right :
11             if nums[left] == val:
12                 nums[left], nums[right] = nums[right], nums[left]
13                 right -= 1
14             else:
15                 left += 1
16         return left
```

```python
1  #
2  # @lc app=leetcode.cn id=28 lang=python3
```

```python
#
# [28] 实现 strStr()
#
class Solution:
    def strStr(self, haystack: str, needle: str) -> int:
        if not needle or haystack == needle:
            return 0
        elif len(haystack)<= len(needle):
            return -1

        leng = len(needle)
        for i in range(len(haystack)-leng +1):
            if needle == haystack[i:i+leng]:
                return i
        return -1
```

```python
#
# @lc app=leetcode.cn id=29 lang=python3
#
# [29] 两数相除
#
class Solution:
    def divide(self, dividend: int, divisor: int) -> int:
        if (dividend < 0 and divisor < 0) or (dividend > 0 and divisor > 0):
            positive = 1
        else:
            positive = -1

        dividend, divisor = abs(dividend), abs(divisor)
        res = 0
        while dividend >= divisor:
            temp, i = divisor, 1
            while dividend >= temp:
                dividend -= temp
                res += i
                # 除数乘以2 商一下子也多2
                i <<= 1
                temp <<= 1

        # 防止溢出
        return min(max(positive * res,-2**31) , 2**31-1)
```

```python
#
# @lc app=leetcode.cn id=31 lang=python3
#
# [31] 下一个排列
```

```python
#
class Solution:
    def nextPermutation(self, nums: List[int]) -> None:
        # i为数组倒数第二个值，j为倒数第一个值
        i = len(nums) - 2
        j = len(nums) - 1
        # 从右到左找到第一次断崖
        # 第一次非逆序的地方
        while i >= 0 and nums[i] >= nums[i+1]:
            i -= 1

        # 从右到左找到比崖底水平面高的第一个元素
        if i >= 0:
            while j >= 0 and nums[i] >= nums[j]:
                j -= 1
            nums[i], nums[j] = nums[j], nums[i]

        self.reverse(nums, i+1)

    # 用于原地反转nums中从start之后的所有元素
    def reverse(self, nums, start):
        i, j = start, len(nums) - 1
        while i < j:
            nums[i], nums[j] = nums[j], nums[i]
            i += 1
            j -= 1
        return
```

```python
#
# @lc app=leetcode.cn id=32 lang=python3
#
# [32] 最长有效括号
#
class Solution:
    def longestValidParentheses(self, s: str) -> int:
        '''
        # 栈法
        res = []
        stack = []
        for i in range(len(s)):
            if(stack and s[i]==")"):
                res.append(stack.pop())
                res.append(i)
            if(s[i]=="("):
                stack.append(i)
```

```python
19          res.sort()
20          max_len = 0
21          i=0
22          while i < len(res)-1:
23              tmp = i
24              # 最长连续值
25              while( i < len(res)-1 and res[i+1]-res[i] == 1):
26                  i += 1
27              max_len = max(max_len,i-tmp+1)
28              i += 1
29          return max_len
30          '''

31
32          # 动态规划
33          if not s:
34              return 0
35          dp = [0] * len(s)
36          for i in range(1,len(s)):
37              if s[i]==")":
38                  # ()对
39                  if s[i-1]=="(":
40                      dp[i] = dp[i-2] + 2
41                  # 连着两个))
42                  if s[i-1]==")" and i-1-dp[i-1]>=0 and s[i-1-dp[i-1]]=="(":
43                      dp[i] = dp[i-dp[i-1]-2] + dp[i-1] + 2
44          return max(dp)
```

```python
1  #
2  # @lc app=leetcode.cn id=33 lang=python3
3  #
4  # [33] 搜索旋转排序数组
5  #
6  class Solution:
7      def search( self , nums: List[int], target: int) -> int:
8          if not nums:
9              return -1
10         l,r = 0 , len(nums) -1
11
12         while l <= r:
13             mid = (l+r)//2
14             if nums[mid] == target:
15                 return mid
16             # mid在前半段 或者l mid r 都在右边
17             if nums[l] <= nums[mid]:
18                 if nums[l] <= target <= nums[mid]:
19                     r = mid -1
```

```python
            else:
                l = mid + 1
            # l 在左半段 、 mid 在后半段
            else:
                if nums[mid] <= target <= nums[r]:
                    l = mid + 1
                else:
                    r = mid - 1
        return -1
```

```python
#
# @lc app=leetcode.cn id=34 lang=python3
#
# [34] 在排序数组中查找元素的第一个和最后一个位置
#
class Solution:
    def searchRange(self, nums: List[int], target: int) -> List[int]:
        if len(nums) == 0:
            return [-1, -1]
        min = 0
        max = len(nums) - 1
        while min <= max:
            pos = (min + max) // 2
            if nums[pos] > target:
                max = pos - 1
            elif nums[pos] < target:
                min = pos + 1
            else:
                # when nums[pos] == target
                # find the min and max
                for i in range(pos, max + 1):
                    if nums[i] == target:
                        max = i
                for i in range(pos, min - 1, -1):
                    if nums[i] == target:
                        min = i
                return [min, max]
        return [-1, -1]
```

```python
#
# @lc app=leetcode.cn id=35 lang=python3
#
# [35] 搜索插入位置
#
class Solution:
    def searchInsert(self, nums: List[int], target: int) -> int:
```

```
 8            left  = 0
 9            right  = len(nums) − 1
10            while  left  <= right:
11                mid = (left + right)//2
12                if nums[mid] == target:
13                    return mid
14                elif  target  < nums[mid]:
15                    right  = mid − 1
16                else :
17                    left  = mid + 1
18           return  left
```

```python
 1  #
 2  # @lc app=leetcode.cn id=36 lang=python3
 3  #
 4  # [36] 有效的数独
 5  #
 6  class  Solution :
 7      def isValidSudoku(self, board: List[List[str]])  −> bool:
 8          return ( self .is_row_valid(board) and
 9                   self .is_col_valid(board) and
10                   self .is_square_valid(board))
11
12      def is_row_valid(self, board):
13          for  row in board:
14              if not self .is_unit_valid(row):
15                  return False
16          return True
17
18      def is_col_valid( self , board):
19          # 列转化成行
20          for  col in zip(∗board):
21              if not self .is_unit_valid(col):
22                  return False
23          return True
24
25      def is_square_valid(self, board):
26          for  i in (0, 3, 6):
27              for j in (0, 3, 6):
28                  square = [board[x][y] for x in range(i, i + 3) for y in range(j, j + 3)]
29                  if not self .is_unit_valid(square):
30                      return False
31          return True
32
33      def is_unit_valid( self , unit ):
34          unit = [i for i in unit if i != '.']
```

```
35              return len(set(unit)) == len(unit)
```

```
1    #
2    # @lc app=leetcode.cn id=37 lang=python3
3    #
4    # [37] 解数独
5    #
6    class Solution:
7        def solveSudoku(self, board: List[List[str]]) -> None:
8            self.dfs(board)
9
10       def dfs(self,board):
11           for i in range(9):
12               for j in range(9):
13                   if board[i][j] == '.':
14                       for k in '123456789':
15                           board[i][j] = k
16                           # 修改一个值判断是不是合法的
17                           # 如果这个递归可以返回true并且当前填入的数字也没毛病
18                           # 则证明我们解完了数独
19                           if self.isOK(board,i,j) and self.dfs(board):
20                               return True
21                           board[i][j] = '.'
22                       return False
23           # 全部填完之后返回True
24           return True
25
26       def isOK(self,board,x,y):
27           #列符合
28           for i in range(9):
29               if i != x and board[i][y] == board[x][y]:
30                   return False
31           #检查行是否符合
32           for j in range(9):
33               if j != y and board[x][j] == board[x][y]:
34                   return False
35           row_start = 3*(x // 3)
36           col_start = 3*(y // 3)
37           for i in range(row_start,row_start+3):
38               for j in range(col_start,col_start+3):
39                   if (i!= x or j!= y) and board[i][j] == board[x][y]:
40                       return False
41           return True
```

```
1    #
2    # @lc app=leetcode.cn id=38 lang=python3
```

```python
#
# [38] 外观数列
#
class Solution:
    def countAndSay(self, n: int) -> str:
        s = '1'
        for _ in range(n-1):
            s = self.count(s)
        return s

    def count(self, s):
        m = list(s)
        # 加一个后面不会溢出(随便加一个就行)
        m.append(5)
        res = ()
        i, j = 0, 0
        while i < len(m)-1:
            j += 1
            if m[j] != m[i]:
                res += (str(j-i), m[i])
                i = j
        # 用空元素链接res
        return ''.join(res)
```

```python
#
# @lc app=leetcode.cn id=39 lang=python3
#
# [39] 组合总和
#
class Solution:
    def combinationSum(self, candidates: List[int], target: int) -> List[List[int]]:
        candidates.sort()
        res = []
        self.dfs(candidates, target, 0, [], res)
        return res

    def dfs(self, nums, target, index, path, res):
        if target < 0:
            return
        if target == 0:
            res.append(path)
            return
        for i in range(index, len(nums)):
            self.dfs(nums, target-nums[i], i, path+[nums[i]], res)
```

```python
#
```

```python
# @lc app=leetcode.cn id=40 lang=python3
#
# [40] 组合总和 II
#
class Solution:
    def combinationSum2(self, candidates: List[int], target: int) -> List[List[int]]:
        candidates.sort()
        res = []
        self.combine_sum_2(candidates, target, 0, [], res)
        return res

    def combine_sum_2(self, nums, target, start, path, res):
        # 超过了
        if target < 0:
            return
        if target == 0:
            res.append(path)
            return

        for i in range(start, len(nums)):
            # 解集不重复
            if i > start and nums[i] == nums[i - 1]:
                continue
            self.combine_sum_2(nums, target - nums[i],
                    i + 1, path + [nums[i],], res)
```

```python
#
# @lc app=leetcode.cn id=41 lang=python3
#
# [41] 缺失的第一个正数
#
class Solution:
    def firstMissingPositive(self, nums: List[int]) -> int:
        self.bucket_sort(nums)

        for i in range(len(nums)):
            if nums[i] != (i+1):
                return i+1
        return len(nums)+1

    def bucket_sort(self, nums):
        for i in range(len(nums)):
            while 0 <= nums[i] < len(nums) and nums[i] != nums[nums[i]-1]:
                temp = nums[i]-1
                nums[i] = nums[temp]
                nums[temp] = temp +1
```

```
21          return
```

```python
#
# @lc app=leetcode.cn id=42 lang=python3
#
# [42] 接雨水
#
class Solution:
    def trap(self, height: List[int]) -> int:
        if not height:   # 边界检查
            return 0
        l ,r = 0 , len(height) - 1

        res = 0
        l_max , r_max = 0 , 0
        while l < r :
            if height[l] < height[r]:
                if height[l] >= l_max:
                    l_max = height[l]
                else:
                    res += l_max - height[l]
                l += 1
            else :
                if height[r] >= r_max:
                    r_max = height[r]
                else:
                    res += r_max - height[r]

                r -= 1
        return res
```

```python
#
# @lc app=leetcode.cn id=43 lang=python3
#
# [43] 字符串相乘
#
class Solution:
    def multiply(self, num1: str, num2: str) -> str:

        #把num1,num2翻转方便计算
        num1 = num1[::-1]; num2 = num2[::-1]
        #每一位互相乘的结果用一维数组去储存
        arr = [0 for i in range(len(num1)+len(num2))]
        #填充这个一维数组
        for i in range(len(num1)):
            for j in range(len(num2)):
```

```python
16              arr[i+j] += int(num1[i]) * int(num2[j])
17
18          res = []
19          # arr是反的
20          #计算每一位的终极结果
21          for i in range(len(arr)):
22              #digit表示这一位的数字
23              digit = arr[i] % 10
24              #carry表示加给下一位的量
25              carry = arr[i] // 10
26              if i < len(arr)-1:
27                  #下一位加上
28                  arr[i+1] += carry
29              #更新答案
30              res.insert(0, str(digit))
31          #去除首位为0的情况
32          while res[0] == '0' and len(res) > 1:
33              res.pop(0)
34          #连接成字符串
35          return ''.join(res)
```

```python
1   #
2   # @lc app=leetcode.cn id=45 lang=python3
3   #
4   # [45] 跳跃游戏 II
5   #
6   class Solution:
7       def jump(self, nums: List[int]) -> int:
8           if len(nums) <= 1:
9               return 0
10          # (start -> end )
11          end = nums[0]
12          start = 0
13          step = 1
14          maxDis = nums[0]
15          while end < len(nums) - 1:
16              # 看一步最远能走到哪
17              for i in range(start + 1, end + 1):
18                  maxDis = max(maxDis, nums[i] + i)
19              start = end
20              end = maxDis
21              step += 1
22          return step
```

```python
1   #
2   # @lc app=leetcode.cn id=46 lang=python3
```

```python
#
# [46] 全排列
#
class Solution:
    def permute(self, nums: List[int]) -> List[List[int]]:
        #nums.sort()
        res = []
        self.dfs(nums, [], res)
        return res

    def dfs(self, nums, path, res):
        if not nums:
            # nums已经全部压入到path里面了
            res.append(path)
            return
        for i in range(len(nums)):
            self.dfs(nums[:i]+nums[i+1:], path+[nums[i]], res)
```

```python
#
# @lc app=leetcode.cn id=47 lang=python3
#
# [47] 全排列 II
#
class Solution:
    def permuteUnique(self, nums: List[int]) -> List[List[int]]:
        res = []
        self.dfs(nums, [], res)
        return res

    def dfs(self, nums, path, res):
        if not nums and path not in res:
            # nums已经全部压入到path里面了
            res.append(path)
            return
        for i in range(len(nums)):
            self.dfs(nums[:i]+nums[i+1:], path+[nums[i]], res)
```

```python
#
# @lc app=leetcode.cn id=48 lang=python3
#
# [48] 旋转图像
#
class Solution:
    def rotate(self, matrix: List[List[int]]) -> None:
        if matrix is None or len(matrix) == 1:
            return
```

```
10        ls = len(matrix)
11
12        for i in range(ls // 2):
13            # 那一圈的半行
14            begin, end = i, ls - 1 - i # 左右都往内部i个单位
15            for k in range(ls- 1 - 2 * i ): # 减两个i的单位
16                # 顺着转
17                temp = matrix[end - k][begin] # 左下角
18                matrix[end - k][begin] = matrix[end][end - k] # 右下角给左下角
19                matrix[end][end - k] = matrix[begin + k][end] # 右上角给右下角
20                matrix[begin + k][end] = matrix[begin][begin + k] # 左上角给右上角
21                matrix[begin][begin + k] = temp # 左下角给左上角
22        return
```

```
1  #
2  # @lc app=leetcode.cn id=49 lang=python3
3  #
4  # [49] 字母异位词分组
5  #
6  class Solution:
7      def groupAnagrams(self, strs: List [ str ]) -> List[List[str ]]:
8          dic = {}
9          # key是单词对应的元素
10         # value是字符串
11         for word in strs :
12             key = ''.join(sorted(word))
13             if key not in dic :
14                 dic [key] = []
15             dic [key].append(word)
16         res = []
17         for i in dic:
18             res .append(dic[i])
19         return res
```

```
1  #
2  # @lc app=leetcode.cn id=50 lang=python3
3  #
4  # [50] Pow(x, n)
5  #
6  class Solution:
7      def myPow(self, x: float , n: int ) -> float:
8          if n == 0:
9              return 1
10         if n < 0:
11             return 1 / self .myPow(x, -n)
12         if n % 2:
```

```
13            return x * self.myPow(x, n-1)
14        return self.myPow(x*x, n // 2)
```

```
1  #
2  # @lc app=leetcode.cn id=51 lang=python3
3  #
4  # [51] N皇后
5  #
6  class Solution:
7      def solveNQueens(self, n: int) -> List[List[str]]:
8          result = []
9          # C[i]表示第i行皇后在哪一列
10         C = [-1 for _ in range(n)]
11         self.dfs(C,result,0)
12         return result
13
14     def dfs(self,C,res,row):
15         N = len(C)
16         # 终止条件
17         if N == row:
18             path = [["." for _ in range(N)] for _ in range(N)]
19             for i in range(N):
20                 # (i,C[i])位置对应皇后
21                 path[i][C[i]] = "Q"
22             path = ["".join(r) for r in path]
23             # if path not in res:
24             # 不用排除
25             res.append(path)
26             return
27         # 对该行每一列都进行尝试,可以的话下一行
28         for j in range(N):
29             if j not in C and self.isOK(C,row,j):
30                 C[row] = j
31                 self.dfs(C,res,row+1)
32                 C[row] = -1
33
34     # 对该行之前的都进行判断,返回合理与否
35     def isOK(self,C,row,col):
36         for i in range(row):
37             # 同一列
38             # 同一对角线
39             if C[i] == col or abs(i-row) == abs(C[i]-col):
40                 return False
41         return True
```

```
1  #
```

```python
# @lc app=leetcode.cn id=52 lang=python3
#
# [52] N皇后 II
#
class Solution:
    def totalNQueens(self, n: int) -> int:
        self.res = 0
        # C[i]表示第i行皇后在哪一列
        C = [-1 for _ in range(n)]
        self.dfs(C,0)
        return self.res

    def dfs(self,C,row):
        N = len(C)
        # 终止条件
        if N == row:
            # 不用排除
            self.res += 1
        # 对该行每一列都进行尝试,可以的话下一行
        for j in range(N):
            if j not in C and self.isOK(C,row,j):
                C[row] = j
                self.dfs(C,row+1)
                C[row] = -1

    # 对该行之前的都进行判断,返回合理与否
    def isOK(self,C,row,col):
        for i in range(row):
            # 同一列
            # 同一对角线
            if C[i] == col or abs(i-row) == abs(C[i]-col):
                return False
        return True
```

```python
#
# @lc app=leetcode.cn id=53 lang=python3
#
# [53] 最大子序和
#
class Solution:
    def maxSubArray(self, nums: List[int]) -> int:
        temp = maxsum = nums[0]
        for num in nums[1:]:
            # num 要么单独一个子列,要么归入别的子列
            temp = max(num,temp+num)
            maxsum = max(temp,maxsum)
```

```
13          return maxsum
```

```python
#
# @lc app=leetcode.cn id=54 lang=python3
#
# [54] 螺旋矩阵
#
class Solution:
    def spiralOrder(self, matrix: List[List[int]]) -> List[int]:
        if not matrix:
            return []

        '''
        # 常规方法太烦了
        res = []
        xbegin = ybegin = 0
        xend = len(matrix[0]) - 1
        yend = len(matrix) - 1
        while True:
            # 横
            for j in range(xbegin,xend+1):
                res.append(matrix[ybegin][j])
            ybegin += 1
            if ybegin > yend:
                break
            # 竖
            for j in range(ybegin,yend+1):
                res.append(matrix[j][xend])
            xend -= 1
            if xbegin > xend:
                break
            # 横
            for j in range(xend,xbegin-1,-1):
                res.append(matrix[yend][j])
            yend -= 1
            if ybegin > yend:
                break
            # 竖
            for j in range(yend,ybegin-1,-1):
                res.append(matrix[j][xbegin])
            xbegin += 1
            if xbegin > xend:
                break
        return res
        '''
```

```python
45          m,n = len(matrix),len(matrix[0])
46          x = y = di = 0
47          dx = [0,1,0,-1]
48          dy = [1,0,-1,0]
49          res = []
50          visited = set()
51
52          for i in range(m*n):
53              res.append(matrix[x][y])
54              visited.add((x,y))
55              nx,ny = x+dx[di],y+dy[di]
56              if 0<=nx<m and 0<=ny<n and (nx,ny) not in visited:
57                  x,y = nx,ny
58              else:
59                  di = (di+1)%4 # 如果不满足条件，换一个方向进行遍历
60                  x,y = x+dx[di],y+dy[di]
61          return res
```

