LeetCode 题解 (Python 版本)

胡欣毅 (icedomain_hu@qq.com)

最后更新 March 4, 2020

```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:
23                 v2 = l2.val
```

```python
24                l2 = l2.next
25            # 除数、余数
26            val = (v1+v2+jingwei) % 10
27            jingwei = (v1+v2+jingwei) // 10
28            n.next = ListNode(val)
29            # 指向下一个
30            n = n.next
31        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:
            return nums2[k]
```

```python
        if not nums2:
            return nums1[k]
        l1 , l2 = len(nums1)//2,len(nums2)//2
        val1 ,val2 = nums1[l1],nums2[l2]

        if l1+l2<k:# 往右找
            if val1 > val2:
                return self.findk(nums1, nums2[l2 + 1:], k - l2 - 1)
            else:
                return self.findk(nums1[l1 + 1:],nums2, k - l1 - 1)
        else: # 往左找
            if val1 > val2:
                return self.findk(nums1[:l1],nums2, k)

            else:
                return self.findk(nums1, nums2[:l2], k)
```

```python
#
# @lc app=leetcode.cn id=5 lang=python3
#
# [5] 最长回文子串
#
class Solution:
    def longestPalindrome(self, s: str) -> str:
        if s is None:
            return None

        # 动态规划
        dp = [[0 for _ in range(len(s))] for _ in range(len(s))]
        left , right , max_len = 0, 0, 0

        for j in range(len(s)):
            # 对角线置1
            dp[j][j] = 1
            for i in range(j):
                if s[i] == s[j] and (j-i < 2 or dp[i+1][j-1]):
                    dp[i][j] = 1
                if dp[i][j] and max_len < j-i+1:
                    max_len = j - i + 1
                    left , right = i, j
        return s[left:right+1]
```

```python
#
# @lc app=leetcode.cn id=6 lang=python3
#
# [6] Z 字形变换
```

```python
#
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
        sign = 1
```

```python
        if str[0] == '+' or str[0] == '-':
            if str[0] == '-':
                sign = -1
            str = str[1:]
        res = 0
        for char in str:
            if char >= '0' and char <= '9':
                res = res * 10 + ord(char) - ord('0')
            if char < '0' or char > '9':
                break
        return max(-2**31, min(sign * res, 2**31-1))
```

```python
#
# @lc app=leetcode.cn id=9 lang=python3
#
# [9] 回文数
#
class Solution:
    def isPalindrome(self, x: int) -> bool:
        if x < 0:
            return False
        # 最高位的位数
        d = 1
        while x // d >= 10:
            d *= 10
        while x > 0:
            # p q 对应最高位和最低位
            p = x // d
            q = x % 10
            if p != q:
                return False
            # x 去掉最高位,去掉最低位
            x = x % d // 10
            # x 去掉了两位,d也减两位
            d //= 100
        return True
```

```python
#
# @lc app=leetcode.cn id=10 lang=python3
#
# [10] 正则表达式匹配
#
class Solution:
    def isMatch(self, s: str, p: str) -> bool:
        '''
        # 递归写法
```

```python
10          # s已被匹配且p已耗完
11          if not s and not p:
12              return True
13          # p已耗完但s未被完全匹配
14          if len(s) > 0 and len(p) == 0:
15              return False
16
17          # 如果模式第二个字符是*
18          if len(p) > 1 and p[1] == '*':
19              if len(s) > 0 and (s[0] == p[0] or p[0] == '.'): # ax a* or ax .*
20                  # 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
21                  return self.isMatch(s, p[2:]) or self.isMatch(s[1:], p)
22              else:
23                  # 如果第一个字符不匹配, p往后移2位, 相当于忽略x*
24                  return self.isMatch(s, p[2:])
25          # 如果模式第二个字符不是*
26          if len(s) > 0 and (s[0] == p[0] or p[0] == '.'):
27              return self.isMatch(s[1:], p[1:])
28          else:
29              return False
30          '''
31          # 动态规划
32          # 初始化dp表, 初始化表的第一列和第一行
33          # p对应列 s对应行
34          dp = [[False for j in range(len(p) + 1)] for i in range(len(s) + 1)]
35          dp[0][0] = True # s 和 p 都为空时
36          # 若 s 为空时
37          # 处理第一行
38          # p 与 dp 有一位的错位(多了一个空导致的)
39          for j in range(1, len(p) + 1):
40              # dp[0][j]= (p[j−1]=="*")and(j>=2)and(dp[0][j−2])
41              # 等同于下列语句
42              if p[j − 1] == '*':
43                  if j >= 2:
44                      dp[0][j] = dp[0][j − 2]
45          # 第一列就第一个是 True,下面都是 False
46          # 不用处理 pass
47
48          for i in range(1, len(s) + 1):
49              for j in range(1, len(p) + 1):
50                  # j−1才为正常字符串中的索引
51                  # p当前位置为"*"时
52                  # 代表空串−−dp[i][j−2]
53                  # 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]=='.')
54                  if p[j − 1] == '*':
55                      dp[i][j] = dp[i][j − 2] or (
```

```python
                                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)
                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])
```

```python
        for i in range(len(nums)-2):
            l,r = i+1,len(nums)-1
            while l < r:
                sum_val = nums[i]+nums[l]+nums[r]
                if sum_val == target:
                    return sum_val
                if abs(res-target)>abs(sum_val-target):
                    res = sum_val
                if sum_val < target:
                    l+=1
                else:
                    r -= 1
        return res
```

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

```python
#
# @lc app=leetcode.cn id=18 lang=python3
#
# [18] 四数之和
#
class Solution:
```

```python
 7      def fourSum(self, nums: List[int], target: int) -> List[List[int]]:
 8          res = []
 9          # 去除异常
10          if not nums or len(nums) < 4:
11              return res
12          nums.sort()
13          # 第一个数遍历
14          for i in range(len(nums) - 3):
15              if i > 0 and nums[i] == nums[i - 1]:
16                  continue
17              # 第二个数遍历
18              for j in range(i + 1, len(nums) - 2):
19                  if j > i + 1 and nums[j] == nums[j - 1]:
20                      continue
21                  # 双指针
22                  L, R = j + 1, len(nums) - 1
23                  while L < R:
24                      if nums[i] + nums[j] + nums[L] + nums[R] == target:
25                          res.append([nums[i], nums[j], nums[L], nums[R]])
26                          while L < R and nums[L] == nums[L + 1]:
27                              L += 1
28                          while L < R and nums[R] == nums[R - 1]:
29                              R -= 1
30                          L += 1
31                          R -= 1
32                      elif nums[i] + nums[j] + nums[L] + nums[R] < target:
33                          L += 1
34                      else:
35                          R -= 1
36          return res
```

```python
 1  #
 2  # @lc app=leetcode.cn id=19 lang=python3
 3  #
 4  # [19] 删除链表的倒数第N个节点
 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 removeNthFromEnd(self, head: ListNode, n: int) -> ListNode:
14          if head is None:
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
```

```
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
```

```
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 .
7   # class ListNode:
```

```python
 8  #     def __init__(self, x):
 9  #         self.val = x
10  #         self.next = None
11
12  class Solution:
13      def mergeTwoLists(self, l1: ListNode, l2: ListNode) -> ListNode:
14          dummy = now = ListNode(-1)
15          while l1 and l2:
16              if l1.val <= l2.val:
17                  now.next = l1
18                  l1 = l1.next
19              else:
20                  now.next = l2
21                  l2 = l2.next
22              now = now.next
23          now.next = l1 or l2
24          return dummy.next
```

```python
 1  #
 2  # @lc app=leetcode.cn id=22 lang=python3
 3  #
 4  # [22] 括号生成
 5  #
 6  class Solution:
 7      def generateParenthesis(self, n: int) -> List[str]:
 8          res = []
 9          if n > 0:
10              self.dfs(n,'',res,0,0)
11          return res
12
13      def dfs(self,n,path,res, left, right):
14          # 终止条件
15          if len(path) == 2 * n:
16              res.append(path)
17              return
18          # 左括号(够了没
19          if left < n:
20              self.dfs(n,path+'(',res, left+1, right)
21          # 右括号补成和左括号一样多
22          if left > right:
23              self.dfs(n,path+')',res, left, right+1)
```

```python
 1  #
 2  # @lc app=leetcode.cn id=23 lang=python3
 3  #
 4  # [23] 合并K个排序链表
```

```
 5  #
 6  # Definition for singly-linked list.
 7  # class ListNode:
 8  #     def __init__(self, x):
 9  #         self.val = x
10  #         self.next = None

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

18      def mergeK(self, lists, low, high):
19          if low == high:
20              return lists[low]
21          elif low + 1 == high:
22              return self.mergeTwolists(lists[low], lists[high])
23          mid = (low + high) // 2
24          return self.mergeTwolists(self.mergeK(lists, low, mid), self.mergeK(lists, mid + 1, high))

26      def mergeTwolists(self, l1, l2):
27          if l1 is None:
28              return l2
29          if l2 is None:
30              return l1
31          head = curr = ListNode(-1)
32          while l1 and l2:
33              if l1.val <= l2.val:
34                  curr.next = l1
35                  l1 = l1.next
36              else:
37                  curr.next = l2
38                  l2 = l2.next
39              curr = curr.next
40          curr.next = l1 or l2
41          return head.next
```

```
 1  #
 2  # @lc app=leetcode.cn id=24 lang=python3
 3  #
 4  # [24] 两两交换链表中的节点
 5  #
 6  # Definition for singly-linked list.