```python
1  #
2  # @lc app=leetcode.cn id=55 lang=python3
3  #
4  # [55] 跳跃游戏
5  #
6  class Solution:
7      def canJump(self, nums: List[int]) -> bool:
8          start = end = 0
9          while start <= end < len(nums) - 1:
10             end = max(end, nums[start] + start)
11             start += 1
12         return end >= len(nums) - 1
```

```python
1  #
2  # @lc app=leetcode.cn id=56 lang=python3
3  #
4  # [56] 合并区间
5  #
6  class Solution:
7      def merge(self, intervals : List[List[int]]) -> List[List[int]]:
8          if len(intervals) <= 1:
9              return intervals
10         res = []
11         intervals.sort(key = lambda x: x[0])
12         s , e = intervals[0][0] , intervals[0][1]
13
14         for i in range(1,len(intervals)):
15             # 后边跟着的区间和[s,e]的交叉,相当于合并
```

```python
16          if e >= intervals[i][0] :
17              e = max( e , intervals[i][1] )
18          # 紧跟着的区间在[s,e]后面
19          else :
20              res.append([s,e])
21              s ,e = intervals[i][0] , intervals[i][1]
22      res.append([s,e])
23      return res
```

```python
1  #
2  # @lc app=leetcode.cn id=57 lang=python3
3  #
4  # [57] 插入区间
5  #
6  class Solution:
7      def insert ( self , intervals : List [List [int ]], newInterval: List [int ]) -> List[List[int ]]:
8          s, e = newInterval[0], newInterval[1]
9          left , right = [], []
10         for inter in intervals :
11             # 左边部分
12             if s > inter [1]:
13                 left .append(inter)
14             # 右边部分
15             elif e < inter [0]:
16                 right .append(inter)
17             # 和区间交叉部分,合并
18             else :
19                 s = min(s, inter [0])
20                 e = max(e, inter [1])
21         return left + [[s, e]] + right
```

```python
1  #
2  # @lc app=leetcode.cn id=58 lang=python3
3  #
4  # [58] 最后一个单词的长度
5  #
6  class Solution:
7      def lengthOfLastWord(self, s: str ) -> int:
8          if not s :
9              return 0
10         tmp = s.split( '␣' )
11         tmp = [t for t in tmp if len(t) > 0]
12         if len(tmp) == 0:
13             return 0
14         else :
15             return len(tmp[−1])
```

```
#
# @lc app=leetcode.cn id=59 lang=python3
#
# [59] 螺旋矩阵 II
#
class Solution:
    def generateMatrix(self, n: int) -> List[List[int]]:
        mat = [[0 for _ in range(n)] for _ in range(n)]

        b,e = 0 , n - 1
        val = 1
        while b < e :
            # 横
            for i in range(b,e):
                mat[b][i] = val
                val += 1
            # 竖
            for i in range(b,e):
                mat[i][e] = val
                val += 1
            # 横
            for i in range(e,b,-1):
                mat[e][i] = val
                val += 1
            # 竖
            for i in range(e,b,-1):
                mat[i][b] = val
                val += 1
            b += 1
            e -= 1

        # n为奇数,中间还有一个值
        if n % 2 :
            mat[b][e] = val
        return mat
```

```
#
# @lc app=leetcode.cn id=60 lang=python3
#
# [60] 第k个排列
#
class Solution:
    def getPermutation(self, n: int, k: int) -> str:
        # 待选择的字符串
        nums = [str(i) for i in range(1,n+1)]
        # 0!, 1!, ..., (n - 1)!
```

```python
        factorials  = [1]
        for i in range(1, n):
            factorials .append(factorials[i − 1] ∗ i)

        # 第几个转化为第几个的索引(减1)
        k −= 1

        res = []
        for i in range(n − 1, −1, −1):
            # 计算第几个区间,首位所在的区间 k//(n−1)!
            # 第一个区间首位是1,第二个区间首位是2
            idx = k // factorials [i]
            # 减去多个区间对应的值
            k −= idx ∗ factorials[i]
            # 结果值添加对应的数字
            res .append(nums[idx])
            # 因为排列不重复,nums需要去掉对应元素
            nums.pop(idx)

        return ''.join(res)
```

```python
#
# @lc app=leetcode.cn id=61 lang=python3
#
# [61] 旋转链表
#
# Definition for  singly−linked  list .
# class ListNode:
#     def __init__(self, x):
#         self .val = x
#         self .next = None

class Solution:
    def rotateRight( self , head: ListNode, k: int ) −> ListNode:
        if  head is  None or k == 0:
            return head

        pointer = head
        length = 1
        while pointer.next:
            pointer = pointer.next
            length += 1

        # 左部分多少个
        k = length − k%length

```

```
26          # 连成一个环
27          pointer.next = head
28
29          for i in range(k):
30              pointer = pointer.next
31
32          # 断开
33          head = pointer.next
34          pointer.next = None
35          return head
```

```
1  #
2  # @lc app=leetcode.cn id=62 lang=python3
3  #
4  # [62] 不同路径
5  #
6  class Solution:
7      def uniquePaths(self, m: int, n: int) -> int:
8          mat = [[0 for _ in range(n)] for _ in range(m)]
9          for r in range(m):
10             mat[r][0] = 1
11         for c in range(n):
12             mat[0][c] = 1
13         for r in range(1,m):
14             for c in range(1,n):
15                 mat[r][c] = mat[r-1][c] + mat[r][c-1]
16         return mat[-1][-1]
```

```
1  #
2  # @lc app=leetcode.cn id=63 lang=python3
3  #
4  # [63] 不同路径 II
5  #
6  class Solution:
7      def uniquePathsWithObstacles(self, obstacleGrid: List[List[int]]) -> int:
8          if not obstacleGrid:
9              return
10         r, c = len(obstacleGrid), len(obstacleGrid[0])
11         mat = [[0 for _ in range(c)] for _ in range(r)]
12         # 到起点看这里有没有问题
13         mat[0][0] = 1 - obstacleGrid[0][0]
14
15         for i in range(1, r):
16             mat[i][0] = mat[i-1][0] * (1 - obstacleGrid[i][0])
17         for i in range(1, c):
18             mat[0][i] = mat[0][i-1] * (1 - obstacleGrid[0][i])
```

```python
19
20          for i in range(1, r):
21              for j in range(1, c):
22                  mat[i][j] = (mat[i][j−1] + mat[i−1][j]) * (1 − obstacleGrid[i][j])
23          return mat[−1][−1]
```

```python
1  #
2  # @lc app=leetcode.cn id=64 lang=python3
3  #
4  # [64] 最小路径和
5  #
6  class Solution:
7      def minPathSum(self, grid: List[List[int]]) −> int:
8          m,n = len(grid),len(grid[0])
9          dp = [[0 for _ in range(n)] for _ in range(m)]
10         dp[0][0] = grid[0][0]
11         for r in range(1,m):
12             dp[r][0] = dp[r−1][0] + grid[r][0]
13         for c in range(1,n):
14             dp[0][c] = dp[0][c−1] + grid[0][c]
15         for r in range(1,m):
16             for c in range(1,n):
17                 dp[r][c] = min(dp[r−1][c] , dp[r][c−1])+grid[r][c]
18         return dp[m−1][n−1]
```

```python
1  #
2  # @lc app=leetcode.cn id=66 lang=python3
3  #
4  # [66] 加一
5  #
6  class Solution:
7      def plusOne(self, digits: List[int]) −> List[int]:
8          '''
9          # 数值操作
10         num = 0
11         for i in range(len(digits)):
12             num = num * 10 + digits[i]
13         num = num + 1
14         res = []
15         while num > 0:
16             res.append(num%10)
17             num //= 10
18         res.reverse()
19         return res
20         '''
21
```

```
22            # 列表操作
23            digits[-1] += 1
24            digits.insert(0, 0)
25            for i in range(len(digits)-1,0,-1):
26                carry = digits[i] // 10
27                digits[i] %= 10
28                digits[i-1] += carry
29
30            if digits[0] == 0:
31                digits.pop(0)
32
33            return digits
```

```
1  #
2  # @lc app=leetcode.cn id=67 lang=python3
3  #
4  # [67] 二进制求和
5  #
6  class Solution:
7      def addBinary(self, a: str, b: str) -> str:
8          if not a:
9              return b
10         if not b:
11             return a
12         # 最后都是1 前面的相加 再加1 补0
13         if a[-1] == '1' and b[-1] == '1':
14             return self.addBinary(self.addBinary(a[0:-1],b[0:-1]),'1')+'0'
15         # 最后都是0 补0
16         if a[-1] == '0' and b[-1] == '0':
17             return self.addBinary(a[0:-1],b[0:-1])+'0'
18         # 最后一个1 一个0 补1
19         else:
20             return self.addBinary(a[0:-1],b[0:-1])+'1'
```

```
1  #
2  # @lc app=leetcode.cn id=69 lang=python3
3  #
4  # [69] x 的平方根
5  #
6  class Solution:
7      def mySqrt(self, x: int) -> int:
8          l ,r = 0 , x
9          while l <= r:
10             mid = (l+r)//2
11             if mid**2 <= x < (mid+1)**2:
12                 return mid
```

```
13          elif  x < mid**2:
14              r = mid
15          else :
16              l = mid+1
```

```
1   #
2   # @lc app=leetcode.cn id=70 lang=python3
3   #
4   # [70] 爬楼梯
5   #
6   class  Solution :
7       def  climbStairs( self , n: int )  −> int:
8           if  n == 1:
9               return  1
10          # 初始的两个   输入1 or 2
11          a, b = 1, 2
12          # 从n大于3开始
13          for  i  in  range(2, n):
14              b ,a = a+b , b
15          return  b
```

```
1   #
2   # @lc app=leetcode.cn id=71 lang=python3
3   #
4   # [71] 简化路径
5   #
6   class  Solution :
7       def  simplifyPath( self , path: str )  −> str:
8           res = []
9           for  child  in  path. split ( '/'):
10              if  child  in  ('' , '.'):
11                  pass
12              elif  child  == '..':
13                  if  res :res .pop()
14              else :
15                  res .append(child)
16          return  '/' + '/'. join (res )
```

```
1   #
2   # @lc app=leetcode.cn id=72 lang=python3
3   #
4   # [72] 编辑距离
5   #
6   class  Solution :
7       def  minDistance(self, word1: str , word2: str )  −> int:
8           l1, l2 = len(word1)+1, len(word2)+1
9           dp = [[0 for  _ in range(l2)]  for  _ in range(l1)]
```

```python
           # 行列处理 对应从空到一个字符串 或 一个字符串到空
        for i in range(l1):
            dp[i][0] = i
        for j in range(l2):
            dp[0][j] = j
        for i in range(1, l1):
            for j in range(1, l2):
                if word1[i-1]==word2[j-1]:
                    dp[i][j] = dp[i-1][j-1]
                else:
                    # 三个分别对应于加、减、替换
                    dp[i][j] = min(dp[i-1][j],
                                   dp[i][j-1],
                                   dp[i-1][j-1]
                                   )+1
        return dp[-1][-1]
```

```python
#
# @lc app=leetcode.cn id=73 lang=python3
#
# [73] 矩阵置零
#
class Solution:
    def setZeroes(self, matrix: List[List[int]]) -> None:
        '''
        # 直接法
        row = []
        col = []
        m = len(matrix)
        n = len(matrix[0])
        for i in range(m):
            for j in range(n):
                if matrix[i][j] == 0:
                    row.append(i)
                    col.append(j)
        row = set(row)
        col = set(col)

        for i in row:
            for j in range(n):
                matrix[i][j] = 0
        for j in col:
            for i in range(m):
                matrix[i][j] =0

        return matrix
```

```python
30          '''
31          # 第一行出现一个0
32          firstRowHasZero = not all(matrix[0])
33          m = len(matrix)
34          n = len(matrix[0])
35          # 第一行第一列做标记
36          for i in range(1,m):
37              for j in range(n):
38                  if matrix[i][j] == 0:
39                      matrix[0][j] = matrix[i][0] = 0
40          # 置0
41          for i in range(1,m):
42              for j in range(n-1,-1,-1):
43                  if matrix[i][0] == 0 or matrix[0][j] == 0:
44                      matrix[i][j] = 0
45          # 补一下第一行的
46
47          if firstRowHasZero:
48              matrix[0] = [0] * n
49
50          return matrix
```

```python
1   #
2   # @lc app=leetcode.cn id=74 lang=python3
3   #
4   # [74] 搜索二维矩阵
5   #
6   class Solution:
7       def searchMatrix(self, matrix: List[List[int]], target: int) -> bool:
8           if len(matrix)==0 or len(matrix[0])==0 or target < matrix[0][0] or target > matrix
                  [-1][-1]:
9               return False
10          row = 0
11          col = len(matrix[0]) -1
12          while row < len(matrix) and col >= 0 :
13              if matrix[row][col] > target:
14                  col -= 1
15              elif matrix[row][col] < target:
16                  row += 1
17              else :
18                  return True
19          return False
```

```python
1   #
2   # @lc app=leetcode.cn id=75 lang=python3
3   #
```

```
# [75] 颜色分类
#
class Solution:
    def sortColors(self, nums: List[int]) -> None:
        count = [0,0,0]
        for num in nums:
            count[num] += 1
        idx = 0
        for i in range(3):
            for j in range(count[i]):
                nums[idx] = i
                idx += 1
```

```
#
# @lc app=leetcode.cn id=76 lang=python3
#
# [76] 最小覆盖子串
#
class Solution:
    def minWindow(self, s: str, t: str) -> str:
        if s is None or len(s) < len(t):
            return ""
        res = ""
        left = 0
        right = 0
        min_len = len(s)
        count = 0

        m = {}
        # 统计t中字符数目
        for i in t:
            m[i] = m.get(i,0) + 1

        while right < len(s):
            if s[right] in m:
                # 先找到一个区间能包含t,但长度不一定是最短的
                m[s[right]] -= 1
                if m[s[right]] >= 0:
                    count += 1
                # 找到了一个区间
                while (count == len(t)):
                    # 选择更短的子串
                    if (right - left + 1 < min_len):
                        min_len = right-left+1
                        res = s[left:right+1]
```

```python
                    if s[left] in m:
                        m[s[left]] += 1
                        if m[s[left]] > 0:
                            count -= 1
                    left += 1
            right += 1

        return res
```

```python
#
# @lc app=leetcode.cn id=77 lang=python3
#
# [77] 组合
#
class Solution:
    def combine(self, n: int, k: int) -> List[List[int]]:
        res = []
        self.dfs(n,k,1,[], res)
        return res

    def dfs(self,n,k,start,path,res):
        if 0 == k and path not in res:
            res.append(path)
        for i in range(start,n+1):
            self.dfs(n,k-1,i+1,path+[i],res)
```

```python
#
# @lc app=leetcode.cn id=78 lang=python3
#
# [78] 子集
#
class Solution:
    def subsets(self, nums: List[int]) -> List[List[int]]:
        res = []
        nums.sort()
        self.dfs(nums, 0, [], res)
        return res

    def dfs(self, nums, index, path, res):
        res.append(path)
        for i in range(index, len(nums)):
            self.dfs(nums, i+1, path+[nums[i]], res)
```

```python
#
# @lc app=leetcode.cn id=79 lang=python3
#
# [79] 单词搜索
```

```
 5  #
 6  class Solution:
 7      def exist(self, board: List[List[str]], word: str) -> bool:
 8          m, n = len(board), len(board[0])
 9          visited = [[False for i in range(n)] for i in range(m)]
10          # 遍历寻找开头
11          for i in range(m):
12              for j in range(n):
13                  if self.dfs(board,word,visited,i,j,0):
14                      return True
15          return False
16
17      def dfs(self,board,word,visited,i,j,start):
18          # 终止条件
19          if start == len(word):
20              return True
21          # 溢出 剪枝 or 已经访问过了
22          if i < 0 or j < 0 or i >= len(board) or j >= len(board[0]) or visited[i][j] or board[i][j]
                  != word[start]:
23              return False
24
25          if board[i][j] == word[start]:
26              visited[i][j] = True
27              ret = self.dfs(board,word,visited,i+1,j,start+1) or \
28                      self.dfs(board,word,visited,i-1,j,start+1) or \
29                      self.dfs(board,word,visited,i,j+1,start+1) or \
30                      self.dfs(board,word,visited,i,j-1,start+1)
31              visited[i][j] = False
32
33              return ret
```

```
 1  #
 2  # @lc app=leetcode.cn id=80 lang=python3
 3  #
 4  # [80] 删除排序数组中的重复项 II
 5  #
 6  class Solution:
 7      def removeDuplicates(self, nums: List[int]) -> int:
 8          if not nums:
 9              return 0
10          # 初始化第一个
11          i, count = 1, 1
12
13          while i < len(nums):
14              if nums[i] == nums[i - 1]:
15                  count += 1
```

```
16            if count > 2:
17                nums.pop(i)
18                # 这里的减一和后面对消
19                i -= 1
20            else:
21                count = 1
22            i += 1
23        return len(nums)
```

```
1  #
2  # @lc app=leetcode.cn id=81 lang=python3
3  #
4  # [81] 搜索旋转排序数组 II
5  #
6  class Solution:
7      def search(self, nums: List[int], target: int) -> bool:
8          if not nums:
9              return False
10         l,r = 0 , len(nums) -1
11
12         while l <= r:
13             mid = (l+r)//2
14             if nums[mid] == target:
15                 return True
16             # mid在前半段 或者l mid r 都在右边
17             if nums[l] < nums[mid]:
18                 if nums[l] <= target < nums[mid]:
19                     r = mid -1
20                 else:
21                     l = mid +1
22             # l 在左半段 、mid 在后半段
23             elif nums[mid] < nums[l]:
24                 if nums[mid] < target <= nums[r]:
25                     l = mid +1
26                 else:
27                     r = mid -1
28             else:
29                 l += 1
30         return False
```

```
1  #
2  # @lc app=leetcode.cn id=82 lang=python3
3  #
4  # [82] 删除排序链表中的重复元素 II
5  #
6  # Definition for singly-linked list.
```

```python
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution:
    def deleteDuplicates(self, head: ListNode) -> ListNode:
        dummy = ListNode(0)
        dummy.next = head
        prev = dummy

        while head and head.next:
            if head.val == head.next.val:
                while head and head.next and head.val == head.next.val:
                    head = head.next
                head = head.next
                prev.next = head
            # 两个指针都往后走
            else:
                prev = prev.next
                head = head.next
        return dummy.next
```

```python
#
# @lc app=leetcode.cn id=83 lang=python3
#
# [83] 删除排序链表中的重复元素
#
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution:
    def deleteDuplicates(self, head: ListNode) -> ListNode:
        point = head
        while point:
            while point.next and point.val == point.next.val:
                point.next = point.next.next
            point = point.next
        return head
```

```python
#
# @lc app=leetcode.cn id=84 lang=python3
#
```

```python
# [84] 柱状图中最大的矩形
#
class Solution:
    def largestRectangleArea(self, heights: List[int]) -> int:
        # 此处较为巧妙。若heights数组中元素都是单增序列，则最后无法出栈stack，也就无法计算
        #       最大面积，所以补个0，使之最后可以出栈
        heights.append(0)
        stack = [-1]
        res = 0

        for idx, val in enumerate(heights):
            # 不是递增栈
            while heights[stack[-1]] > val:
                h = heights[stack.pop()]
                w = idx - stack[-1] -1
                res = max(res, h*w)
            stack.append(idx)
        return res
```

```python
#
# @lc app=leetcode.cn id=85 lang=python3
#
# [85] 最大矩形
#
class Solution:
    def maximalRectangle(self, matrix: List[List[str]]) -> int:
        '''
        if not matrix or not matrix[0]:
            return 0
        m , n = len(matrix),len(matrix[0])
        # height 的尾部多了一个0,防止递增错误
        height = [0] * (n+1)
        max_area = 0
        for i in range(m):
            # 计算h
            for j in range(n):
                # 遍历到的每行的h
                height[j] = height[j]+1 if matrix[i][j]=='1' else 0
            # 找出所有h和w的组合
            # 同84题
            stack = [-1]
            for k in range(n + 1):
                while height[k] < height[stack[-1]]:
                    h = height[stack.pop()]
                    w = k - stack[-1] - 1
                    max_area = max(max_area, h * w)
```

```python
                    stack.append(k)
            return max_area
            '''
        if not matrix or not matrix[0]:
            return 0
        m , n = len(matrix),len(matrix[0])
        # 申请辅助数组并初始化
        # 向上、向左、向右能延伸到的最远的地方
        left , right, height = [0]*n, [n]*n, [0]*n
        max_A = 0
        # 从第一行开始遍历
        for i in range(m):
            # 用来记录下标
            cur_left, cur_right = 0, n
            # 从第一个元素开始遍历
            for j in range(n):
                # 如果矩阵中当前坐标为1时，我们将height对应的下标加一
                # left取cur_left和left[i]中取最大的
                if matrix[i][j] == "1":
                    height[j] = height[j] + 1
                    left[j] = max(left[j], cur_left)
                else: # 否则赋值位0
                    height[j], left[j] = 0, 0
                    cur_left = j+1
            # right数组从末尾开始遍历
            for j in range(n-1, -1, -1):
                if matrix[i][j] == "1":
                    right[j] = min(right[j], cur_right)
                else:
                    right[j] = n
                    cur_right = j
            for j in range(n):
                # 计算到前行为止最大的面积
                max_A = max(max_A,(right[j]-left[j])*height[j])
        return max_A
```

```python
#
# @lc app=leetcode.cn id=86 lang=python3
#
# [86] 分隔链表
#
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None
```

```
11
12  class Solution:
13      def partition(self, head: ListNode, x: int) -> ListNode:
14          h1 = l1 = ListNode(0)
15          h2 = l2 = ListNode(0)
16
17          while head:
18              if head.val < x:
19                  l1.next = head
20                  l1 = l1.next
21              else:
22                  l2.next = head
23                  l2 = l2.next
24              head = head.next
25          # l1 l2都在各自的尾部了
26          l2.next = None
27          l1.next = h2.next
28
29          return h1.next
```

```
1  #
2  # @lc app=leetcode.cn id=88 lang=python3
3  #
4  # [88] 合并两个有序数组
5  #
6  class Solution:
7      def merge(self, nums1: List[int], m: int, nums2: List[int], n: int) -> None:
8          # 从后往前
9          p1 = m - 1
10         p2 = n - 1
11         p = m + n -1
12         # 两个都没放完
13         while p1 >= 0 and p2 >= 0:
14             if nums1[p1] >= nums2[p2]:
15                 nums1[p] = nums1[p1]
16                 p1 -= 1
17             else:
18                 nums1[p] = nums2[p2]
19                 p2 -= 1
20             p -= 1
21         # p1没放完，那就不用再操作了
22         # p2没放完
23         while p2 >= 0:
24             nums1[p] = nums2[p2]
25             p -= 1
26             p2 -= 1
```

```
1   #
2   # @lc app=leetcode.cn id=89 lang=python3
3   #
4   # [89] 格雷编码
5   #
6   class Solution:
7       def grayCode(self, n: int) -> List[int]:
8           res = [0]
9           for i in range(n):
10              for j in range(len(res)-1,-1,-1):
11                  res.append(res[j] + (1 << i))
12          return res
```

```
1   #
2   # @lc app=leetcode.cn id=90 lang=python3
3   #
4   # [90] 子集 II
5   #
6   class Solution:
7       def subsetsWithDup(self, nums: List[int]) -> List[List[int]]:
8           res = []
9           nums.sort()
10          # self.dfs(nums, 0, [], res)
11          self.dfs2(nums, 0, [], res)
12          return res
13
14      def dfs(self, nums, index, path, res):
15          if path not in res:
16              res.append(path)
17          for i in range(index, len(nums)):
18              self.dfs(nums, i+1, path+[nums[i]], res)
19
20      def dfs2(self, nums, index, path, res):
21          res.append(path)
22          for i in range(index, len(nums)):
23              if i > index and nums[i] == nums[i-1]:
24                  continue
25              self.dfs2(nums, i+1, path+[nums[i]], res)
```

```
1   #
2   # @lc app=leetcode.cn id=91 lang=python3
3   #
4   # [91] 解码方法
5   #
6   class Solution:
7       def numDecodings(self, s: str) -> int:
```

```python
 8          if  s  is  None or s[0]  ==  '0':
 9              return  0
10          # dp[i] 表示s中前i个字符组成的子串的解码方法的个数，长度比输入数组长多多1，并将 dp
               [0] 初始化为1
11          dp = [0] * (len(s)+1)
12          dp[0]  = dp[1] = 1
13          for  i  in  range(2,len(s)+1):
14              if  s[i − 1] >= '1' and s[i − 1] <= '9':
15                  dp[i]  += dp[i − 1]
16              if  s[i−2]=='1' or (s[i−2] == '2' and s[i−1] <= '6'):
17                  dp[i]  += dp[i − 2]
18          return  dp[−1]
```

```python
 1  #
 2  # @lc app=leetcode.cn id=92 lang=python3
 3  #
 4  # [92] 反转链表 II
 5  #
 6  # Definition for  singly−linked  list .
 7  # class  ListNode:
 8  #     def ___init___(self, x):
 9  #         self . val  = x
10  #         self . next = None
11
12  class  Solution :
13      def reverseBetween(self,  head: ListNode, m: int,  n:  int )  −> ListNode:
14          dummy = ListNode(0)
15          dummy.next = head
16          prev = dummy
17          # 走m−1个
18          for  i  in  range(m−1):
19              prev = prev.next
20          # 反转
21          temp = None
22          cur = prev.next
23          for  i  in  range(n−m+1):
24              next = cur.next
25              # reverse
26              cur.next = temp
27              temp = cur
28              # 下一个
29              cur = next
30          # cur指向的是最后部分,中间已经没有了
31          # None 的下一个
32          # 最后面一段
33          prev.next.next = cur
```

```
34            '''
35    #       wi = temp
36    #       while wi.next :
37    #           wi = wi.next
38    #       wi.next = cur
39            '''
40            # 中间一段
41            prev.next = temp
42
43            return dummy.next
```

```
1   #
2   # @lc app=leetcode.cn id=93 lang=python3
3   #
4   # [93] 复原IP地址
5   #
6   class Solution:
7       def restoreIpAddresses( self , s: str ) −> List[str]:
8           res = []
9           self .dfs(s ,[], res ,0)
10          return res
11
12      def dfs( self ,s,ip,res,start ):
13          # 终止条件
14          if len(ip) == 4 and start == len(s):
15              address = '.'.join (ip)
16              res.append(address)
17              return
18
19          # 特殊场景下可以剪枝
20          # 剩下的子串太长(剩下的ip位都超过了3位)或太短(剩下的ip位都小于1位了)
21          if len(s) −start > 3*(4−len(ip)) or len(s) −start < (4−len(ip)):
22              return
23
24          # 最多三位(+0,+1,+2)
25          for i in range(0,3):
26              substr = s[start :start +i+1]
27              # 允许单个0,但是不允许0开头的一串,比如025
28              if i != 0 and substr[0] == '0':
29                  continue
30              if substr and int(substr) >= 0 and int(substr) <= 255:
31                  self .dfs(s,ip+[substr ],res , start + i + 1)
```

```
1   #
2   # @lc app=leetcode.cn id=94 lang=python3
3   #
```

```python
# [94] 二叉树的中序遍历
#
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    def inorderTraversal(self, root: TreeNode) -> List[int]:
        if root is None:
            return None
        result = []
        stack = []
        p = root
        while stack or p:
            # 先把左边的压进去
            if p:
                stack.append(p)
                p = p.left
            else:
                p = stack.pop()
                result.append(p.val)
                p = p.right

        return result
```

```python
#
# @lc app=leetcode.cn id=95 lang=python3
#
# [95] 不同的二叉搜索树 II
#
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    def generateTrees(self, n: int) -> List[TreeNode]:
        if n == 0:
            return []
        return self.get_trees(1,n)
```

```
19    def get_trees(self,start,end):
20        res = []
21        if start > end:
22            # 空子树情况
23            return [None]
24        for i in range(start,end+1):
25            lefts = self.get_trees(start,i-1)
26            rights = self.get_trees(i+1,end)
27            # lefts 和 rights 有可能是空的[None]
28            for l in lefts:
29                for r in rights:
30                    root = TreeNode(i)
31                    root.left = l
32                    root.right = r
33                    res.append(root)
34        return res
```

```
1   #
2   # @lc app=leetcode.cn id=96 lang=python3
3   #
4   # [96] 不同的二叉搜索树
5   #
6   class Solution:
7       def numTrees(self, n: int) -> int:
8           f = [0 for _ in range(n+1)]
9           f[0] = f[1] = 1
10          for k in range(2,n+1):
11              for i in range(k+1):
12                  f[k] += f[i-1]*f[k-i]
13          return f[n]
```

```
1   #
2   # @lc app=leetcode.cn id=97 lang=python3
3   #
4   # [97] 交错字符串
5   #
6   class Solution:
7       def isInterleave(self, s1: str, s2: str, s3: str) -> bool:
8           l1, l2, l3 = len(s1), len(s2), len(s3)
9           if l1+l2 != l3:
10              return False
11
12          dp = [[True for _ in range(l2+1)] for _ in range(l1+1)]
13          # 边界条件
14          # 用s1去填
15          for i in range(1, l1+1):
```

```
16              dp[i][0] = dp[i-1][0] and s1[i-1] == s3[i-1]
17          # 用s2去填
18          for j in range(1, l2+1):
19              dp[0][j] = dp[0][j-1] and s2[j-1] == s3[j-1]
20
21          for i in range(1, l1+1):
22              for j in range(1, l2+1):
23                  dp[i][j] = (dp[i-1][j] and s1[i-1] == s3[i+j-1]) or \
24                  (dp[i][j-1] and s2[j-1] == s3[i+j-1])
25
26          return dp[l1][l2]
```

```
1  #
2  # @lc app=leetcode.cn id=98 lang=python3
3  #
4  # [98] 验证二叉搜索树
5  #
6  # Definition for a binary tree node.
7  # class TreeNode:
8  #     def __init__(self, x):
9  #         self.val = x
10 #         self.left = None
11 #         self.right = None
12
13 class Solution:
14     def isValidBST(self, root: TreeNode) -> bool:
15         return self.isOK(root,-float('inf'),float('inf'))
16
17     def isOK(self,root,low,upper):
18         if root is None:
19             return True
20         elif root.val > low and root.val < upper:
21             return self.isOK(root.left,low,root.val) and self.isOK(root.right,root.val,upper)
22         else:
23             return False
```

```
1  #
2  # @lc app=leetcode.cn id=99 lang=python3
3  #
4  # [99] 恢复二叉搜索树
5  #
6  # Definition for a binary tree node.
7  # class TreeNode:
8  #     def __init__(self, x):
9  #         self.val = x
10 #         self.left = None
```

```python
#            self.right = None
#
class Solution:
    def recoverTree(self, root: TreeNode) -> None:
        cur, pre = root, None
        first, second = None, None
        stack = []

        while cur or stack:
            if cur:
                stack.append(cur)
                cur = cur.left
            else:
                node = stack.pop()
                if pre and pre.val >= node.val:
                    if not first:
                        first = pre
                    second = node

                pre = node
                cur = node.right

        first.val, second.val = second.val, first.val
        '''
        # 定义
        self.pre = None
        self.m1, self.m2 = None, None

        self.inorderTraversal(root)
        self.m1.val, self.m2.val = self.m2.val, self.m1.val
        '''

    # 中序遍历
    def inorderTraversal(self, root):
        if root:
            self.inorderTraversal(root.left)
            if self.pre and self.pre.val > root.val:
                if self.m1 == None:
                    self.m1 = self.pre
                self.m2 = root
            self.pre = root
            self.inorderTraversal(root.right)
```

```python
#
# @lc app=leetcode.cn id=100 lang=python3
#
```

```python
# [100] 相同的树
#
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    def isSameTree(self, p: TreeNode, q: TreeNode) -> bool:
        if p is None and q is None:
            return True
        elif p and q and p.val == q.val:
            return self.isSameTree(p.left,q.left) and self.isSameTree(p.right, q.right)
        elif p or q :
            return False
```

```python
#
# @lc app=leetcode.cn id=101 lang=python3
#
# [101] 对称二叉树
#
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    def isSymmetric(self, root: TreeNode) -> bool:
        if root is None:
            return True
        return self.yes(root.left ,root.right)

    def yes(self, left ,right):
        if left is None and right is None:
            return True
        if left and right and left.val == right.val:
            if self.yes(left.left ,right.right) and self.yes(left.right ,right.left):
                return True
        return False
```

```python
#
# @lc app=leetcode.cn id=102 lang=python3
```

```python
#
# [102] 二叉树的层次遍历
#
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    def levelOrder(self, root: TreeNode) -> List[List[int]]:
        if not root:
            return []
        result = [[]]
        self.traverse(root,0, result)
        return result

    def traverse(self, root, level, result):
        if not root:
            return
        if level >= len(result):
            result.append([])
        result[level].append(root.val)
        self.traverse(root.left, level+1,result)
        self.traverse(root.right, level+1,result)
```

```python
#
# @lc app=leetcode.cn id=103 lang=python3
#
# [103] 二叉树的锯齿形层次遍历
#
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    def zigzagLevelOrder(self, root: TreeNode) -> List[List[int]]:
        if not root:
            return []
        result = [[]]
        self.traverse(root,0, result, True)
        return result
```

```
20
21      def traverse ( self ,root, level , result , flag ):
22          if  root  is  None:
23              return
24          if  level  >= len(result):
25              result .append([])
26
27          if  flag :
28              result [ level ]. append(root.val)
29          else :
30              result [ level ]. insert (0,root. val)
31          self . traverse (root. left , level +1,result,  not  flag )
32          self . traverse (root. right , level +1,result,  not  flag )
```

```
1   #
2   # @lc app=leetcode.cn id=104 lang=python3
3   #
4   # [104] 二叉树的最大深度
5   #
6   # Definition  for  a  binary  tree  node.
7   # class  TreeNode:
8   #     def  ___init___(self, x):
9   #         self . val  = x
10  #         self . left  = None
11  #         self . right  = None
12
13  class  Solution :
14      def  maxDepth(self, root: TreeNode)  −> int :
15          if  root  is  None:
16              return  0
17          elif  root. left  and root.right :
18              return  1 + max(self.maxDepth(root.left),self.maxDepth(root.right))
19          elif  root. left :
20              return  1 + self .maxDepth(root.left)
21          elif  root. right :
22              return  1 + self .maxDepth(root.right)
23          else :
24              return  1
```

```
1   #
2   # @lc app=leetcode.cn id=105 lang=python3
3   #
4   # [105] 从前序与中序遍历序列构造二叉树
5   #
6   # Definition  for  a  binary  tree  node.
7   # class  TreeNode:
```

```python
#     def __init__(self, x):
#         self.val = x
#         self.left  = None
#         self.right  = None

class Solution:
    def buildTree(self, preorder: List[int], inorder: List[int]) -> TreeNode:
        if inorder:
            # 前序的头就是root
            # 中序中,root左边就是左子树,右边是右子树
            idx = inorder.index(preorder.pop(0))
            root = TreeNode(inorder[idx])
            # 递归构造子树先left后right
            root.left  = self.buildTree(preorder, inorder[0:idx])
            root.right  = self.buildTree(preorder, inorder[idx+1:])
            return root
        else:
            return None
```

```python
#
# @lc app=leetcode.cn id=106 lang=python3
#
# [106] 从中序与后序遍历序列构造二叉树
#
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left  = None
#         self.right  = None

class Solution:
    def buildTree(self, inorder: List[int], postorder: List[int]) -> TreeNode:
        if inorder :
            # 后序的尾部就是root
            # 中序中,root值左边就是左子树,右边是右子树
            idx = inorder.index(postorder.pop())
            root = TreeNode(inorder[idx])
            # 递归构造子树先right后left
            root.right  = self.buildTree(inorder[idx+1:],postorder)
            root.left   = self.buildTree(inorder[0:idx],postorder)
            return root
        else:
            return None
```

```python
#
```

```python
# @lc app=leetcode.cn id=107 lang=python3
#
# [107] 二叉树的层次遍历 II
#
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left  = None
#         self.right  = None

class Solution:
    def levelOrderBottom(self, root: TreeNode) -> List[List[int]]:

        if not root:
            return []
        # use stack
        stack = [[root]]
        res = []
        while stack:
            # 取出最新装入的list
            top = stack.pop()
            # 一直在头部插入以达到倒序
            res.insert(0, [t.val for t in top])
            # 向下新一轮扫描
            temp = []
            for node in top:
                if node.left  :
                    temp.append(node.left)
                if node.right  :
                    temp.append(node.right)
            if temp:
                stack.append(temp)
        return res
        '''
        # 递归法
        if not root:
            return []
        result = [[]]
        self.traverse(root,0, result)
        result.reverse()
        return result
        '''

    def traverse( self ,root, level , result ):
        if root is None:
```

```
48              return
49          if  level  >=  len(result):
50              result .append([])
51          result [ level ]. append(root.val)
52          self . traverse (root. left , level +1,result )
53          self . traverse (root. right , level +1,result )
```

```
1   #
2   # @lc app=leetcode.cn id=108 lang=python3
3   #
4   # [108] 将有序数组转换为二叉搜索树
5   #
6   # Definition for  a  binary  tree  node.
7   # class TreeNode:
8   #     def ___init___(self, x):
9   #          self . val  = x
10  #          self . left  = None
11  #          self . right  = None
12
13  class  Solution:
14      def sortedArrayToBST(self, nums: List[int])  −> TreeNode:
15          if  not  nums :
16              return  None
17          mid = len(nums)//2
18
19          root  = TreeNode(nums[mid])
20          root. left  = self .sortedArrayToBST(nums[:mid])
21          root. right  = self .sortedArrayToBST(nums[mid+1:])
22
23          return  root
```

```
1   #
2   # @lc app=leetcode.cn id=109 lang=python3
3   #
4   # [109] 有序链表转换二叉搜索树
5   #
6   # Definition for  singly−linked  list .
7   # class ListNode:
8   #     def ___init___(self, x):
9   #          self . val  = x
10  #          self . next = None
11
12  # Definition for  a  binary  tree  node.
13  # class TreeNode:
14  #     def ___init___(self, x):
15  #          self . val  = x
```