 7  # class ListNode:
 8  #     def __init__(self, x):
```

```python
#           self.val = x
#           self.next = None

class Solution:
    def swapPairs(self, head: ListNode) -> ListNode:
        prev = dummy = ListNode(-1)
        dummy.next = head
        while prev.next and prev.next.next:
            # prev a b -> prev b a (交换a,b)
            a = prev.next
            b = prev.next.next
            prev.next, b.next, a.next = b, a, b.next
            prev = a
        return dummy.next
```

```python
#
# @lc app=leetcode.cn id=25 lang=python3
#
# [25] K 个一组翻转链表
#
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution:
    def reverseKGroup(self, head: ListNode, k: int) -> ListNode:
        if head is None or k < 2:
            return head
        dummy = ListNode(0)
        dummy.next = head
        start = dummy
        end = start.next

        count = 0
        while end:
            count += 1
            if count % k == 0:
                # 返回为新一轮的头
                start = self.reverse(start, end.next)
                end = start.next
            else:
                end = end.next
        return dummy.next
```

```
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
```

```
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)
```

```
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
```

```
1   #
2   # @lc app=leetcode.cn id=28 lang=python3
3   #
```

```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)

        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)
```

```
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
20                 else :
```

```
21              l = mid +1
22        # l 在左半段 、 mid 在后半段
23        else :
24            if nums[mid] <= target <= nums[r]:
25                l = mid +1
26            else :
27                r = mid −1
28    return −1
```

```
#
# @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]
```

```
#
# @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:
        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
#
# @lc app=leetcode.cn id=36 lang=python3
#
# [36] 有效的数独
#
class Solution:
    def isValidSudoku(self, board: List[List[str]]) -> bool:
        return (self.is_row_valid(board) and
                self.is_col_valid(board) and
                self.is_square_valid(board))

    def is_row_valid(self, board):
        for row in board:
            if not self.is_unit_valid(row):
                return False
        return True

    def is_col_valid(self, board):
        # 列转化成行
        for col in zip(*board):
            if not self.is_unit_valid(col):
                return False
        return True

    def is_square_valid(self, board):
        for i in (0, 3, 6):
            for j in (0, 3, 6):
                square = [board[x][y] for x in range(i, i + 3) for y in range(j, j + 3)]
                if not self.is_unit_valid(square):
                    return False
        return True

    def is_unit_valid(self, unit):
        unit = [i for i in unit if i != '.']
        return len(set(unit)) == len(unit)
```

```python
#
# @lc app=leetcode.cn id=37 lang=python3
#
# [37] 解数独
#
class Solution:
    def solveSudoku(self, board: List[List[str]]) -> None:
        self.dfs(board)

    def dfs(self, board):
        for i in range(9):
            for j in range(9):
                if board[i][j] == '.':
                    for k in '123456789':
                        board[i][j] = k
                        # 修改一个值判断是不是合法的
                        # 如果这个递归可以返回true并且当前填入的数字也没毛病
                        # 则证明我们解完了数独
                        if self.isOK(board,i,j) and self.dfs(board):
                            return True
                        board[i][j] = '.'