```python
16  #         self.left  = None
17  #         self.right  = None
18
19  class  Solution:
20      def sortedListToBST(self, head: ListNode) -> TreeNode:
21          '''
22          if  not head :
23              return None
24          if  not head.next:
25              return TreeNode(head.val)
26
27          slow = head
28          fast = head.next.next
29          while  fast  and fast.next:
30              fast = fast.next.next
31              slow = slow.next
32          head2 = slow.next
33          slow.next = None
34          root = TreeNode(head2.val)
35          root.left  = self.sortedListToBST(head)
36          root.right  = self.sortedListToBST(head2.next)
37          return root
38          '''
39
40          if  not head :
41              return None
42          nums = []
43          while  head:
44              nums.append(head.val)
45              head = head.next
46          return  self.sortedArrayToBST(nums)
47
48      def sortedArrayToBST(self, nums):
49          if  not nums :
50              return None
51          mid = len(nums)//2
52
53          root = TreeNode(nums[mid])
54          root.left  = self.sortedArrayToBST(nums[:mid])
55          root.right  = self.sortedArrayToBST(nums[mid+1:])
56
57          return root
```

```python
1  #
2  # @lc app=leetcode.cn id=110 lang=python3
3  #
```

```python
# [110] 平衡二叉树
#
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

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

    def check(self, root):
        if root is None:
            return 0
        l = self.check(root.left)
        r = self.check(root.right)
        if l == -1 or r == -1 or abs(l-r)>1:
            return -1
        return 1 + max(l,r)
```

```python
#
# @lc app=leetcode.cn id=111 lang=python3
#
# [111] 二叉树的最小深度
#
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    def minDepth(self, root: TreeNode) -> int:
        if root is None:
            return 0
        if root.left is None or root.right is None:
            return self.minDepth(root.left) + self.minDepth(root.right) + 1
        return min(self.minDepth(root.left), self.minDepth(root.right)) + 1
```

```python
#
# @lc app=leetcode.cn id=112 lang=python3
#
# [112] 路径总和
```

```python
#
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left  = None
#         self.right  = None

class Solution:
    def hasPathSum(self, root: TreeNode, sum: int) -> bool:
        if root is None:
            return False

        sum -= root.val
        if sum == 0 and root.left is None and root.right is None:
            return True
        left  = self.hasPathSum(root.left,sum)
        right = self.hasPathSum(root.right,sum)
        return left or right
```

```python
#
# @lc app=leetcode.cn id=113 lang=python3
#
# [113] 路径总和 II
#
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left  = None
#         self.right  = None

class Solution:
    def pathSum(self, root: TreeNode, sum: int) -> List[List[int]]:
        if root is None:
            return []
        result = []
        self.dfs(root, sum, [], result)
        return result

    def dfs(self,root,sum,path,result):
        if root is None:
            return
        if root.left is None and root.right is None and sum == root.val:
            path.append(root.val)
            result.append(path)
```

```
27
28          self.dfs(root.left, sum − root.val, path + [root.val], result)
29          self.dfs(root.right, sum − root.val, path + [root.val], result)
```

```
1   #
2   # @lc app=leetcode.cn id=114 lang=python3
3   #
4   # [114] 二叉树展开为链表
5   #
6   # Definition for a binary tree node.
7   # class TreeNode:
8   #     def __init__(self, x):
9   #         self.val = x
10  #         self.left = None
11  #         self.right = None
12
13  class Solution:
14      def flatten(self, root: TreeNode) −> None:
15          if root is None:
16              return
17
18          self.flatten(root.left)
19          self.flatten(root.right)
20
21          if root.left is None:
22              return
23
24          # 左子树插到root和root.right之间
25          p = root.left
26          # 左子链的最后一个节点
27          while p.right:
28              p = p.right
29          p.right = root.right
30          root.right = root.left
31          root.left = None
```

```
1   #
2   # @lc app=leetcode.cn id=115 lang=python3
3   #
4   # [115] 不同的子序列
5   #
6   class Solution:
7       def numDistinct(self, s: str, t: str) −> int:
8           if s is None or t is None:
9               return 0
10          ls = len(s)
```

```
11          lt = len(t)
12          dp = [ [0 for _ in range(lt+1) ] for _ in range(ls+1)]
13
14          # init
15          # 当子串长度为0时，所有次数都是1
16          # 当母串长度为0时，所有次数都是0 (默认是0,不用重复了)
17          for i in range(ls+1):
18              dp[i][0] = 1
19
20          for i in range(1,ls+1):
21              for j in range(1,lt+1):
22                  # 跳过上一个字符串匹配过程
23                  dp[i][j] = dp[i-1][j]
24                  # 要匹配的话
25                  if s[i-1] == t[j-1]:
26                      dp[i][j] += dp[i - 1][j - 1]
27
28          return dp[-1][-1]
```

```
1  #
2  # @lc app=leetcode.cn id=116 lang=python3
3  #
4  # [116] 填充每个节点的下一个右侧节点指针
5  #
6  """
7  # Definition for a Node.
8  class Node:
9      def __init__(self, val: int = 0, left : 'Node' = None, right: 'Node' = None, next: 'Node' =
           None):
10         self.val = val
11         self.left = left
12         self.right = right
13         self.next = next
14  """
15  class Solution:
16      def connect(self, root: 'Node') -> 'Node':
17          if root is None or root.left is None:
18              return root
19          # 左右链接
20          root.left.next = root.right
21          if root.next :
22              root.right.next = root.next.left
23          else:
24              root.right.next = None
25
26          self.connect(root.left )
```

```python
27          self.connect(root.right)
28
29          return root
```

```python
1  #
2  # @lc app=leetcode.cn id=117 lang=python3
3  #
4  # [117] 填充每个节点的下一个右侧节点指针 II
5  #
6  """
7  # Definition for a Node.
8  class Node:
9      def __init__(self, val: int = 0, left: 'Node' = None, right: 'Node' = None, next: 'Node' =
          None):
10         self.val = val
11         self.left = left
12         self.right = right
13         self.next = next
14 """
15 class Solution:
16     def connect(self, root: 'Node') -> 'Node':
17         head = root
18         dummy = Node(-1)
19         prev = dummy
20         # dummy 当前行的最左端节点
21         while root:
22             if root.left:
23                 prev.next = root.left
24                 prev = prev.next
25             if root.right:
26                 prev.next = root.right
27                 prev = prev.next
28             root = root.next
29             # 行的尾部
30             if root is None:
31                 # dummy.next为前面prev.next 第一次赋值的节点
32                 root = dummy.next
33                 #前面链接断开,开始新的一行
34                 dummy.next = None
35                 # prev值新的
36                 prev = dummy
37         return head
```

```python
1  #
2  # @lc app=leetcode.cn id=118 lang=python3
3  #
```

```python
# [118] 杨辉三角
#
class Solution:
    def generate(self, numRows: int) -> List[List[int]]:
        # 全部都用1先填充
        out = [[1]*(i+1) for i in range(numRows)]
        for r in range(numRows):
            for col in range(1,r):
                out[r][col] = out[r-1][col-1] + out[r-1][col]
        return out
```

```python
#
# @lc app=leetcode.cn id=119 lang=python3
#
# [119] 杨辉三角 II
#
class Solution:
    def getRow(self, rowIndex: int) -> List[int]:
        '''
        if rowIndex == 0:
            return [1]
        rowIndex += 1
        # 全部都用1先填充
        out = [[1]*(i+1) for i in range(rowIndex)]
        for r in range(rowIndex):
            for col in range(1,r):
                out[r][col] = out[r-1][col-1] + out[r-1][col]
        return out[-1]
        '''
        # 先用1填充
        res = [1]*(rowIndex+1)
        # 从后往前,从上往下覆盖
        for r in range(2,rowIndex+1):
            for col in range(r-1,0,-1):# 逆序
                res[col] += res[col-1]
        return res
```

```python
#
# @lc app=leetcode.cn id=120 lang=python3
#
# [120] 三角形最小路径和
#
class Solution:
    def minimumTotal(self, triangle: List[List[int]]) -> int:
        if not triangle:
            return
```

```
10            # 倒数第二行到最上面一行
11            for i in range( len( triangle )−2, −1, −1):
12                # 每行的第一列到最后一列
13                for j in range(len( triangle [i])):
14                    triangle [i][j] += min(triangle[i+1][j], triangle [i+1][j+1])
15            return triangle [0][0]
```

```
1   #
2   # @lc app=leetcode.cn id=121 lang=python3
3   #
4   # [121] 买卖股票的最佳时机
5   #
6   class Solution:
7       def maxProfit(self, prices: List [int]) −> int:
8           if not prices:
9               return 0
10          minelement = float('inf')
11          profit = 0
12          for i in range(len( prices )):
13              minelement = min(minelement, prices[i])
14              profit = max(profit, prices [i] − minelement)
15          return profit
```

```
1   #
2   # @lc app=leetcode.cn id=122 lang=python3
3   #
4   # [122] 买卖股票的最佳时机 II
5   #
6   class Solution:
7       def maxProfit(self, prices: List [int]) −> int:
8           if not prices:
9               return 0
10          profit = 0
11          for i in range(1,len( prices )):
12              if prices [i]>prices [i−1]:
13                  profit += (prices[i]−prices [i−1])
14          return profit
```

```
1   #
2   # @lc app=leetcode.cn id=123 lang=python3
3   #
4   # [123] 买卖股票的最佳时机 III
5   #
6   class Solution:
7       def maxProfit(self, prices: List [int]) −> int:
8           '''
9           """
```

```
10            对于任意一天考虑四个变量:
11    #   fstBuy: 在该天第一次买入股票可获得的最大收益
12    #    fstSell: 在该天第一次卖出股票可获得的最大收益
13    #   secBuy: 在该天第二次买入股票可获得的最大收益
14    #    secSell: 在该天第二次卖出股票可获得的最大收益
15        分别对四个变量进行相应的更新, 最后secSell就是最大
16        收益值(secSell >= fstSell)
17        """
18        fstBuy, fstSell = -float('inf'),0
19        secBuy,secSell= -float('inf'),0
20        for i in prices:
21            fstBuy = max(fstBuy, -i)
22             fstSell  = max(fstSell, fstBuy + i)
23            secBuy = max(secBuy, fstSell - i)
24             secSell  = max(secSell, secBuy + i)
25        return secSell
26        '''
27        if not prices:
28            return 0
29        num = len(prices)
30
31        forward  = [0]*num
32        backward = [0]*num
33        # 前向
34        current_min = prices[0]
35        for i in range(1,len(prices)):
36            current_min = min(current_min, prices[i])
37            forward[i]  = max(forward[i-1] , prices[i]-current_min )
38        # 后向
39        total_max = 0
40        current_max = prices[-1]
41        for i in range(len(prices) - 2, -1, -1):
42            current_max = max(current_max, prices[i])
43            backward[i]  = max(backward[i+1], current_max - prices[i])
44            total_max = max(total_max, backward[i] + forward[i])
45        return total_max
```

```
1  #
2  # @lc app=leetcode.cn id=124 lang=python3
3  #
4  # [124] 二叉树中的最大路径和
5  #
6  # Definition for a binary tree node.
7  # class TreeNode:
8  #     def __init__(self, x):
9  #         self.val = x
```

```python
10  #         self.left  = None
11  #         self.right = None
12
13  class Solution:
14      def maxPathSum(self, root: TreeNode) -> int:
15          self.res = -float('inf')
16          self.maxend(root)
17          return self.res
18
19      def maxend(self,root):
20          if root is None:
21              return 0
22          left = self.maxend(root.left)
23          right = self.maxend(root.right)
24          self.res = max(self.res, left + root.val + right)
25          return max(root.val + max(left, right), 0)
```

```python
1  #
2  # @lc app=leetcode.cn id=125 lang=python3
3  #
4  # [125] 验证回文串
5  #
6  class Solution:
7      def isPalindrome(self, s: str) -> bool:
8          # 检测字符串是否由字母和数字组成
9          alnum = [t.lower() for t in s if t.isalnum()]
10         leng = len(alnum)
11         mid = leng//2
12         if leng < 2:
13             return True
14         for i in range(mid):
15             if alnum[i] != alnum[leng - i -1]:
16                 return False
17         return True
```

```python
1  #
2  # @lc app=leetcode.cn id=126 lang=python3
3  #
4  # [126] 单词接龙 II
5  #
6  class Solution:
7      def findLadders(self, beginWord: str, endWord: str, wordList: List[str]) -> List[List[str]]:
8          import collections
9          wordset = set(wordList)
10
11         level = {beginWord}
```

```
12          parents = collections.defaultdict(set)
13
14          while level and endWord not in parents:
15              next_level = collections.defaultdict(set)
16              for word in level:
17                  # 不同位置都可以插入不同字母进行新单词重构
18                  for i in range(len(beginWord)):
19                      for c in 'abcdefghijklmnopqrstuvwxyz':
20                          newWord = word[:i] + c + word[i+1:]
21                          if newWord in wordset and newWord not in parents:
22                              next_level[newWord].add(word)
23
24              level = next_level
25              parents.update(next_level)
26          res = [[endWord]]
27          # parents相当于是逆向
28          while res and res[0][0] != beginWord:
29              # 确定是等长的
30              res = [[p]+r for r in res for p in parents[r[0]]]
31          return res
```

```
1   #
2   # @lc app=leetcode.cn id=127 lang=python3
3   #
4   # [127] 单词接龙
5   #
6   class Solution:
7       def ladderLength(self, beginWord: str, endWord: str, wordList: List[str]) -> int:
8           # 防止时间超出
9           wordset = set(wordList)
10          # 初始化
11          bfs = [(beginWord, 1)]
12          while bfs:
13              word,length = bfs.pop(0) # 左边弹出
14              if word == endWord:
15                  return length
16              for i in range(len(word)):
17                  for c in "abcdefghijklmnopqrstuvwxyz":
18                      # 不同位置都可以插入不同字母进行新单词重构
19                      newWord = word[:i] + c + word[i + 1:]
20                      if newWord in wordset and newWord != word:
21                          wordset.remove(newWord)
22                          bfs.append((newWord, length + 1))
23          return 0
```

```
1   #
```

```python
# @lc app=leetcode.cn id=128 lang=python3
#
# [128] 最长连续序列
#
class Solution:
    def longestConsecutive(self, nums: List[int]) -> int:
        maxLen = 0
        while nums:
            n = nums.pop()
            # 往大处搜索
            i1 = n + 1
            while i1 in nums:
                nums.remove(i1)
                i1 += 1
            # 往小处搜索
            i2 = n - 1
            while i2 in nums:
                nums.remove(i2)
                i2 -= 1
            maxLen = max(maxLen, i1 - i2 - 1)
        return maxLen
```

```python
#
# @lc app=leetcode.cn id=129 lang=python3
#
# [129] 求根到叶子节点数字之和
#
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    def sumNumbers(self, root: TreeNode) -> int:
        return self.sum_tree(root,0)

    def sum_tree(self,root,sum):
        if root is None:
            return 0
        if root.left is None and root.right is None:
            return sum*10+root.val

        return self.sum_tree(root.left,sum*10+root.val)+ self.sum_tree(root.right,sum*10+root.val)
```

```python
#
# @lc app=leetcode.cn id=130 lang=python3
#
# [130] 被围绕的区域
#
class Solution:
    def solve(self, board: List[List[str]]) -> None:
        if len(board) <= 2 or len(board[0])<=2:
            return
        row, col = len(board), len(board[0])
        # 对边界上的所有点分别进行深度遍历
        # 第一列和最后一列
        for i in range(row):
            self.dfs(board,i,0,    row,col)
            self.dfs(board,i,col-1,row,col)
        # 第一行和最后一行
        for j in range(1,col-1):
            self.dfs(board,0,    j,row,col)
            self.dfs(board,row-1,j,row,col)

        for i in range(row):
            for j in range(col):
                if board[i][j] == "O":
                    board[i][j] = "X"
                if board[i][j] == "T":
                    board[i][j] = "O"
        return

    def dfs(self,board,i,j,row,col):
        if i < 0 or j < 0 or i >= row or j >= col or board[i][j] != "O":
            return
        else:
            board[i][j] = "T"
            self.dfs(board,i-1,j,row,col)
            self.dfs(board,i,j-1,row,col)
            self.dfs(board,i+1,j,row,col)
            self.dfs(board,i,j+1,row,col)
        return
```

```python
#
# @lc app=leetcode.cn id=131 lang=python3
#
# [131] 分割回文串
#
class Solution:
    def partition(self, s: str) -> List[List[str]]:
```

```
 8          res = []
 9          self.dfs(s,res,  [],   0)
10          return res
11
12      def dfs(self,s, res, path, start):
13          if start == len(s):
14              res.append(path)
15              return
16          for i in range(start, len(s)):
17              if self.isPalindrome(s, start, i):
18                  self.dfs(s,res,path +[s[start:i+1]],i + 1)
19      # 判断回文
20      def isPalindrome(self, s, begin, end):
21          while begin < end :
22              if s[begin] != s[end]:
23                  return False
24              begin += 1
25              end -= 1
26          return True
```

```
 1  #
 2  # @lc app=leetcode.cn id=132 lang=python3
 3  #
 4  # [132] 分割回文串 II
 5  #
 6  class Solution:
 7      def minCut(self, s: str) -> int:
 8          n = len(s)
 9          dp = [[False for _ in range(n)] for _ in range(n)]
10          # f[0->n](共n+1个) f[n]=-1
11          # f(i) [i, n-1]最小裁剪数
12          f = [n] *(n+1)
13          f[-1] = -1
14          # f 从右往左更新
15          # dp (i 往左更新,j往右更新)
16          for i in range(n-1,-1,-1):
17              for j in range(i,n):
18                  if (s[i] == s[j] and (j - i < 2 or dp[i + 1][j - 1])) :
19                      dp[i][j] = True
20                      # 如果满足回文的条件
21                      # f 选取裁剪更少的方案
22                      f[i] = min(f[i], f[j + 1] + 1)
23          return f[0]
```

```
 1  #
 2  # @lc app=leetcode.cn id=133 lang=python3
```

```python
#
# [133] 克隆图
#
"""
# Definition for a Node.
class Node:
    def __init__(self, val = 0, neighbors = []):
        self.val = val
        self.neighbors = neighbors
"""
class Solution:
    def cloneGraph(self, node: 'Node') -> 'Node':
        if not node:
            return None
        '''
        # BFS
        queue = [node]
        copy_node = Node(node.val)
        visited = {node: copy_node}
        while queue:
            node = queue.pop(0)
            for i in node.neighbors:
                if i in visited:
                    visited[node].neighbors.append(visited[i])
                else:
                    copy_node_ne = Node(i.val)
                    visited[node].neighbors.append(copy_node_ne)
                    visited[i] = copy_node_ne
                    queue.append(i)

        return copy_node
        '''
        # DFS
        stack = [node]
        copy_node = Node(node.val)
        visited = {node: copy_node}
        while stack:
            node = stack.pop()
            for i in node.neighbors:
                if i in visited:
                    visited[node].neighbors.append(visited[i])
                else:
                    copy_node_ne = Node(i.val)
                    visited[node].neighbors.append(copy_node_ne)
                    visited[i] = copy_node_ne
                    stack.append(i)
```

```
49
50            return copy_node
```

```
1    #
2    # @lc app=leetcode.cn id=134 lang=python3
3    #
4    # [134] 加油站
5    #
6    class Solution:
7        def canCompleteCircuit(self, gas: List[int], cost: List[int]) -> int:
8            sumGas = sumCost = 0
9            start = 0
10           diff = 0
11           for i in range(len(gas)):
12               sumGas += gas[i]
13               sumCost += cost[i]
14               diff += gas[i] - cost[i]
15               if diff < 0:
16                   start = i + 1 ## 下一个开始
17                   diff = 0
18           return start if sumGas - sumCost >= 0 else -1
```

```
1    #
2    # @lc app=leetcode.cn id=135 lang=python3
3    #
4    # [135] 分发糖果
5    #
6    class Solution:
7        def candy(self, ratings: List[int]) -> int:
8            if not ratings:
9                return 0
10           leng = len(ratings)
11           res = [1 for _ in range(leng)]
12           for i in range(1,leng):
13               # 右边大
14               if ratings[i] > ratings[i-1]:
15                   res[i] = res[i-1] + 1
16           for i in range(leng-1 , 0 , -1 ):
17               # 左边大
18               if ratings[i-1] > ratings[i]:
19                   res[i-1] = max(res[i]+1 ,res[i-1])
20           return sum(res)
```

```
1    #
2    # @lc app=leetcode.cn id=136 lang=python3
3    #
4    # [136] 只出现一次的数字
```

```python
#
class Solution:
    def singleNumber(self, nums: List[int]) -> int:
        '''
        return 2*sum(set(nums)) - sum(nums)
        '''
        res = 0
        for i in range(len(nums)):
            res = res ^ nums[i]
        return res
```

```python
#
# @lc app=leetcode.cn id=137 lang=python3
#
# [137] 只出现一次的数字 II
#
class Solution:
    def singleNumber(self, nums: List[int]) -> int:
        return (3 * sum(set(nums)) - sum(nums) ) //2
```

```python
#
# @lc app=leetcode.cn id=138 lang=python3
#
# [138] 复制带随机指针的链表
#
"""
# Definition for a Node.
class Node:
    def __init__(self, x: int, next: 'Node' = None, random: 'Node' = None):
        self.val = int(x)
        self.next = next
        self.random = random
"""
class Solution:
    def copyRandomList(self, head: 'Node') -> 'Node':
        if head is None:
            return None
        # 复制next部分
        headcopy = head
        while headcopy:
            node = Node(headcopy.val)
            node.next = headcopy.next
            headcopy.next = node
            headcopy = node.next
        # 复制random部分
        headcopy = head
```

```
27        while headcopy:
28            if headcopy.random:
29                headcopy.next.random = headcopy.random.next
30            headcopy = headcopy.next.next
31
32        # 拆分两个单链表
33        src = head
34        pnew = res = head.next
35
36        while pnew.next:
37            src.next = pnew.next
38            src = src.next
39            pnew.next = src.next
40            pnew = pnew.next
41        src.next = None
42        pnew.next = None
43
44        return res
```

```
1  #
2  # @lc app=leetcode.cn id=139 lang=python3
3  #
4  # [139] 单词拆分
5  #
6  class Solution:
7      def wordBreak(self, s: str, wordDict: List[str]) -> bool:
8          n = len(s)
9          dp = [False for _ in range(n+1)]
10         dp[0] = True
11
12         for i in range(n+1):
13             for j in range(i-1,-1,-1):
14                 if dp[j] and s[j:i] in wordDict:
15                     dp[i] = True
16                     break
17
18         return dp[-1]
```

```
1  #
2  # @lc app=leetcode.cn id=140 lang=python3
3  #
4  # [140] 单词拆分 II
5  #
6  class Solution:
7      def wordBreak(self, s: str, wordDict: List[str]) -> List[str]:
8          n = len(s)
```

```python
        dp = [False for _ in range(n+1)]
        dp[0] = True
        # prev true 表示s[j,i)是一个合法单词,从j处切开
        prev = [[False for _ in range(n)] for _ in range(n+1) ]

        for i in range(n+1):
            for j in range(i-1,-1,-1):
                if dp[j] and s[j:i] in wordDict:
                    dp[i] = True
                    prev[i][j] = True

        res = []
        self.dfs(s,prev,n ,[], res)
        return res

    def dfs(self,s,prev,cur,path,res):
        if cur == 0:
            # 终止条件
            temp = " ".join(list(reversed(path)))
            res.append(temp)
            return

        for i in range(cur-1,-1,-1):
            if prev[cur][i]:
                self.dfs(s,prev,i,path+[s[i:cur]], res)
```

```python
#
# @lc app=leetcode.cn id=141 lang=python3
#
# [141] 环形链表
#
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution:
    def hasCycle(self, head: ListNode) -> bool:
        '''
        try:
            slow = head
            fast = head.next
            while slow is not fast:
                slow = slow.next
                fast = fast.next.next
```

```
21            return True
22        except:
23            return False
24        '''
25        fast = slow = head
26        while fast and fast.next:
27            fast = fast.next.next
28            slow = slow.next
29            if slow == fast:
30                return True
31        return False
```

```
1  #
2  # @lc app=leetcode.cn id=142 lang=python3
3  #
4  # [142] 环形链表 II
5  #
6  # Definition for singly-linked list.
7  # class ListNode:
8  #     def __init__(self, x):
9  #         self.val = x
10 #         self.next = None
11
12 class Solution:
13     def detectCycle(self, head: ListNode) -> ListNode:
14         fast = slow = head
15         while fast and fast.next:
16             slow = slow.next
17             fast = fast.next.next
18             if slow == fast:
19                 #相遇了
20                 res = head
21                 while res != slow:
22                     slow = slow.next
23                     res = res.next
24                 return res
25         return None
```

```
1  #
2  # @lc app=leetcode.cn id=143 lang=python3
3  #
4  # [143] 重排链表
5  #
6  # Definition for singly-linked list.
7  # class ListNode:
8  #     def __init__(self, x):
```

```python
 9  #        self.val = x
10  #        self.next = None
11
12  class Solution:
13      def reorderList(self, head: ListNode) -> None:
14          if head is None or head.next is None:
15              return head
16          p1 ,p2 = head , head
17          while p2 and p2.next :
18              p1 = p1.next
19              p2 = p2.next.next
20          # head2 是后面半部分
21          head2 = p1.next
22          p1.next = None
23          # head head2 对应前后两部分
24
25          cur = head2
26          rever = None
27          # 反转
28          while cur:
29              temp = cur.next
30              cur.next = rever
31              rever = cur
32              cur = temp
33
34          # head rever 两个合并
35          p1 = head
36          while rever:
37              # 两个链的下一个
38              temp = p1.next
39              temp2 = rever.next
40              # 链接好
41              p1.next = rever
42              rever.next = temp
43              # 下一个循环
44              p1 = temp
45              rever = temp2
46          return head
```

```python
1  #
2  # @lc app=leetcode.cn id=144 lang=python3
3  #
4  # [144] 二叉树的前序遍历
5  #
6  # Definition for a binary tree node.
7  # class TreeNode:
```

```python
 8  #      def __init__(self, x):
 9  #          self.val = x
10  #          self.left  = None
11  #          self.right  = None
12
13  class Solution:
14      def preorderTraversal(self, root: TreeNode) -> List[int]:
15          if root is None:
16              return []
17          result = []
18          stack = []
19          stack.append(root)
20
21          while stack:
22              p = stack.pop()
23              result.append(p.val)
24              if p.right:
25                  stack.append(p.right)
26              if p.left:
27                  stack.append(p.left)
28          return result
```

```python
 1  #
 2  # @lc app=leetcode.cn id=145 lang=python3
 3  #
 4  # [145] 二叉树的后序遍历
 5  #
 6  # Definition for a binary tree node.
 7  # class TreeNode:
 8  #      def __init__(self, x):
 9  #          self.val = x
10  #          self.left  = None
11  #          self.right  = None
12
13  class Solution:
14      def postorderTraversal(self, root: TreeNode) -> List[int]:
15          if root is None:
16              return []
17          result = []
18          stack = []
19          stack.append(root)
20          while stack:
21              p = stack.pop()
22              result.append(p.val)
23              if p.left:
24                  stack.append(p.left)
```

```python
25             if p.right:
26                 stack.append(p.right)
27         return result[::-1]
```

```python
1  #
2  # @lc app=leetcode.cn id=146 lang=python3
3  #
4  # [146] LRU缓存机制
5  #
6  class LRUCache:
7      def __init__(self, capacity: int):
8          self.capacity = capacity
9          self.cache = {}
10         self.queue = []
11
12     def update(self,key):
13         # 移到头部去
14         self.queue.remove(key)
15         self.queue.insert(0,key)
16
17     def get(self, key: int) -> int:
18         if key in self.cache:
19             self.update(key)
20             return self.cache[key]
21         else:
22             return -1
23
24     def put(self, key: int, value: int) -> None:
25         if not key or not value:
26             return None
27         if key in self.cache: # 已经在了
28             self.queue.remove(key)
29         elif len(self.queue) == self.capacity: # 满了
30             del self.cache[self.queue.pop()]
31
32         self.cache[key] = value
33         self.queue.insert(0,key)
34
35 # Your LRUCache object will be instantiated and called as such:
36 # obj = LRUCache(capacity)
37 # param_1 = obj.get(key)
38 # obj.put(key,value)
```

```python
1  #
2  # @lc app=leetcode.cn id=147 lang=python3
3  #
```

```python
# [147] 对链表进行插入排序
#
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution:
    def insertionSortList(self, head: ListNode) -> ListNode:
        dummy = ListNode(-1000)
        dummy.next = head
        p = dummy
        cur = head
        while cur and cur.next:
            val = cur.next.val
            # 顺序的
            if cur.val < val:
                cur = cur.next
                continue
            # 找到p(小于的最后一个节点)
            # 这个相当于p重新初始化
            if p.next.val > val:
                p = dummy
            while p.next.val < val:
                p = p.next
            # 右边的节点插入到左边去
            next_step = cur.next
            cur.next = cur.next.next
            next_step.next = p.next
            p.next = next_step
        return dummy.next
```

```python
#
# @lc app=leetcode.cn id=148 lang=python3
#
# [148] 排序链表
#
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution:
    def sortList(self, head: ListNode) -> ListNode:
```

```
14            if  head is  None or head.next is  None:
15                return  head
16         fast  = slow = head
17         pre  = None
18         while  fast  and fast.next:
19             fast  = fast.next.next
20             pre  = slow
21             slow = slow.next
22         pre.next = None
23         return  self.mergeTwoLists(self.sortList(head), self.sortList(slow))
24
25     def  mergeTwoLists(self, l1, l2):
26         res  = now = ListNode(−1000)
27         while  l1  and l2:
28             if  l1.val  <= l2.val:
29                 now.next = l1
30                 l1  = l1.next
31             else:
32                 now.next = l2
33                 l2  = l2.next
34             now = now.next
35         now.next = l1 or l2
36         return  res.next
```

```
1   #
2   # @lc app=leetcode.cn id=149 lang=python3
3   #
4   # [149] 直线上最多的点数
5   #
6   class  Solution:
7       def  maxPoints(self, points: List[List[int]]) −> int:
8           if  points  is  None :
9               return  0
10          res  = 0
11          # 两重循环
12          # 双重字典
13          for  i  in  range(len(points)):
14              line_map = {}
15              same = max_point_num = 0
16              for  j  in  range(i + 1, len(points)):
17                  dx, dy = points[j][0] − points[i][0], points[j][1] − points[i][1]
18                  # 同一个点
19                  if  dx == 0 and dy == 0:
20                      same += 1
21                      continue
22                  # 去除最大公约数部分
```

```
23                    gcd = self.generateGCD(dx, dy)
24                    if gcd != 0:
25                        dx //= gcd
26                        dy //= gcd
27
28                    if dx in line_map:
29                        if dy in line_map[dx]:
30                            line_map[dx][dy] += 1
31                        else:
32                            line_map[dx][dy] = 1
33                    else:
34                        line_map[dx] = {}
35                        line_map[dx][dy] = 1
36                    max_point_num = max(max_point_num, line_map[dx][dy])
37                res = max(res, max_point_num + same + 1)
38            return res
39
40        # 辗转相除法求最大公约数
41        def generateGCD(self, x, y):
42            if y == 0:
43                return x
44            else:
45                return self.generateGCD(y, x % y)
```

```
1   #
2   # @lc app=leetcode.cn id=150 lang=python3
3   #
4   # [150] 逆波兰表达式求值
5   #
6   class Solution:
7       def evalRPN(self, tokens: List[str]) -> int:
8           nums = []
9           for t in tokens:
10              if t not in ['+','-','*','/']:
11                  nums.append(int(t))
12              else:
13                  r = nums.pop()
14                  l = nums.pop()
15                  if t == '+':
16                      temp = l+r
17                  elif t == '-':
18                      temp = l-r
19                  elif t == '*':
20                      temp = l*r
21                  elif t == '/':
22                      if l*r < 0 and l%r != 0:
```

```
23                          temp = l//r + 1
24                      else:
25                          temp = l//r
26                  nums.append(temp)
27          return nums.pop()
```

```
1   #
2   # @lc app=leetcode.cn id=151 lang=python3
3   #
4   # [151] 翻转字符串里的单词
5   #
6   class Solution:
7       def reverseWords(self, s: str) -> str:
8           if not s:
9               return s
10
11          '''
12          temp = s.split(' ')
13          temp = [t for t in temp if len(t) > 0]
14          temp.reverse()
15          return ' '.join(temp)
16          '''
17          s = s + "␣"
18          l = 0
19          res = []
20          for i in range(l,len(s)):
21              if s[i] == "␣":
22                  if l != i:
23                      res.append(s[l:i])
24                  l = i + 1
25
26          res.reverse()
27          return "␣".join(res)
```

```
1   #
2   # @lc app=leetcode.cn id=152 lang=python3
3   #
4   # [152] 乘积最大子序列
5   #
6   class Solution:
7       def maxProduct(self, nums: List[int]) -> int:
8           if not nums:
9               return 0
10          maxtmp = mintmp = res = nums[0]
11          for i in range(1,len(nums)):
12              maxtmp ,mintmp =max(nums[i] , nums[i]*maxtmp ,nums[i]*mintmp) ,\
```

```
13                        min(nums[i] , nums[i]*maxtmp ,nums[i]*mintmp)
14                res = max(maxtmp,res)
15            return res
```

```
1   #
2   # @lc app=leetcode.cn id=153 lang=python3
3   #
4   # [153] 寻找旋转排序数组中的最小值
5   #
6   class Solution:
7       def findMin(self, nums: List[int]) -> int:
8           if len(nums) == 1 or nums[0] < nums[-1]: # 升序
9               return nums[0]
10          l , r = 0, len(nums)-1
11          while l < r :
12              mid = (l+r)//2
13              # 左边
14              if nums[0] <= nums[mid]:
15                  l = mid + 1
16              # 在右边
17              else :
18                  r = mid
19          return nums[l]
```

```
1   #
2   # @lc app=leetcode.cn id=154 lang=python3
3   #
4   # [154] 寻找旋转排序数组中的最小值 II
5   #
6
7   class Solution:
8       def findMin(self, nums: List[int]) -> int:
9           if len(nums) == 1 or nums[0] < nums[-1]: # 升序
10              return nums[0]
11
12          l , r = 0, len(nums)-1
13          while l < r :
14              mid = (l+r)//2
15              # 左边
16              if nums[mid] > nums[r]:
17                  l = mid + 1
18              # 在右边
19              elif nums[mid] < nums[r]:
20                  r = mid
21              # nums[mid] == nums[r]情况
22              else :
```

```
23              r −= 1
24          return nums[l]
```

```
1  #
2  # @lc app=leetcode.cn id=155 lang=python3
3  #
4  # [155] 最小栈
5  #
6  class MinStack:
7      def __init__(self):
8          self .stack = []
9          self .min_stack = []
10
11     def push(self, x: int) −> None:
12         self .stack.append(x)
13         if len( self .min_stack) == 0:
14             self .min_stack.append(x)
15             return
16         # x 和栈尾 哪个小压哪个
17         if x <= self.min_stack[−1]:
18             self .min_stack.append(x)
19         else :
20             self .min_stack.append(self.min_stack[−1])
21
22     def pop(self) −> None:
23         if len( self .stack)>0:
24             self .min_stack.pop()
25             self .stack.pop()
26
27     def top( self ) −> int:
28         if len( self .stack)>0:
29             return self .stack[−1]
30         return None
31
32     def getMin(self) −> int:
33         if len( self .min_stack)>0:
34             return self .min_stack[−1]
35         return None
36
37 # Your MinStack object will be instantiated and called as such:
38 # obj = MinStack()
39 # obj.push(x)
40 # obj.pop()
41 # param_3 = obj.top()
42 # param_4 = obj.getMin()
```

```python
#
# @lc app=leetcode.cn id=160 lang=python3
#
# [160] 相交链表
#
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution:
    def getIntersectionNode(self, headA: ListNode, headB: ListNode) -> ListNode:
        p1, p2 = headA, headB
        # 初始化两个运动结点p1和p2
        while p1 != p2:
            # 只要两个结点还未相遇
            p1 = headB if p1 is None else p1.next
            # 如果p1走到了链表A的末尾，则换到链表B上
            p2 = headA if p2 is None else p2.next
            # 如果p2走到了链表B的末尾，则换到链表A上

        return p1
        # 当p1和p2都换到对方的链表上，再次相遇后第一个结点即为首个公共结点，否则为None
```

```python
#
# @lc app=leetcode.cn id=162 lang=python3
#
# [162] 寻找峰值
#

class Solution:
    def findPeakElement(self, nums: List[int]) -> int:
        n = len(nums)
        if n == 1:
            return 0

        l, r = 0, len(nums) - 1
        while l <= r:
            mid = (l+r) // 2
            if (mid == 0 or nums[mid] > nums[mid-1]) and (mid == n - 1 or nums[mid] > nums[
                mid+1]):
                return mid
            elif mid > 0 and nums[mid-1] > nums[mid]:
                r = mid - 1
            else:
```

```
21          l = mid + 1
```

```
1  #
2  # @lc app=leetcode.cn id=165 lang=python3
3  #
4  # [165] 比较版本号
5  #
6  class Solution:
7      def compareVersion(self, version1: str, version2: str) -> int:
8          vs1 = version1.split('.')
9          vs2 = version2.split('.')
10         l1, l2 = len(vs1), len(vs2)
11         if (l1 > l2):
12             vs2 += [0]*(l1-l2)
13         elif l1 < l2:
14             vs1 += [0]*(l2-l1)
15         n = max(l1,l2)
16         for i in range(n):
17             if int(vs1[i]) > int(vs2[i]):
18                 return 1
19             elif int(vs1[i]) < int(vs2[i]):
20                 return -1
21         return 0
```

```
1  #
2  # @lc app=leetcode.cn id=167 lang=python3
3  #
4  # [167] 两数之和 II - 输入有序数组
5  #
6  class Solution:
7      def twoSum(self, numbers: List[int], target: int) -> List[int]:
8          l = 0
9          r = len(numbers)-1
10         while l <= r:
11             temp = numbers[l]+numbers[r]
12             if temp == target:
13                 return [l+1, r+1]
14             elif temp < target:
15                 l += 1
16             elif temp > target:
17                 r -= 1
```

```
1  #
2  # @lc app=leetcode.cn id=168 lang=python3
3  #
4  # [168] Excel表列名称
5  #
```

```python
class Solution:
    def convertToTitle(self, n: int) -> str:
        capitals = [chr(x) for x in range(ord('A'), ord('Z')+1)]
        result = []

        while n > 0:
            result.append(capitals[(n-1)%26])
            n = (n-1) // 26
        result.reverse()
        return ''.join(result)
```

```python
#
# @lc app=leetcode.cn id=169 lang=python3
#
# [169] 多数元素
#
class Solution:
    def majorityElement(self, nums: List[int]) -> int:
        nums.sort()
        return nums[len(nums)//2]
```

```python
#
# @lc app=leetcode.cn id=171 lang=python3
#
# [171] Excel表列序号
#
class Solution:
    def titleToNumber(self, s: str) -> int:
        res = 0
        for i in s:
            res = res*26 + ord(i)-ord('A')+1
        return res
```