                    return False
        # 全部填完之后返回True
        return True

    def isOK(self,board,x,y):
        #列符合
        for i in range(9):
            if i != x and board[i][y] == board[x][y]:
                return False
        #检查行是否符合
        for j in range(9):
            if j != y and board[x][j] == board[x][y]:
                return False
        row_start = 3*(x // 3)
        col_start = 3*(y // 3)
        for i in range(row_start,row_start+3):
            for j in range(col_start,col_start+3):
                if (i!= x or j!= y) and board[i][j] == board[x][y]:
                    return False
        return True
```

```python
#
# @lc app=leetcode.cn id=38 lang=python3
#
# [38] 外观数列
```

```python
#
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
#
# @lc app=leetcode.cn id=40 lang=python3
#
```

```python
# [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
        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)):
                arr[i+j] += int(num1[i]) * int(num2[j])
```

```
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)
```

```
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
```

```
1  #
2  # @lc app=leetcode.cn id=46 lang=python3
3  #
4  # [46] 全排列
```

```
5   #
6   class Solution:
7       def permute(self, nums: List[int]) -> List[List[int]]:
8           #nums.sort()
9           res = []
10          self.dfs(nums, [], res)
11          return res
12
13      def dfs(self, nums, path, res):
14          if not nums:
15              # nums已经全部压入到path里面了
16              res.append(path)
17              return
18          for i in range(len(nums)):
19              self.dfs(nums[:i]+nums[i+1:], path+[nums[i]], res)
```

```
1   #
2   # @lc app=leetcode.cn id=47 lang=python3
3   #
4   # [47] 全排列 II
5   #
6   class Solution:
7       def permuteUnique(self, nums: List[int]) -> List[List[int]]:
8           res = []
9           self.dfs(nums, [], res)
10          return res
11
12      def dfs(self, nums, path, res):
13          if not nums and path not in res:
14              # nums已经全部压入到path里面了
15              res.append(path)
16              return
17          for i in range(len(nums)):
18              self.dfs(nums[:i]+nums[i+1:], path+[nums[i]], res)
```

```
1   #
2   # @lc app=leetcode.cn id=48 lang=python3
3   #
4   # [48] 旋转图像
5   #
6   class Solution:
7       def rotate(self, matrix: List[List[int]]) -> None:
8           if matrix is None or len(matrix) == 1:
9               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)
```

```python
#
# @lc app=leetcode.cn id=51 lang=python3
#
# [51] N皇后
#
class Solution:
    def solveNQueens(self, n: int) -> List[List[str]]:
        result = []
        # C[i]表示第i行皇后在哪一列
        C = [-1 for _ in range(n)]
        self.dfs(C,result,0)
        return result

    def dfs(self,C,res,row):
        N = len(C)
        # 终止条件
        if N == row :
            path = [["." for _ in range(N)] for _ in range(N)]
            for i in range(N):
                # (i,C[i])位置对应皇后
                path[i][C[i]] = "Q"
            path = ["".join(r) for r in path]
            # if path not in res:
            # 不用排除
            res.append(path)
            return
        # 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,res,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=52 lang=python3
#
# [52] N皇后 II
```

```python
#
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)
        return maxsum
```

```python
#
```

```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
        '''

        m,n = len(matrix),len(matrix[0])
        x = y = di = 0
        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
```

```
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
```

```
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
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
```

```
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
```

```
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])
```

```
1   #
2   # @lc app=leetcode.cn id=59 lang=python3
3   #
```

```
# [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)!
        factorials = [1]
        for i in range(1, n):
            factorials.append(factorials[i - 1] * i)
```

```
14
15          # 第几个转化为第几个的索引(减1)
16          k -= 1
17
18          res = []
19          for i in range(n - 1, -1, -1):
20              # 计算第几个区间,首位所在的区间 k//(n-1)!
21              # 第一个区间首位是1,第二个区间首位是2
22              idx = k // factorials[i]
23              # 减去多个区间对应的值
24              k -= idx * factorials[i]
25              # 结果值添加对应的数字
26              res.append(nums[idx])
27              # 因为排列不重复,nums需要去掉对应元素
28              nums.pop(idx)
29
30          return ''.join(res)
```

```
1   #
2   # @lc app=leetcode.cn id=61 lang=python3
3   #
4   # [61] 旋转链表
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 rotateRight(self, head: ListNode, k: int) -> ListNode:
14          if head is None or k == 0:
15              return head
16
17          pointer = head
18          length = 1
19          while pointer.next:
20              pointer = pointer.next
21              length += 1
22
23          # 左部分多少个
24          k = length - k%length
25
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]*n]*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[m-1][n-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])
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[r−1][c−1]
```

```
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]
```

```
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          num = 0
9          for i in range(len(digits)):
10             num += digits[i] * pow(10, len(digits) − i −1 )
11         num = num + 1
12         out = []
13         while num > 0:
14             out.append(num%10)
15             num //= 10
16         out.reverse()
17         return out
```

```
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:
```

```python
        if len(a)==0:
            return b
        if len(b)==0:
            return a
        # 最后都是1 前面的相加 再加1 补0
        if a[-1] == '1' and b[-1] == '1':
            return self.addBinary(self.addBinary(a[0:-1],b[0:-1]),'1')+'0'
        # 最后都是0 补0
        if a[-1] == '0' and b[-1] == '0':
            return self.addBinary(a[0:-1],b[0:-1])+'0'