```python
#
# @lc app=leetcode.cn id=172 lang=python3
#
# [172] 阶乘后的零
#
class Solution:
    def trailingZeroes(self, n: int) -> int:
        count = 0
        while n >0:
            n //= 5
            count += n
        return count
```

```python
#
# @lc app=leetcode.cn id=173 lang=python3
#
# [173] 二叉搜索树迭代器
#

# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class BSTIterator:
    def __init__(self, root: TreeNode):
        # 包含按排序顺序的所有节点的数组
        self.nodes_sorted = []
        self.index = -1
        self._inorder(root)

    def _inorder(self, root):
        if not root:
            return
        self._inorder(root.left)
        self.nodes_sorted.append(root.val)
        self._inorder(root.right)

    def next(self) -> int:
        """
        @return the next smallest number
        """
        self.index += 1
        return self.nodes_sorted[self.index]

    def hasNext(self) -> bool:
        """
        @return whether we have a next smallest number
        """
        return self.index + 1 < len(self.nodes_sorted)

# Your BSTIterator object will be instantiated and called as such:
# obj = BSTIterator(root)
# param_1 = obj.next()
# param_2 = obj.hasNext()
```

```python
#
```

```python
# @lc app=leetcode.cn id=174 lang=python3
#
# [174] 地下城游戏
#
class Solution:
    def calculateMinimumHP(self, dungeon: List[List[int]]) -> int:
        m,n = len(dungeon),len(dungeon[0])
        res = [[0 for _ in range(n)] for _ in range(m)]

        # 逆序遍历
        # 逆序初始化
        res[m-1][n-1] = max(-dungeon[m-1][n-1],0)+1
        for r in range(m-2,-1,-1):
            res[r][n-1] = max(res[r+1][n-1] -dungeon[r][n-1] ,1)
        for c in range(n-2,-1,-1):
            res[m-1][c] = max(res[m-1][c+1] -dungeon[m-1][c] ,1)
        # 从下往上从右往左遍历
        for r in range(m-2,-1,-1):
            for c in range(n-2,-1,-1):
                res[r][c] = max(
                    min(res[r][c+1],res[r+1][c]) - dungeon[r][c],
                    1)
        return res[0][0]
```

```python
#
# @lc app=leetcode.cn id=179 lang=python3
#
# [179] 最大数
#

# Python的富比较方法包括__lt__、__gt__分别表示:小于、大于，对应的操作运算符为: "<
    " 、 ">"
class LargerNumKey(str):
    def __lt__(x, y):
        return x+y < y+x

class Solution:
    def largestNumber(self, nums: List[int]) -> str:
        if set(nums) == {0}:
            return '0'
        str_nums = sorted([str(i) for i in nums], key=LargerNumKey,reverse = True)
        largest = "".join(str_nums)
        return largest
```

```python
#
# @lc app=leetcode.cn id=187 lang=python3
```

```python
#
# [187] 重复的DNA序列
#
class Solution:
    def findRepeatedDnaSequences(self, s: str) -> List[str]:
        dic, res = {}, set()
        for i in range(len(s)-9):
            dic[s[i:i+10]] = dic.get(s[i:i+10], 0)+1
            if dic[s[i:i+10]] > 1:
                res.add(s[i:i+10])
        return list(res)
```

```python
#
# @lc app=leetcode.cn id=188 lang=python3
#
# [188] 买卖股票的最佳时机IV
#
class Solution:
    def maxProfit(self, k: int, prices: List[int]) -> int:
        #交易次数太多，用贪心
        if k >= len(prices)//2:
            return self.greedy(prices)

        # k=0的时候此时sell为空
        # k小，动态规划
        buy, sell = [-prices[0]]*k , [0]*(k+1)
        for p in prices[1:]:
            for i in range(k):
                # 买的收益 = max(买、买了再买)
                buy[i] = max(buy[i], sell[i-1]-p)
                # 卖的收益 = (卖/买)
                sell[i] = max(sell[i], buy[i]+p)

        return max(sell)

    def greedy(self, prices):
        res = 0
        for i in range(1, len(prices)):
            if prices[i] > prices[i-1]:
                res += prices[i] - prices[i-1]
        return res
```

```python
#
# @lc app=leetcode.cn id=189 lang=python3
#
# [189] 旋转数组
```

```
#
class Solution:
    def rotate(self, nums: List[int], k: int) -> None:
        tmp = [0] * len(nums)
        for i in range(len(nums)):
            tmp[(i+k)%len(nums)] = nums[i] #recycle

        for i in range(len(nums)):
            nums[i] = tmp[i]
```

```
#
# @lc app=leetcode.cn id=190 lang=python3
#
# [190] 颠倒二进制位
#
class Solution:
    def reverseBits(self, n: int) -> int:
        res = 0
        bitsSize = 31
        while bitsSize > -1 and n:
            res += ((n%2) << bitsSize)
            n = n >> 1
            bitsSize -= 1
        return res
```

```
#
# @lc app=leetcode.cn id=198 lang=python3
#
# [198] 打家劫舍
#
class Solution:
    def rob(self, nums: List[int]) -> int:
        if not nums:
            return 0
        f1 = 0
        f2 = 0
        for i in nums:
            fi = max(f2+i,f1)
            f1 ,f2 = fi ,f1
        return f1
```

```
#
# @lc app=leetcode.cn id=199 lang=python3
#
# [199] 二叉树的右视图
#
```

```python
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    def rightSideView(self, root: TreeNode) -> List[int]:
        res = []
        self.dfs(root, 0,res)
        return res

    def dfs(self,root, depth,res):
        if not root:
            return
        if depth >= len(res):
            res.append(0)
        res[depth] = root.val
        # 先进行左子树的迭代,右子树迭代出来的值会覆盖到之前的上面去
        self.dfs(root.left, depth + 1,res)
        self.dfs(root.right, depth + 1,res)
```

```python
#
# @lc app=leetcode.cn id=200 lang=python3
#
# [200] 岛屿数量
#
class Solution:
    def numIslands(self, grid: List[List[str]]) -> int:
        if not grid:
            return 0
        m,n = len(grid),len(grid[0])

        res = 0
        for r in range(m):
            for c in range(n):
                if grid[r][c] == "1":
                    res += 1
                    self.dfs(grid,r,c,m,n)
        return res

    def dfs(self,grid,i,j,row,col):
        # 终止条件
        if i < 0 or j < 0 or i >= row or j >= col or grid[i][j] == "0":
            return
```

```
24          # 合法的话置位
25          grid[i][j] = "0"
26          self.dfs(grid,i-1,j,row,col)
27          self.dfs(grid,i,j-1,row,col)
28          self.dfs(grid,i+1,j,row,col)
29          self.dfs(grid,i,j+1,row,col)
```

```
1  #
2  # @lc app=leetcode.cn id=201 lang=python3
3  #
4  # [201] 数字范围按位与
5  #
6  class Solution:
7      def rangeBitwiseAnd(self, m: int, n: int) -> int:
8          '''
9          # 时间溢出
10         res = m
11         for i in range(m+1,n+1):
12             res = res & i
13             if res == 0:
14                 break
15         return res
16         '''
17
18         i = 0
19         while m != n:
20             m >>= 1
21             n >>= 1
22             i += 1
23         return m << i
```

```
1  #
2  # @lc app=leetcode.cn id=202 lang=python3
3  #
4  # [202] 快乐数
5  #
6  class Solution:
7      def isHappy(self, n: int) -> bool:
8          mem = set()
9          while n != 1:
10             # 求和
11             n = sum([int(i) ** 2 for i in str(n)])
12             if n in mem:
13                 # 陷入死循环了
14                 return False
15             else:
```

```
16              mem.add(n)
17          else:
18              return True
```

```
1  #
2  # @lc app=leetcode.cn id=203 lang=python3
3  #
4  # [203] 移除链表元素
5  #
6
7  # Definition for singly-linked list.
8  # class ListNode:
9  #     def __init__(self, x):
10 #         self.val = x
11 #         self.next = None
12
13 class Solution:
14     def removeElements(self, head: ListNode, val: int) -> ListNode:
15         dummy = ListNode(-1)
16         dummy.next = head
17         prev, curr = dummy, head
18         while curr:
19             if curr.val == val:
20                 # prev 跟上了 curr
21                 prev.next = curr.next
22             else:
23                 prev = curr
24             curr = curr.next
25         return dummy.next
```

```
1  #
2  # @lc app=leetcode.cn id=204 lang=python3
3  #
4  # [204] 计数质数
5  #
6  class Solution:
7      def countPrimes(self, n: int) -> int:
8          if n <= 2:
9              return 0
10         res = [0,0]+ [1]*(n-2)
11         for i in range(2,n):
12             # 这些没改过
13             if res[i] == 1:
14                 for j in range(2,(n-1)//i+1):
15                     res[i*j] = 0
16         return sum(res)
```

```python
#
# @lc app=leetcode.cn id=205 lang=python3
#
# [205] 同构字符串
#
class Solution:
    def isIsomorphic(self, s: str, t: str) -> bool:
        if len(s) != len(t):
            return False

        mapStoT = [0] * 128
        mapTtoS = [0] * 128
        for i in range(len(s)):
            s_num, t_num = ord(s[i]), ord(t[i])
            if mapStoT[s_num] == 0 and mapTtoS[t_num] == 0:
                mapStoT[s_num] = t_num
                mapTtoS[t_num] = s_num
            elif mapTtoS[t_num] != s_num or mapStoT[s_num] != t_num:
                return False
        return True
```

```python
#
# @lc app=leetcode.cn id=206 lang=python3
#
# [206] 反转链表
#
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution:
    def reverseList(self, head: ListNode) -> ListNode:
        if head is None or head.next is None:
            return head
        curr = head # 他来往后走
        prev = None # 新的反转的
        while curr:
            # 下一步先保存下来
            nextcurr = curr.next
            # 反转的接上去
            curr.next = prev
            prev = curr
            # 下一步
            curr = nextcurr
```

```
26          return prev
```

```
1   #
2   # @lc app=leetcode.cn id=213 lang=python3
3   #
4   # [213] 打家劫舍 II
5   #
6   class Solution:
7       def rob(self, nums: List[int]) -> int:
8           if not nums:
9               return 0
10          if len(nums) == 1:
11              return nums[0]
12          # 奇偶串
13          return max(
14              self.robb(nums[0:-1]),
15              self.robb(nums[1:])
16          )
17
18      def robb(self, nums):
19          mx = prev = 0
20          for i in nums:
21              temp = mx
22              mx = max(mx, prev + i)
23              prev = temp
24          return mx
```

```
1   #
2   # @lc app=leetcode.cn id=215 lang=python3
3   #
4   # [215] 数组中的第K个最大元素
5   #
6
7   class Solution:
8       def findKthLargest(self, nums: List[int], k: int) -> int:
9           '''
10          nums.sort()
11          return nums[-k]
12          '''
13          return self.qSelect(nums, 0, len(nums) - 1, k)
14
15      def qSelect(self, nums, start, end, k):
16          '''
17          if start > end:
18              return float('inf')
19          '''
```

```python
20          # 找一个参照值
21          pivot = nums[end]
22          left = start
23          for i in range(start, end):
24              # 比参照大的都移到左边去
25              if nums[i] >= pivot:
26                  nums[left], nums[i] = nums[i], nums[left]
27                  left += 1
28          # 参照值也拉倒左边去
29          nums[left], nums[end] = nums[end], nums[left]
30          # 左边的个数够没(从0开始到k−1,共k个)
31          if left == k−1:
32              return nums[left]
33          # 还不够
34          elif left < k−1:
35              return self.qSelect(nums, left + 1, end, k)
36          # 太多了
37          else:
38              return self.qSelect(nums, start, left − 1, k)
```

```python
#
# @lc app=leetcode.cn id=216 lang=python3
#
# [216] 组合总和 III
#
class Solution:
    def combinationSum3(self, k: int, n: int) -> List[List[int]]:
        res = []
        self.dfs(k,n ,1,[], res)
        return res

    def dfs(self,k,target,start,path,res):
        # 终止条件
        if target == 0 and len(path) == k:
            res.append(path)
            return
        elif target < 0 or len(path) > k or start > 9:
            return

        for i in range(start,10):
            self.dfs(k,target−i,i+1,path+[i],res)
```

```python
#
# @lc app=leetcode.cn id=217 lang=python3
#
# [217] 存在重复元素
```

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

```python
#
# @lc app=leetcode.cn id=219 lang=python3
#
# [219] 存在重复元素 II
#
class Solution:
    def containsNearbyDuplicate(self, nums: List[int], k: int) -> bool:
        dic = {}
        for key, val in enumerate(nums):
            if val in dic and key - dic[val]<=k:
                return True
            dic[val] = key
        return False
```

```python
#
# @lc app=leetcode.cn id=220 lang=python3
#
# [220] 存在重复元素 III
#
class Solution:
    def containsNearbyAlmostDuplicate(self, nums: List[int], k: int, t: int) -> bool:
        if t < 0 or k < 0:
            return False
        all_buckets = {}
        # 桶的大小设成t+1更加方便
        bucket_size = t + 1
        for i in range(len(nums)):
            # 放入哪个桶
            bucket_num = nums[i] // bucket_size
            # 桶中已经有元素了
            if bucket_num in all_buckets:
                return True
            # 把nums[i]放入桶中
            all_buckets[bucket_num] = nums[i]
            # 检查前一个桶
            if (bucket_num - 1) in all_buckets and abs(all_buckets[bucket_num - 1] - nums[i])
                <= t:
                return True
            # 检查后一个桶
            if (bucket_num + 1) in all_buckets and abs(all_buckets[bucket_num + 1] - nums[i])
                <= t:
```

```python
            return True

            # 如果不构成返回条件，那么当i >= k 的时候就要删除旧桶了，以维持桶中的元素索引
            #   跟下一个i+1索引只差不超过k
            if i >= k:
                all_buckets.pop(nums[i-k]//bucket_size)

        return False
```

```python
#
# @lc app=leetcode.cn id=221 lang=python3
#
# [221] 最大正方形
#
class Solution:
    def maximalSquare(self, matrix: List[List[str]]) -> int:
        if not matrix:
            return 0
        row ,col = len(matrix) ,len(matrix[0])

        # 多了一行一列
        dp = [ [0 for _ in range(col + 1)] for _ in range(row + 1)]
        res = 0
        for i in range(1, row +1):
            for j in range(1, col + 1):
                if matrix[i - 1][j - 1] == "1":
                    # 否则dp为0, 不用操作
                    dp[i][j] = min(dp[i-1][j - 1], dp[i - 1][j], dp[i][j - 1]) + 1
                    res = max(res, dp[i][j] ** 2)
        return res
```

```python
#
# @lc app=leetcode.cn id=222 lang=python3
#
# [222] 完全二叉树的节点个数
#

# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    def countNodes(self, root: TreeNode) -> int:
```

```python
        if not root:
            return 0

        # return 1 + self.countNodes(root.left) + self.countNodes(root.right)

        h_l, h_r = 0, 0
        # 计算当前节点左子树的最大高度
        curRoot = root
        while curRoot.left:
            h_l += 1
            curRoot = curRoot.left
        # 计算当前节点右子树的最大高度
        curRoot = root
        if curRoot.right:
            h_r += 1
            curRoot = curRoot.right
            while curRoot.left:
                h_r += 1
                curRoot = curRoot.left

        # 左右子树最大高度相同，说明左子树为满二叉树，在右子树继续递归求解
        if h_l == h_r:
            sumNodes_r = self.countNodes(root.right)
            sumNodes_l = 2**h_l - 1
        # 左子树高度更高，说明右子树为满二叉树，在左子树继续递归求解
        if h_l == h_r + 1:
            sumNodes_l = self.countNodes(root.left)
            sumNodes_r = 2**h_r - 1

        # 返回左子节点个数+右子节点个数+当前根节点
        return sumNodes_l + sumNodes_r + 1
```

```python
#
# @lc app=leetcode.cn id=223 lang=python3
#
# [223] 矩形面积
#
class Solution:
    def computeArea(self, A: int, B: int, C: int, D: int, E: int, F: int, G: int, H: int) -> int:
        x = min( C,G )- max(A,E)
        y = min( D,H )- max(B,F)
        return (A-C)*(B-D) + (E-G)*(F-H) - max(x,0)*max(y,0)
```

```python
#
# @lc app=leetcode.cn id=224 lang=python3
#
```

```python
# [224] 基本计算器
#
class Solution:
    def calculate(self, s: str) -> int:
        res = 0
        sign = 1
        stack = []
        i = 0
        while i < len(s):
            c = s[i]
            if c.isdigit():
                start = i
                while i < len(s) and s[i].isdigit():
                    i += 1
                res += sign * int(s[start:i])
                # 因为后加1,不满足while的时候此时的i已经不是数字,需要回退一步,和后边加1对冲
                i -= 1
            elif c == '+':
                sign = 1
            elif c == '-':
                sign = -1
            elif c == "(":
                stack.append(res)
                stack.append(sign)
                res = 0
                sign = 1
            elif c == ")":
                # 现在的res是括号里面的计算结果
                # 需要乘以对应的符号
                res *= stack.pop()
                res += stack.pop()
            i += 1
        return res
```

```python
#
# @lc app=leetcode.cn id=225 lang=python3
#
# [225] 用队列实现栈
#
class MyStack:
    def __init__(self):
        self.list = []

    def push(self, x: int) -> None:
        # 尾部压入
        self.list.append(x)
```

```python
    def pop(self) -> int:
        # 尾部弹出
        if len( self . list ) == 0:
            return
        else :
            temp = self. list [−1]
            del  self . list [−1]
            return temp

    def top( self ) -> int:
        if len( self . list ) == 0:
            return
        else :
            return  self . list [−1]

    def empty(self) -> bool:
        return len( self . list ) == 0


# Your MyStack object will be instantiated and called as such:
# obj = MyStack()
# obj.push(x)
# param_2 = obj.pop()
# param_3 = obj.top()
# param_4 = obj.empty()
```

```python
#
# @lc app=leetcode.cn id=226 lang=python3
#
# [226] 翻转二叉树
#
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self . val = x
#         self . left = None
#         self . right = None