        # 最后一个1 一个0 补1
        else:
            return self.addBinary(a[0:-1],b[0:-1])+'1'
```

```python
#
# @lc app=leetcode.cn id=69 lang=python
#
# [69] x 的平方根
#
class Solution(object):
    def mySqrt(self, x):
        """
        :type x: int
        :rtype: int
        """
        l = 0
        r = x
        while l <= r:
            mid = (l+r)//2
            if mid**2 <= x < (mid+1)**2:
                return mid
            elif x < mid**2:
                r = mid
            else:
                l = mid+1
```

```python
#
# @lc app=leetcode.cn id=70 lang=python3
#
# [70] 爬楼梯
#
class Solution:
    def climbStairs(self, n: int) -> int:
        if n == 1:
            return 1
        # 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)]
10          for i in range(l1):
11              dp[i][0] = i
12          for j in range(l2):
13              dp[0][j] = j
14          for i in range(1, l1):
15              for j in range(1, l2):
16                  if word1[i-1]==word2[j-1]:
17                      dp[i][j] = dp[i-1][j-1]
18                  else:
19                      # 三个分别对应于加、减、替换
20                      dp[i][j] = min(dp[i-1][j],
21                                     dp[i][j-1],
22                                     dp[i-1][j-1]
23                                     )+1
```

```
24        return dp[−1][−1]
```

```
1    #
2    # @lc app=leetcode.cn id=73 lang=python3
3    #
4    # [73] 矩阵置零
5    #
6    class Solution:
7        def setZeroes( self , matrix: List [ List [ int ]])  −> None:
8            """"
9            Do not return anything, modify matrix in−place instead.
10           """"
11           '''
12           # 直接法
13           row = []
14           col = []
15           hang = len(matrix)
16           lie  = len(matrix[0])
17           for  i  in  range(hang):
18               for  j  in  range( lie ):
19                   if  matrix[i ][ j ]  == 0:
20                       row.append(i)
21                       col .append(j)
22           row = set(row)
23           col  = set(col)
24
25           for  i  in  row:
26               for  j  in  range( lie ):
27                   matrix[i ][ j ]  = 0
28           for  j  in  col :
29               for  i  in  range(hang):
30                   matrix[i ][ j ]  =0
31
32           return matrix
33           '''
34           firstRowHasZero = not all(matrix[0])
35           hang = len(matrix)
36           lie  = len(matrix[0])
37           # 第一行第一列做标记
38           for  i  in  range(1,hang):
39               for  j  in  range( lie ):
40                   if  matrix[i ][ j ]  == 0:
41                       matrix [0][ j ]  = matrix[i ][0]  = 0
42           # 置0
43           for  i  in  range(1,hang):
44               for  j  in  range( lie −1,−1,−1):
```

```
45              if matrix[i][0] == 0 or matrix[0][j] == 0:
46                  matrix[i][j] = 0
47          # 补一下第一行的
48
49          if firstRowHasZero:
50              matrix[0] = [0] * lie
51
52          return matrix
```

```
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
```

```
1   #
2   # @lc app=leetcode.cn id=75 lang=python3
3   #
4   # [75] 颜色分类
5   #
6   class Solution:
7       def sortColors( self , nums: List[int]) -> None:
8           """
9           Do not return anything, modify nums in-place instead.
10          """
11          count = [0 ,0,0]
12          for num in nums:
13              count[num] += 1
14          idx = 0
15          for i in range(3):
16              for j in range(count[i]) :
17                  nums[idx] = i
```

```
18              idx += 1
```

```python
#
# @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)
        m = {}
        count = 0
        # 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]

                    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
#
```

```python
# [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
#
class Solution:
    def exist(self, board: List[List[str]], word: str) -> bool:
        m, n = len(board), len(board[0])
        visited = [[False for i in range(n)] for i in range(m)]
        # 遍历寻找开头
        for i in range(m):
            for j in range(n):
                if self.dfs(board,word,visited,i,j,0):
                    return True
        return False
```

```python
    def dfs(self,board,word,visited,i,j,start):
        # 终止条件
        if start == len(word):
            return True
        # 溢出 剪枝 or 已经访问过了
        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]:
            return False

        if board[i][j] == word[start]:
            visited[i][j] = True
            ret = self.dfs(board,word,visited,i+1,j,start+1) or \
                    self.dfs(board,word,visited,i-1,j,start+1) or \
                    self.dfs(board,word,visited,i,j+1,start+1) or \
                    self.dfs(board,word,visited,i,j-1,start+1)
            visited[i][j] = False

            return ret
```

```python
#
# @lc app=leetcode.cn id=82 lang=python3
#
# [82] 删除排序链表中的重复元素 II
#
# 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:
        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
            else:
                prev = prev.next
                head = head.next
```

```
27          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
#
# [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
#
```

```python
# [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)
                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
```

46

```python
                    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

class Solution:
    def partition(self, head: ListNode, x: int) -> ListNode:
        h1 = l1 = ListNode(0)
        h2 = l2 = ListNode(0)

        while head:
            if head.val < x:
                l1.next = head
                l1 = l1.next
            else:
                l2.next = head
                l2 = l2.next
            head = head.next
        l2.next = None
        l1.next = h2.next

        return h1.next
```

```python
#
# @lc app=leetcode.cn id=88 lang=python3
#
```

```python
# [88] 合并两个有序数组
#
class Solution:
    def merge(self, nums1: List[int], m: int, nums2: List[int], n: int) -> None:
        """
        Do not return anything, modify nums1 in-place instead.