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

```python
#
# @lc app=leetcode.cn id=229 lang=python3
#
# [229] 求众数 II
#
class Solution:
    def majorityElement(self, nums: List[int]) -> List[int]:
        # 摩尔投票法得到两个大多数
        result1 , result2 = -1, -1
        score1 , score2 = 0 , 0
        for i in range(len(nums)):
            if (result1==nums[i]):
                score1+=1
            elif (result2==nums[i]):
                score2+=1
            elif (score1==0):
                result1=nums[i]
                score1=1
            elif (score2==0):
                result2=nums[i]
                score2=1
            else :
                score1 -= 1
                score2 -= 1

        # 统计两个大多数的出现次数
        time1,time2 = 0 , 0
        for i in range(len(nums)):
            if    (nums[i]==result1): time1+=1
            elif (nums[i]==result2): time2+= 1

        # 得到结果
        result = []
        if (time1>len(nums)/3): result.append(result1)
        if (time2>len(nums)/3): result.append(result2)
        return result
```

```python
#
# @lc app=leetcode.cn id=230 lang=python3
#
# [230] 二叉搜索树中第K小的元素
#

# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
```

```
10  #          self.val  = x
11  #          self.left  = None
12  #          self.right  = None
13
14  class Solution:
15      def kthSmallest(self, root: TreeNode, k: int) -> int:
16          '''
17          # 方法一
18          reslist  = self.inorder(root)
19          return  reslist [k - 1]
20          '''
21          # 方法二
22          # 左子树有多少个点
23          n = self.count(root.left )
24          if n == k -1 :
25              return root.val
26          # 递归到左子树
27          elif  n > k - 1:
28              return  self.kthSmallest(root.left ,k)
29          # 递归到右子树
30          else :
31              return  self.kthSmallest(root.right ,k-1-n)
32
33      def inorder( self ,r):
34          if r:
35              return  self.inorder(r. left ) + [r.val] + self.inorder(r.right )
36          else :
37              return  []
38
39      def count( self ,root):
40          if not root :
41              return 0
42          return  self.count(root. left ) + self.count(root.right ) + 1
```

```
1   #
2   # @lc app=leetcode.cn id=231 lang=python3
3   #
4   # [231] 2的幂
5   #
6   class Solution:
7       def isPowerOfTwo(self, n: int ) -> bool:
8           while n > 1:
9               n /= 2
10          if n == 1:
11              return True
12          else :
```

```
13                return False
```

```
1   #
2   # @lc app=leetcode.cn id=232 lang=python3
3   #
4   # [232] 用栈实现队列
5   #
6   class MyQueue:
7       def __init__(self):
8           self.stack = []
9
10      def push(self, x: int) -> None:
11          # 尾部加入
12          self.stack.append(x)
13
14      def pop(self) -> int:
15          temp = self.stack[0]
16          self.stack.pop(0)
17          return temp
18
19      def peek(self) -> int:
20          return self.stack[0]
21
22      def empty(self) -> bool:
23          return len(self.stack) == 0
24
25  # Your MyQueue object will be instantiated and called as such:
26  # obj = MyQueue()
27  # obj.push(x)
28  # param_2 = obj.pop()
29  # param_3 = obj.peek()
30  # param_4 = obj.empty()
```

```
1   #
2   # @lc app=leetcode.cn id=233 lang=python3
3   #
4   # [233] 数字 1 的个数
5   #
6   class Solution:
7       def countDigitOne(self, n: int) -> int:
8           res = 0
9           a = 1
10          b = 1
11          while n >= 1:
12              # 用(x+8)//10来判断一个数是否大于等于2
13              # 从低位到高位
```

```python
14          res += (n + 8)//10*a
15          if n % 10 == 1:
16              res += b
17          b += n % 10 * a
18          a *= 10
19          n //= 10
20      return res
```

```python
1  #
2  # @lc app=leetcode.cn id=234 lang=python3
3  #
4  # [234] 回文链表
5  #
6  # Definition for singly-linked list.
7  # class ListNode:
8  #     def __init__(self, x):
9  #         self.val = x
10 #         self.next = None
11
12 class Solution:
13     def isPalindrome(self, head: ListNode) -> bool:
14         if head is None:
15             return True
16         rev = None
17         slow = fast = head
18         # fast 到尾部
19         # slow 到中部
20         # rev 前半部分的反向
21         while fast and fast.next:
22             fast = fast.next.next
23             rev, rev.next ,slow = slow, rev , slow.next
24         # 奇
25         if fast:
26             slow = slow.next
27         # 一个向左,一个向右
28         while rev :
29             if rev.val != slow.val:
30                 return False
31             slow = slow.next
32             rev = rev.next
33         return True
```

```python
1  #
2  # @lc app=leetcode.cn id=235 lang=python3
3  #
4  # [235] 二叉搜索树的最近公共祖先
```

```python
#
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    def lowestCommonAncestor(self, root: 'TreeNode', p: 'TreeNode', q: 'TreeNode') -> 'TreeNode':
        if p is None or q is None or root is None:
            return None
        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
```

```python
#
# @lc app=leetcode.cn id=236 lang=python3
#
# [236] 二叉树的最近公共祖先
#

# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    def lowestCommonAncestor(self, root: 'TreeNode', p: 'TreeNode', q: 'TreeNode') -> 'TreeNode':
        #若root为空或者root为p或者root为q,说明找到了p或q其中一个
        if(root is None or root== p or root== q):
            return root

        left  = self.lowestCommonAncestor(root.left,p,q)
        right = self.lowestCommonAncestor(root.right,p,q)

        #若左子树找到了p,右子树找到了q,说明此时的root就是公共祖先
        if left and right :
            return root
        # 若左子树是none右子树不是，说明右子树找到了p或q
        if not left :
```

```python
28          return right
29      # 同理
30      if not right :
31          return left
32      return None
```

```python
1  #
2  # @lc app=leetcode.cn id=237 lang=python3
3  #
4  # [237] 删除链表中的节点
5  #
6  # Definition for singly−linked list .
7  # class ListNode:
8  #     def __init__(self, x):
9  #         self .val = x
10 #         self .next = None
11
12 class Solution:
13     def deleteNode(self, node):
14         node.val = node.next.val
15         node.next = node.next.next
```

```python
1  #
2  # @lc app=leetcode.cn id=238 lang=python3
3  #
4  # [238] 除自身以外数组的乘积
5  #
6  class Solution:
7      def productExceptSelf(self, nums: List[int]) −> List[int]:
8          res = [1] ∗ len(nums)
9          right = 1
10         for i in range(1, len(nums)):
11             res[i] = res[i − 1] ∗ nums[i − 1]
12
13         for i in range(len(nums) − 1, −1, −1):
14             res[i] ∗= right
15             right ∗= nums[i]
16         return res
```

```python
1  #
2  # @lc app=leetcode.cn id=240 lang=python3
3  #
4  # [240] 搜索二维矩阵 II
5  #
6  class Solution:
7      def searchMatrix(self, matrix, target):
8          if not len(matrix) or not len(matrix[0]):
```

```
 9              return False
10          # 左下角
11          r , c = len(matrix) − 1 , 0
12          while r >= 0 and c < len(matrix[0]):
13              if matrix[r][c] > target :
14                  # 往上
15                  r −= 1
16              elif matrix[r][c] < target :
17                  # 往右
18                  c += 1
19              else :
20                  return True
21          return False
```

```
 1  #
 2  # @lc app=leetcode.cn id=242 lang=python3
 3  #
 4  # [242] 有效的字母异位词
 5  #
 6  class Solution:
 7      def isAnagram(self, s: str, t: str) −> bool:
 8          dic1, dic2 = {}, {}
 9          for item in s:
10              dic1[item] = dic1.get(item, 0) + 1
11          for item in t:
12              dic2[item] = dic2.get(item, 0) + 1
13          return dic1 == dic2
```

```
 1  #
 2  # @lc app=leetcode.cn id=257 lang=python3
 3  #
 4  # [257] 二叉树的所有路径
 5  #
 6  # Definition for a binary tree node.
 7  # class TreeNode:
 8  #     def __init__(self, x):
 9  #         self.val = x
10  #         self.left = None
11  #         self.right = None
12
13  class Solution:
14      def binaryTreePaths(self, root: TreeNode) −> List[str]:
15          if not root:
16              return []
17          res = []
18          self.dfs(root, [] ,res )
```

```
19          paths = ['->'.join(path) for path in res ]
20          return paths
21
22      def dfs( self  , node , path ,res):
23          # 终止条件 没有子节点
24          if not node.left  and not node.right:
25              res.append(path+[str(node.val)])
26              return
27          path = path + [str(node.val)]
28          if node.left :
29              self .dfs(node.left   , path , res  )
30          if node.right :
31              self .dfs(node.right  , path, res  )
```

```
1  #
2  # @lc app=leetcode.cn id=258 lang=python3
3  #
4  # [258] 各位相加
5  #
6  class Solution:
7      def addDigits( self , num: int)  -> int:
8          t = num
9          while t >= 10 :
10             t = sum([int(char) for char in str(t)])
11         return t
```

```
1  #
2  # @lc app=leetcode.cn id=263 lang=python3
3  #
4  # [263] 丑数
5  #
6  class Solution:
7      def isUgly( self , num: int)  -> bool:
8          if num <= 0 :
9              return False
10
11         divisors = [2, 3, 5]
12         for d in divisors :
13             while num % d == 0:
14                 num /= d
15         return num == 1
```

```
1  #
2  # @lc app=leetcode.cn id=264 lang=python3
3  #
4  # [264] 丑数 II
5  #
```

```python
class Solution:
    def nthUglyNumber(self, n: int) -> int:
        ugly = [1]
        i2,i3,i5 = 0,0,0
        idx = 1
        while idx < n :
            newugly = min([ugly[i2]*2 , ugly[i3]*3 ,ugly[i5]*5])
            ugly.append(newugly)

            while ugly[i2]*2<= newugly:
                i2 += 1
            while ugly[i3]*3<= newugly:
                i3 += 1
            while ugly[i5]*5<= newugly:
                i5 += 1
            idx += 1
        return ugly[-1]
```

```python
#
# @lc app=leetcode.cn id=268 lang=python3
#
# [268] 缺失数字
#
class Solution:
    def missingNumber(self, nums: List[int]) -> int:
        return  len(nums)*(len(nums)+1)//2 - sum(nums)
```

```python
#
# @lc app=leetcode.cn id=274 lang=python3
#
# [274] H指数
#
class Solution:
    def hIndex(self, citations: List[int]) -> int:
        citations.sort()
        i = 0
        while i<len(citations) and citations[len(citations)-1-i]>i:
            i += 1
        return i
```

```python
#
# @lc app=leetcode.cn id=275 lang=python3
#
# [275] H指数 II
#
class Solution:
    def hIndex(self, citations: List[int]) -> int:
```

```
8            i = 0
9            while i<len(citations) and citations[len( citations )−1−i]>i:
10               i += 1
11           return i
```

```
1   #
2   # @lc app=leetcode.cn id=278 lang=python3
3   #
4   # [278] 第一个错误的版本
5   #
6   # The isBadVersion API is already defined for you.
7   # @param version, an integer
8   # @return a bool
9   # def isBadVersion(version):
10
11  class Solution:
12      def firstBadVersion( self ,  n):
13          l ,r = 0 , n−1
14          while l <= r:
15              mid = (l+r)//2
16              if isBadVersion(0) == isBadVersion(mid):
17                  l = mid + 1
18              elif isBadVersion(n) == isBadVersion(mid):
19                  r = mid −1
20          return l
```

```
1   #
2   # @lc app=leetcode.cn id=279 lang=python3
3   #
4   # [279] 完全平方数
5   #
6   class Solution:
7       def numSquares(self, n: int )  −> int:
8           dp = list (range(n+1))
9           for  i  in  range(2,n+1):
10              for  j  in  range(1,int (i∗∗(0.5))+1):
11                  dp[i]=min(dp[i],dp[i−j∗j]+1)
12          return dp[−1]
```

```
1   #
2   # @lc app=leetcode.cn id=283 lang=python3
3   #
4   # [283] 移动零
5   #
6   class Solution:
7       def moveZeroes(self, nums: List[int])  −> None:
8           '''
```

```python
        zeros = []
        for i in range(len(nums)):
            if nums[i] == 0 :
                zeros.append(i)

        for i in zeros [::−1]:
            nums.pop(i)
            nums.append(0)
        return nums
        '''
        j = 0
        for i in range(len(nums)):
            if nums[i] != 0:
                nums[j] = nums[i]
                j += 1
        for i in range(j, len(nums)):
            nums[i] = 0
```

```python
#
# @lc app=leetcode.cn id=290 lang=python3
#
# [290] 单词规律
#
class Solution:
    def wordPattern(self, pattern: str, str: str) −> bool:
        '''
        word_list = str.split (' ')
        pattern_list = list (pattern)
        if len(word_list) != len(pattern_list):
            return False
        for i, word in enumerate(word_list):
            idx = word_list.index(word)
            idx2 = pattern_list.index(pattern[i])
            if idx != idx2:
                return False
        return True
        '''

        """
        思路:
        """

        words = str.split ("␣")
        hash_table_pattern = {}
        hash_table_words = {}
```

```python
        if len(words) != len(pattern):
            return False
        #第一步
        for i, letter in enumerate(pattern):
            if letter in hash_table_pattern:
                if hash_table_pattern[letter] != words[i]:
                    return False
            else:
                hash_table_pattern[letter] = words[i]
        #第二步
        for i, word in enumerate(words):
            if word in hash_table_words:
                if hash_table_words[word] != pattern[i]:
                    return False
            else:
                hash_table_words[word] = pattern[i]
        return True
```

```python
#
# @lc app=leetcode.cn id=292 lang=python3
#
# [292] Nim 游戏
#
class Solution:
    def canWinNim(self, n: int) -> bool:
        return n%4 != 0
```

```python
#
# @lc app=leetcode.cn id=299 lang=python3
#
# [299] 猜数字游戏
#
class Solution:
    def getHint(self, secret: str, guess: str) -> str:
        a = b = 0
        dic = {}
        for i in range(len(secret)):
            if secret[i] == guess[i]:
                a += 1
            dic[secret[i]] = dic.get(secret[i],0) + 1
        for i in range(len(guess)):
            if guess[i] in dic and dic[guess[i]] > 0 :
                b += 1
                dic[guess[i]] -= 1
        b -= a
        return f"{a}A{b}B"
```

```python
#
# @lc app=leetcode.cn id=300 lang=python3
#
# [300] 最长上升子序列
#
class Solution:
    def lengthOfLIS(self, nums: List[int]) -> int:
        if not nums:
            return 0

        '''
        dp = [1] * len(nums)
        for i in range(1,len(nums)):
            for j in range(i):
                # 如果要求非严格递增,将此行 '<' 改为 '<=' 即可
                if(nums[j] < nums[i]):
                    dp[i] = max(dp[i] , dp[j] + 1 )
        return max(dp)
        '''

        up_list = []
        for i in range(len(nums)):
            # 二分查找
            left , right = 0, len(up_list)-1
            while left <= right:
                mid = (left+right)//2
                if up_list[mid] < nums[i]:
                    left = mid+1
                else:
                    right = mid-1
            # 若 left 等于数组长度，则需要添加新值；否则，在 left 位置的值覆盖为新值
            if left == len(up_list):
                up_list.append(nums[i])
            else:
                up_list[left] = nums[i]
        return len(up_list)
```

```python
#
# @lc app=leetcode.cn id=303 lang=python3
#
# [303] 区域和检索 - 数组不可变
#
class NumArray:

    def __init__(self, nums: List[int]):
        self.list = [0] *(len(nums)+1)
```

```
10              for i in range(len(nums)):
11                  self.list[i+1] = self.list[i] + nums[i]
12
13      def sumRange(self, i: int, j: int) -> int:
14          return self.list[j+1] - self.list[i]
15
16
17  # Your NumArray object will be instantiated and called as such:
18  # obj = NumArray(nums)
19  # param_1 = obj.sumRange(i,j)
```

```
1   #
2   # @lc app=leetcode.cn id=309 lang=python3
3   #
4   # [309] 最佳买卖股票时机含冷冻期
5   #
6   class Solution:
7       def maxProfit(self, prices: List[int]) -> int:
8           if len(prices) < 2:
9               return 0
10          sale = [0 for _ in range(len(prices))]
11          buy = [0 for _ in range(len(prices))]
12          cool = [0 for _ in range(len(prices))]
13
14          buy[0] = -prices[0]
15
16          for i in range(1, len(prices)):
17              cool[i] = sale[i-1]
18              buy[i] = max(buy[i-1], cool[i-1] - prices[i])
19              sale[i] = max(sale[i-1], buy[i] + prices[i])
20
21          return max(sale[-1], cool[-1])
```

```
1   #
2   # @lc app=leetcode.cn id=313 lang=python3
3   #
4   # [313] 超级丑数
5   #
6   class Solution:
7       def nthSuperUglyNumber(self, n: int, primes: List[int]) -> int:
8           ugly = [1]
9           ls = len(primes)
10          ix = [0]*ls
11          idx = 1
12          while idx < n:
13              newugly = min([ugly[ix[i]]*primes[i] for i in range(ls)])
```

```
14              ugly.append(newugly)
15              for i in range(ls):
16                  while ugly[ix[i]]*primes[i]<= newugly:
17                      ix[i] += 1
18              idx += 1
19          return ugly[-1]
```

```
1  #
2  # @lc app=leetcode.cn id=319 lang=python3
3  #
4  # [319] 灯泡开关
5  #
6  class Solution:
7      def bulbSwitch(self, n: int) -> int:
8          return int(math.sqrt(n))
```

```
1  #
2  # @lc app=leetcode.cn id=322 lang=python3
3  #
4  # [322] 零钱兑换
5  #
6  class Solution:
7      def coinChange(self, coins: List[int], amount: int) -> int:
8          if amount == 0:
9              return 0
10         if not coins :
11             return -1
12
13         coins.sort()
14         dp = [float('inf')] * (amount + 1)
15         dp[0] = 0
16
17         for coin in coins:
18             for j in range(coin, amount+1):
19                 dp[j] = min(dp[j], dp[j - coin] + 1)
20
21         if dp[-1] > amount:
22             return -1
23         else:
24             return dp[-1]
```

```
1  #
2  # @lc app=leetcode.cn id=326 lang=python3
3  #
4  # [326] 3的幂
5  #
6  class Solution:
```

```python
 7        def isPowerOfThree(self, n: int) -> bool:
 8            while n > 1:
 9                n /= 3
10            if  n == 1:
11                return True
12            else:
13                return False
```

```python
 1  #
 2  # @lc app=leetcode.cn id=335 lang=python3
 3  #
 4  # [335] 路径交叉
 5  #
 6  class Solution:
 7      def isSelfCrossing(self, x: List[int]) -> bool:
 8          for i in range(len(x)):
 9              if i + 3 < len(x) and x[i] >= x[i + 2] \
10                  and x[i + 1] <= x[i + 3]:
11                  return True
12              if i + 4 < len(x) and x[i + 1] == x[i + 3] \
13                  and x[i] + x[i + 4] >= x[i + 2]:
14                  return True
15              if i + 5 < len(x) and x[i] < x[i + 2] \
16                  and x[i + 4] < x[i + 2]       \
17                  and x[i + 2] <= x[i] + x[i + 4]  \
18                  and x[i + 1] < x[i + 3] \
19                  and x[i + 3] <= x[i + 1] + x[i + 5]:
20                  return True
21          return False
```

```python
 1  #
 2  # @lc app=leetcode.cn id=342 lang=python3
 3  #
 4  # [342] 4的幂
 5  #
 6  class Solution:
 7      def isPowerOfFour(self, num: int) -> bool:
 8          # bin(4**0) '0b1'
 9          # bin(4**1) '0b100'
10          # bin(4**2) '0b10000'
11          # bin(4**3) '0b1000000'
12          return num > 0 and num & (num-1) == 0 and len(bin(num)[3:]) % 2 == 0
```

```python
 1  #
 2  # @lc app=leetcode.cn id=344 lang=python3
 3  #
 4  # [344] 反转字符串
```

```
 5   #
 6   class Solution:
 7       def reverseString(self, s: List[str]) -> None:
 8           n =len(s)
 9           for i in range(n//2):
10               s[i],s[n-i-1] = s[n-i-1],s[i]
```

```
 1   #
 2   # @lc app=leetcode.cn id=345 lang=python3
 3   #
 4   # [345] 反转字符串中的元音字母
 5   #
 6   class Solution:
 7       def reverseVowels(self, s: str) -> str:
 8           s = list(s)
 9           n =len(s)
10           l ,r = 0,n-1
11           while l < r:
12               if s[l] not in 'aeiouAEIOU':
13                   l += 1
14               elif s[r] not in 'aeiouAEIOU':
15                   r -= 1
16               else:
17                   s[l],s[r] = s[r],s[l]
18                   l += 1
19                   r -= 1
20           return ''.join(s)
```

```
 1   #
 2   # @lc app=leetcode.cn id=349 lang=python3
 3   #
 4   # [349] 两个数组的交集
 5   #
 6   class Solution:
 7       def intersection(self, nums1: List[int], nums2: List[int]) -> List[int]:
 8           return list(set(nums1) & set(nums2))
```

```
 1   #
 2   # @lc app=leetcode.cn id=350 lang=python3
 3   #
 4   # [350] 两个数组的交集 II
 5   #
 6   class Solution:
 7       def intersect(self, nums1: List[int], nums2: List[int]) -> List[int]:
 8           nums1.sort()
 9           nums2.sort()
10           res = []
```

```python
            pos1 = pos2 = 0
            while pos1 < len(nums1) and pos2 < len(nums2):
                if nums1[pos1] == nums2[pos2]:
                    res.append(nums1[pos1])
                    pos1 += 1
                    pos2 += 1
                elif nums1[pos1] < nums2[pos2]:
                    pos1 += 1
                else:
                    pos2 += 1
            return res
```

```python
#
# @lc app=leetcode.cn id=354 lang=python3
#
# [354] 俄罗斯套娃信封问题
#
class Solution:
    def maxEnvelopes(self, envelopes: List[List[int]]) -> int:
        if not envelopes:
            return 0
        '''
        # 超时
        envelopes.sort(key=lambda x:x[0])
        dp = [1] * len(envelopes)
        for i in range(len(envelopes)):
            for j in range(i):
                if envelopes[i][0] > envelopes[j][0] and envelopes[i][1] > envelopes[j][1]:
                    dp[i] = max(dp[i],dp[j]+1)
        return max(dp)
        '''

        from bisect import bisect_left
        # 在L中查找x,x存在时返回x左侧的位置,x不存在返回应该插入的位置
        # 按w升序,h降序排列
        envelopes.sort(key=lambda x:(x[0], -x[1]))
        up_list = []
        for e in envelopes:
            index = bisect_left(up_list, e[1])
            if index == len(up_list):
                up_list.append(e[1])
            else:
                up_list[index] = e[1]
        return len(up_list)
```

```python
#
```

```python
# @lc app=leetcode.cn id=367 lang=python3
#
# [367] 有效的完全平方数
#
class Solution:
    def isPerfectSquare(self, num: int) -> bool:
        '''
        l,r = 1,num
        while l <= r:
            mid = (l+r)//2
            if mid ** 2 == num:
                return True
            elif mid ** 2 < num:
                l = mid +1
            else:
                r = mid -1
        return False
        '''
        x = num
        while x ** 2 > num:
            x = (x+num//x)//2
        return x ** 2 == num
```

```python
#
# @lc app=leetcode.cn id=371 lang=python3
#
# [371] 两整数之和
#
class Solution:
    def getSum(self, a: int, b: int) -> int:
        MAX_INT = 0x7FFFFFFF
        MIN_INT = 0x80000000
        MASK = 0x100000000
        while b:
            a, b = (a ^ b) % MASK, ((a & b) << 1) % MASK
        return a if a <= MAX_INT else ~((a % MIN_INT) ^ MAX_INT)
```

```python
#
# @lc app=leetcode.cn id=374 lang=python3
#
# [374] 猜数字大小
#
# The guess API is already defined for you.
# @return -1 if my number is lower, 1 if my number is higher, otherwise return 0
# def guess(num: int) -> int:

```

```
10  class Solution:
11      def guessNumber(self, n: int) -> int:
12          start, end = 1, n
13          while start <= end:
14              mid = (start + end)//2
15              if guess(mid) == 0:
16                  return mid
17              elif guess(mid) == 1:
18                  start = mid + 1
19              else:
20                  end = mid
```

```
1  #
2  # @lc app=leetcode.cn id=383 lang=python3
3  #
4  # [383] 赎金信
5  #
6  class Solution:
7      def canConstruct(self, ransomNote: str, magazine: str) -> bool:
8          letter_map = {}
9          for i in magazine:
10             letter_map[i] = letter_map.get(i, 0) + 1
11         for i in ransomNote:
12             letter_map[i] = letter_map.get(i, 0) - 1
13             if letter_map[i]<0:
14                 return False
15         return True
```

```
1  #
2  # @lc app=leetcode.cn id=387 lang=python3
3  #
4  # [387] 字符串中的第一个唯一字符
5  #
6  class Solution:
7      def firstUniqChar(self, s: str) -> int:
8          letter_map = {}
9          for i in s:
10             letter_map[i] = letter_map.get(i, 0) + 1
11         for i in range(len(s)):
12             if letter_map[s[i]] == 1:
13                 return i
14         return -1
```

```
1  #
2  # @lc app=leetcode.cn id=393 lang=python3
3  #
4  # [393] UTF-8 编码验证
```

```python
#
class Solution:
    def validUtf8(self, data: List[int]) -> bool:
        # cnt表示后面接几个字节字符
        # cnt 从0到0表示一个字符
        cnt = 0
        for d in data:
            if cnt == 0:
                if (d >> 5) == 0b110:
                    cnt = 1
                elif (d >> 4) == 0b1110:
                    cnt = 2
                elif (d >> 3) == 0b11110:
                    cnt = 3
                # 0xxxxxxx 后面不接
                # 这种情况首位不是0就错
                elif (d >> 7):
                    return False
            else:
                # 如果不接10xxxxxx
                if (d >> 6) != 0b10:
                    return False
                cnt -= 1
        return cnt == 0
```

```python
#
# @lc app=leetcode.cn id=414 lang=python3
#
# [414] 第三大的数
#
class Solution:
    def thirdMax(self, nums: List[int]) -> int:
        nums = list(set(nums))
        if len(nums)<3:
            return max(nums)
        nums.sort()
        return nums[-3]
```

```python
#
# @lc app=leetcode.cn id=416 lang=python3
#
# [416] 分割等和子集
#
class Solution:
    def canPartition(self, nums: List[int]) -> bool:
        # 背包问题+动态规划
```

```
 9          target  = sum(nums)
10          if  target  % 2 == 1:
11              return  False
12          target  //= 2
13
14          # 行nums 列对应 目标值
15          dp = [[False]*( target+1) for _ in range(len(nums))]
16          # 每一列赋值
17          if  nums[0] <= target:
18              dp [0][nums[0]] = True
19
20          for  i  in range(1,len(nums)):
21              for  j  in range(1,target+1):
22                  if  j  >= nums[i]:
23                      dp[i ][ j] = dp[i−1][j] or dp[i−1][j−nums[i]]
24                  else :
25                      dp[i ][ j] = dp[i−1][j]
26          return dp[−1][−1]
```

```
 1  #
 2  # @lc app=leetcode.cn id=434 lang=python3
 3  #
 4  # [434] 字符串中的单词数
 5  #
 6  class  Solution :
 7      def countSegments(self, s:  str ) −> int:
 8          if  not s:
 9              return  0
10          '''
11          segment_count = 0
12          for  i  in range(len(s )):
13              if  i  == 0 and s[i] != ' ':
14                  segment_count = 1
15              elif  s [i−1] == ' ' and s[i] != ' ':
16                  segment_count += 1
17
18          return segment_count
19          '''
20          s_list  = list (s. split ("␣"))
21          s_list  = [i  for  i  in  s_list  if  i != "␣" and i != "" ]
22          return len(s_list )
```

```
 1  #
 2  # @lc app=leetcode.cn id=442 lang=python3
 3  #
 4  # [442] 数组中重复的数据
```

```python
#
class Solution:
    def findDuplicates( self , nums: List[int]) -> List[int]:
        res = []
        for x in nums:
            x = abs(x)
            # 若x出现过了,x-1对应位置的值是负的(减一是为了超出范围)
            if nums[x-1] < 0:
                res.append(x)
            else:
                nums[x-1] *= -1
        return res
```

```python
#
# @lc app=leetcode.cn id=443 lang=python3
#
# [443] 压缩字符串
#
class Solution:
    def compress(self, chars: List[str]) -> int:
        # count 几个一样
        # walker 写入的位置
        # runner 往后跑的
        walker, runner = 0, 0

        while runner < len(chars):
            # 写字符
            chars[walker] = chars[runner]
            count = 1

            while runner + 1 < len(chars) and \
            chars[runner] == chars[runner+1] :
                runner += 1
                count += 1

            if count > 1:
                for c in str(count):
                    # 写数字
                    walker += 1
                    chars[walker] = c

            runner += 1
            walker += 1

        return walker
```

```python
#
# @lc app=leetcode.cn id=448 lang=python3
#
# [448] 找到所有数组中消失的数字
#
class Solution:
    def findDisappearedNumbers(self, nums: List[int]) -> List[int]:
        '''
        # time Limit Exceeded
        res = []
        leng = len(nums)
        for i in range(leng):
            if i+1 not in nums:
                res.append(i+1)
        return res
        '''
        for num in nums:
            index = abs(num)-1
            if nums[index]>0:
                nums[index] *= -1

        res = []
        for i in range(len(nums)):
            if nums[i] > 0:
                res.append(i+1)
        return res
```

```python
#
# @lc app=leetcode.cn id=474 lang=python3
#
# [474] 一和零
#
class Solution:
    def findMaxForm(self, strs: List[str], m: int, n: int) -> int:
        if not strs:
            return 0
        # 准备很多个背包
        dp = [ [0]*(n+1) for _ in range(m+1)]

        for str in strs:
            count0 = str.count('0')
            count1 = str.count('1')

            # 遍历可容纳的背包
            for zeroes in range(m, count0 - 1, -1):
                for ones in range(n, count1 - 1, -1):
```

```python
                            dp[zeroes][ones] = max(dp[zeroes][ones] ,
                                1 + dp[zeroes - count0][ones - count1])
            return dp[m][n]
```

```python
#
# @lc app=leetcode.cn id=485 lang=python3
#
# [485] 最大连续1的个数
#
class Solution:
    def findMaxConsecutiveOnes(self, nums: List[int]) -> int:
        maxval = 0
        tmp = 0
        for i in range(len(nums)):
            if nums[i] != 0 :
                tmp += 1
            else :
                maxval = max(maxval,tmp)
                tmp = 0
        maxval = max(maxval,tmp)
        return maxval
```

```python
#
# @lc app=leetcode.cn id=494 lang=python3
#
# [494] 目标和
#
class Solution:
    def findTargetSumWays(self, nums: List[int], S: int) -> int:
        sum_nums = sum(nums)
        if sum_nums < S or (S + sum_nums)%2 != 0:
            return 0

        target = (S + sum_nums) // 2
        dp = [0]*(target + 1)
        dp[0] = 1
        for num in nums:
            for i in range(target, num-1, -1):
                dp[i] += dp[i - num]
        return dp[-1]
```

```python
#
# @lc app=leetcode.cn id=532 lang=python3
#
# [532] 数组中的K-diff数对
#
class Solution:
```

```python
    def findPairs(self, nums: List[int], k: int) -> int:
        dic = {}
        if k < 0:
            return 0
        res = 0
        for num in nums:
            dic[num] = dic.get(num,0) + 1
        for num in nums:
            # 值在里面 且 k 不为0
            if dic.get(num-k,0) > 0 and k != 0:
                res += 1
                dic[num-k] = 0
            # k 为0,值有多个
            elif k == 0 and dic.get(num,0) > 1:
                res += 1
                dic[num-k] = 0
        return res
```

```python
#
# @lc app=leetcode.cn id=541 lang=python3
#
# [541] 反转字符串 II
#
class Solution:
    def reverseStr(self, s: str, k: int) -> str:
        if len(s)<k:
            return s[::-1]
        if len(s)<2*k:
            return s[:k][::-1]+s[k:]
        return s[:k][::-1]+s[k:2*k] + self.reverseStr(s[2*k:], k)
```

```python
#
# @lc app=leetcode.cn id=547 lang=python3
#
# [547] 朋友圈
#
class Solution:
    def findCircleNum(self, M: List[List[int]]) -> int:
        # 遍历每个人,遍历到过置1
        visited = [0 for _ in range(len(M))]
        # 圈数
        count = 0
        for i in range(len(M)):
            # 等于1表示被别的圈包进去了,等于0表示再开一个圈
            if visited[i] == 0:
                self.dfs(M, visited, i)
```

```
16                    count += 1
17            return count
18
19        # 判断和i认识的都是哪些人
20        def dfs( self , M, visited , i ):
21            # 全1了
22            if sum(visited) == len(M):
23                return
24            for j in range(len(M)):
25                if j != i and visited [j] == 0 and M[i][j] == 1 :
26                    visited [j] = 1
27                    self .dfs(M, visited , j)
```

```
1   #
2   # @lc app=leetcode.cn id=551 lang=python3
3   #
4   # [551] 学生出勤记录 I
5   #
6   class Solution:
7       def checkRecord(self, s: str ) −> bool:
8           count = 0
9           for i in range(len(s)):
10              if s[i] =='A':
11                  # 大于1个A
12                  count += 1
13                  if count > 1:
14                      return False
15              elif s[i] == 'L' and 0 < i < len(s)−1 \
16                  and s[i−1] == 'L' == s[i+1]:
17                  return False
18          return True
```

```
1   #
2   # @lc app=leetcode.cn id=557 lang=python3
3   #
4   # [557] 反转字符串中的单词 III
5   #
6   class Solution:
7       def reverseWords(self, s: str ) −> str:
8           return '␣'.join ([word[::−1] for word in s. split ('␣')])
```

```
1   #
2   # @lc app=leetcode.cn id=561 lang=python3
3   #
4   # [561] 数组拆分 I
5   #
6   class Solution:
```

```python
    def arrayPairSum(self, nums: List[int]) -> int:
        nums.sort()
        return sum(nums[::2])
```

```python
#
# @lc app=leetcode.cn id=566 lang=python3
#
# [566] 重塑矩阵
#
class Solution:
    def matrixReshape(self, nums: List[List[int]], r: int, c: int) -> List[List[int]]:
        row = len(nums)
        col = len(nums[0])
        if row * col != r*c:
            return nums
        res = [[]]
        for i in range(row):
            for j in range(col):
                k = nums[i][j]
                if len(res[-1]) < c:
                    res[-1].append(k)
                else:
                    res.append([k])
        return res
```

```python
#
# @lc app=leetcode.cn id=575 lang=python3
#
# [575] 分糖果
#
class Solution:
    def distributeCandies(self, candies: List[int]) -> int:
        return int(min(len(set(candies)), len(candies)//2))
```

```python
#
# @lc app=leetcode.cn id=581 lang=python3
#
# [581] 最短无序连续子数组
#
class Solution:
    def findUnsortedSubarray(self, nums: List[int]) -> int:
        num_sort = nums[:] # 浅拷贝和深拷贝
        num_sort.sort()
        n=len(nums)
        i,j=0,n-1
        while i<n and nums[i]==num_sort[i]:
            i += 1
```

```
14            while j>i+1 and nums[j]==num_sort[j]:
15                j -= 1
16            return j-i+1
```

```
1  #
2  # @lc app=leetcode.cn id=605 lang=python3
3  #
4  # [605] 种花问题
5  #
6  class Solution:
7      def canPlaceFlowers(self, flowerbed: List[int], n: int) -> bool:
8          # 前后补零解决边界问题
9          nums=[0]+flowerbed+[0]
10         cnt=0
11         i=1
12         while i<len(flowerbed)+1:
13             if nums[i-1]==0 and nums[i]==0 and nums[i+1]==0:
14                 cnt += 1
15                 # 可以种花，则需要间隔一个位置，所以+2
16                 i += 2
17             else:
18                 i+=1
19         return cnt>=n
```

```
1  #
2  # @lc app=leetcode.cn id=628 lang=python3
3  #
4  # [628] 三个数的最大乘积
5  #
6  class Solution:
7      def maximumProduct(self, nums: List[int]) -> int:
8          nums.sort()
9          res1 = nums[-1]*nums[-2]*nums[-3]
10         res2 = nums[-1]*nums[0]*nums[1]
11         return max(res1,res2)
```

```
1  #
2  # @lc app=leetcode.cn id=643 lang=python3
3  #
4  # [643] 子数组最大平均数 I
5  #
6  class Solution:
7      def findMaxAverage(self, nums: List[int], k: int) -> float:
8          tmp = maxmean = sum(nums[:k])
9          for i in range(k,len(nums)):
10             tmp += (nums[i]-nums[i-k])
11             maxmean = max(maxmean,tmp)
```

```python
            return maxmean/k
```

```python
#
# @lc app=leetcode.cn id=661 lang=python3
#
# [661] 图片平滑器
#
class Solution:
    def imageSmoother(self, M: List[List[int]]) -> List[List[int]]:
        R, C = len(M), len(M[0])
        res = [[0] * C for _ in range(R)]

        for r in range(R):
            for c in range(C):
                count = 0
                for nr in (r-1, r, r+1):
                    for nc in (c-1, c, c+1):
                        if 0 <= nr < R and 0 <= nc < C:
                            res[r][c] += M[nr][nc]
                            count += 1
                res[r][c] //= count
        return res
```

```python
#
# @lc app=leetcode.cn id=665 lang=python3
#
# [665] 非递减数列
#
class Solution:
    def checkPossibility(self, nums: List[int]) -> bool:
        count = 0
        for i in range(len(nums)-1):
            if nums[i]>nums[i+1]:
                count +=1
                #变相去掉nums[i]
                if i <1 or nums[i-1] <= nums[i+1]:
                    nums[i] = nums[i+1]
                else:
                    # 变相去掉nums[i+1]
                    nums[i+1]=nums[i]
        return count <= 1
```

```python
#
# @lc app=leetcode.cn id=674 lang=python3
#
# [674] 最长连续递增序列
#
```

```python
class Solution:
    def findLengthOfLCIS(self, nums: List[int]) -> int:
        if not nums:
            return 0
        count = 1
        res = 0
        for i in range(len(nums)-1):
            if nums[i] < nums[i+1]:
                count += 1
            else:
                res = max(res,count)
                count = 1
        return max(res,count)
```

```python
#
# @lc app=leetcode.cn id=680 lang=python3
#
# [680] 验证回文字符串
#
class Solution:
    def validPalindrome(self, s: str) -> bool:
        count = 0
        for i in range(len(s)//2):
            if s[i] != s[-1-i]:
                t, u = s[:i]+s[i+1:], s[:-1-i]+s[len(s)-i:]
                return t == t[::-1] or u == u[::-1]
        return True
```

```python
#
# @lc app=leetcode.cn id=695 lang=python3
#
# [695] 岛屿的最大面积
#
class Solution:
    def maxAreaOfIsland(self, grid: List[List[int]]) -> int:
        res = 0
        for i in range(len(grid)):
            for j in range(len(grid[0])):
                if grid[i][j] == 1:
                    temp = self.dfs(grid, i, j)
                    res = max(res,temp)
        return res

    def dfs(self, grid, i, j):
        # 终止条件
        if i < 0 or j < 0 or i >= len(grid) or j >= len(grid[0]) or grid[i][j] == 0:
```

```
19          return 0
20
21      # 四个方向搜索
22      grid[i][j] = 0
23      res = 1
24      res += self.dfs(grid, i-1, j)
25      res += self.dfs(grid, i, j-1)
26      res += self.dfs(grid, i+1, j)
27      res += self.dfs(grid, i, j+1)
28
29      return res
```

```python
1  #
2  # @lc app=leetcode.cn id=836 lang=python3
3  #
4  # [836] 矩形重叠
5  #
6  class Solution:
7      def isRectangleOverlap(self, rec1: List[int], rec2: List[int]) -> bool:
8          return not (rec1[2] <= rec2[0] or  # rec1的右边在rec2的左边
9                      rec1[3] <= rec2[1] or  # rec1的上边在rec2的下边
10                     rec1[0] >= rec2[2] or  # rec1的左边在rec2的右边
11                     rec1[1] >= rec2[3])    # rec1的下边在rec2的上边
```