        """
        # 从后往前
        p1 = m - 1
        p2 = n - 1
        p = m + n - 1
        # 两个都没放完
        while p1 >= 0 and p2 >= 0:
            if nums1[p1] >= nums2[p2]:
                nums1[p] = nums1[p1]
                p1 -= 1
            else:
                nums1[p] = nums2[p2]
                p2 -= 1
            p -= 1
        # p1没放完，那就不用再操作了
        # p2没放完
        while p2 >= 0:
            nums1[p] = nums2[p2]
            p -= 1
            p2 -= 1
```

```python
#
# @lc app=leetcode.cn id=89 lang=python3
#
# [89] 格雷编码
#

class Solution:
    def grayCode(self, n: int) -> List[int]:
        res = [0]
        for i in range(n):
            for j in range(len(res)-1,-1,-1):
                res.append(res[j] + (1 << i))
        return res
```

```python
#
# @lc app=leetcode.cn id=90 lang=python3
#
# [90] 子集 II
#
```

```
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:
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]
```

```
1   #
2   # @lc app=leetcode.cn id=92 lang=python3
3   #
4   # [92] 反转链表 II
5   #
```

```python
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution:
    def reverseBetween(self, head: ListNode, m: int, n: int) -> ListNode:
        dummy = ListNode(0)
        dummy.next = head
        prev = dummy

        for i in range(m-1):
            prev = prev.next

        temp = None
        cur = prev.next
        for i in range(n-m+1):
            next = cur.next
            # reverse
            cur.next = temp
            temp = cur
            # 下一个
            cur = next

        # 最后面一段
        prev.next.next = cur
        '''
        wi = temp
        while wi.next is not None:
            wi = wi.next
        wi.next = cur
        '''
        # 中间一段
        prev.next = temp

        return dummy.next
```

```python
#
# @lc app=leetcode.cn id=93 lang=python3
#
# [93] 复原IP地址
#
class Solution:
    def restoreIpAddresses(self, s: str) -> List[str]:
        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                 ip.append(substr)
32                 self.dfs(s,ip,res, start + i + 1)
33                 ip.pop()
```

```
 1  #
 2  # @lc app=leetcode.cn id=94 lang=python3
 3  #
 4  # [94] 二叉树的中序遍历
 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 inorderTraversal( self , root: TreeNode) −> List[int]:
15          if root is None:
16              return None
17          result = []
18          stack = []
19          p = root
20          while stack or p:
```

```python
21              # 先把左边的压进去
22              if p:
23                  stack.append(p)
24                  p = p.left
25              else:
26                  p = stack.pop()
27                  result.append(p.val)
28                  p = p.right
29
30          return result
```

```python
1   #
2   # @lc app=leetcode.cn id=95 lang=python3
3   #
4   # [95] 不同的二叉搜索树 II
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 generateTrees(self, n: int) -> List[TreeNode]:
15          if n == 0:
16              return []
17          return self.get_trees(1,n)
18
19      def get_trees(self,start,end):
20          res = []
21          if start > end:
22              # 空子树情况
23              res.append(None)
24              return res
25          for i in range(start,end+1):
26              lefts = self.get_trees(start,i-1)
27              rights = self.get_trees(i+1,end)
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
```

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

```python
#
# @lc app=leetcode.cn id=97 lang=python3
#
# [97] 交错字符串
#
class Solution:
    def isInterleave(self, s1: str, s2: str, s3: str) -> bool:
        l1, l2, l3 = len(s1), len(s2), len(s3)
        if l1+l2 != l3:
            return False

        dp = [[True for _ in range(l2+1)] for _ in range(l1+1)]
        # 边界条件
        # 用s1去填
        for i in range(1, l1+1):
            dp[i][0] = dp[i-1][0] and s1[i-1] == s3[i-1]
        # 用s2去填
        for j in range(1, l2+1):
            dp[0][j] = dp[0][j-1] and s2[j-1] == s3[j-1]

        for i in range(1, l1+1):
            for j in range(1, l2+1):
                dp[i][j] = (dp[i-1][j] and s1[i-1] == s3[i+j-1]) or \
                (dp[i][j-1] and s2[j-1] == s3[i+j-1])

        return dp[l1][l2]
```

```python
#
# @lc app=leetcode.cn id=98 lang=python3
#
# [98] 验证二叉搜索树
#
```

```python
 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          return root.val > low and root.val < upper and self.isOK(root.left,low,root.val) and self.isOK(
                root.right,root.val,upper)
```

```python
 1  #
 2  # @lc app=leetcode.cn id=100 lang=python3
 3  #
 4  # [100] 相同的树
 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 isSameTree(self, p: TreeNode, q: TreeNode) -> bool:
15          if p is None and q is None:
16              return True
17          elif p and q and p.val == q.val:
18              return self.isSameTree(p.left,q.left) and self.isSameTree(p.right, q.right)
19          elif p is not None or q is not None:
20              return False
```

```python
 1  #
 2  # @lc app=leetcode.cn id=101 lang=python3
 3  #
 4  # [101] 对称二叉树
 5  #
 6  # Definition for a binary tree node.
 7  # class TreeNode:
 8  #     def __init__(self, x):
```

```python
 9  #           self.val = x
10  #           self.left = None
11  #           self.right = None
12
13  class Solution:
14      def isSymmetric(self, root: TreeNode) -> bool:
15          if root is None:
16              return True
17          return self.yes(root.left ,root.right)
18
19      def yes(self, left ,right):
20          if left is None and right is None:
21              return True
22          if left and right and left.val == right.val:
23              if self.yes(left.left ,right.right) and self.yes(left.right ,right.left):
24                  return True
25          return False
```

```python
 1  #
 2  # @lc app=leetcode.cn id=102 lang=python3
 3  #
 4  # [102] 二叉树的层次遍历
 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 levelOrder(self, root: TreeNode) -> List[List[int]]:
15          if root is None:
16              return None
17          result = [[]]
18          self.traverse(root,1, result)
19          return result
20
21      def traverse(self ,root, level , result):
22          if root is None:
23              return
24          if level > len(result):
25              result.append([])
26          result[ level −1].append(root.val)
27          self.traverse(root.left , level +1,result)
28          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 root is None:
            return None
        result = [[]]
        self.traverse(root,1, result ,True)
        return result

    def traverse( self ,root, level , result , flag ):
        if root is None:
            return
        if level > len(result):
            result.append([])

        if flag :
            result [ level −1].append(root.val)
        else :
            result [ level −1]. insert (0,root. val)
        self . traverse (root. left , level +1,result, not flag )
        self . traverse (root. right , level +1,result, not flag )
```

```python
#
# @lc app=leetcode.cn id=104 lang=python3
#
# [104] 二叉树的最大深度
#
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
```

```
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:
8   #     def __init__(self, x):
9   #         self.val = x
10  #         self.left = None
11  #         self.right = None
12
13  class Solution:
14      def buildTree(self, preorder: List[int], inorder: List[int]) -> TreeNode:
15          if inorder:
16              # 前序的头就是root
17              # 中序中,root左边就是左子树,右边是右子树
18              idx = inorder.index(preorder.pop(0))
19              root = TreeNode(inorder[idx])
20              # 递归构造子树先left后right
21              root.left = self.buildTree(preorder, inorder[0:idx])
22              root.right = self.buildTree(preorder, inorder[idx+1:])
23              return root
```

```
1   #
2   # @lc app=leetcode.cn id=106 lang=python3
3   #
4   # [106] 从中序与后序遍历序列构造二叉树
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 buildTree(self, inorder: List[int], postorder: List[int]) -> TreeNode:
15          if inorder:
16              # 后序的尾部就是root
17              # 中序中,root值左边就是左子树,右边是右子树
18              idx = inorder.index(postorder.pop())
19              root = TreeNode(inorder[idx])
20              # 递归构造子树先right后left
21              root.right = self.buildTree(inorder[idx+1:],postorder)
22              root.left = self.buildTree(inorder[0:idx],postorder)
23
24              return root
```

```
1   #
2   # @lc app=leetcode.cn id=107 lang=python3
3   #
4   # [107] 二叉树的层次遍历 II
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 levelOrderBottom(self, root: TreeNode) -> List[List[int]]:
15          '''
16          if root is None:
17              return []
18          # use stack
19          stack = [[root]]
20          res = []
21          while len(stack) > 0:
22              # 取出最新装入的list
23              top = stack.pop()
24              # 一直在头部插入以达到倒序
25              res.insert(0, [t.val for t in top])
26              # 向下新一轮扫描
27              temp = []
28              for node in top:
29                  if node.left is not None:
30                      temp.append(node.left)
31                  if node.right is not None:
```

```
32                    temp.append(node.right)
33                if len(temp) > 0:
34                    stack.append(temp)
35            return res
36            '''

38            # 递归法
39            if root is None:
40                return None
41            result = [[]]
42            self.traverse(root,1, result)
43            return reversed(result)

45        def traverse(self,root, level, result):
46            if root is None:
47                return
48            if level > len(result):
49                result.append([])
50            result[level-1].append(root.val)
51            self.traverse(root.left, level+1,result)
52            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

13  class Solution:
14      def sortedArrayToBST(self, nums: List[int]) -> TreeNode:
15          if not nums:
16              return None
17          mid = len(nums)//2

19          root = TreeNode(nums[mid])
20          root.left  = self.sortedArrayToBST(nums[:mid])
21          root.right = self.sortedArrayToBST(nums[mid+1:])

23          return root
```

```python
#
# @lc app=leetcode.cn id=109 lang=python3
#
# [109] 有序链表转换二叉搜索树
#
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None

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

class Solution:
    def sortedListToBST(self, head: ListNode) -> TreeNode:
        '''
        if not head :
            return None
        if not head.next:
            return TreeNode(head.val)

        slow = head
        fast = head.next.next
        while fast and fast.next:
            fast = fast.next.next
            slow = slow.next
        head2 = slow.next
        slow.next = None
        root = TreeNode(head2.val)
        root.left = self.sortedListToBST(head)
        root.right = self.sortedListToBST(head2.next)
        return root
        '''

        if not head :
            return None
        nums = []
        while head:
            nums.append(head.val)
            head = head.next
        return self.sortedArrayToBST(nums)
```

60

```python
    def sortedArrayToBST(self, nums):
        if not nums :
            return None
        mid = len(nums)//2

        root = TreeNode(nums[mid])
        root. left  = self .sortedArrayToBST(nums[:mid])
        root. right  = self .sortedArrayToBST(nums[mid+1:])

        return root
```

```python
#
# @lc app=leetcode.cn id=110 lang=python3
#
# [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
```

```python
10  #          self.left  = None
11  #          self.right  = None
12
13  class Solution:
14      def minDepth(self, root: TreeNode) -> int:
15          if root is None:
16              return 0
17          if root.left is None or root.right is None:
18              return self.minDepth(root.left) + self.minDepth(root.right) + 1
19          return min(self.minDepth(root.left), self.minDepth(root.right)) + 1
```

```python
1   #
2   # @lc app=leetcode.cn id=112 lang=python3
3   #
4   # [112] 路径总和
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 hasPathSum(self, root: TreeNode, sum: int) -> bool:
15          if root is None:
16              return False
17
18          sum -= root.val
19          if sum == 0 and root.left is None and root.right is None:
20              return True
21          left  = self.hasPathSum(root.left,sum)
22          right  = self.hasPathSum(root.right,sum)
23          return left or right
```

```python
1   #
2   # @lc app=leetcode.cn id=113 lang=python3
3   #
4   # [113] 路径总和 II
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 pathSum(self, root: TreeNode, sum: int) −> List[List[int]]:
15          if root is None:
16              return []
17          result = []
18          self.pathSum2(root, sum, [], result)
19          return result
20
21      def pathSum2(self,root,sum,path,result):
22          if root is None:
23              return
24          if root.left is None and root.right is None and sum == root.val:
25              path.append(root.val)
26              result.append(path)
27
28          self.pathSum2(root.left, sum − root.val, path + [root.val], result)
29          self.pathSum2(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          """
16          Do not return anything, modify root in−place instead.
17          """
18          if root is None:
19              return None
20
21          self.flatten(root.left)
22          self.flatten(root.right)
23
24          if root.left is None:
25              return
26          # 左子树插到root和root.right之间
27          p = root.left
```

```python
        # 左子链的最后一个节点
        while p.right:
            p = p.right
        p.right = root.right
        root.right = root.left
        root.left = None
```

```python
#
# @lc app=leetcode.cn id=115 lang=python3
#
# [115] 不同的子序列
#
class Solution:
    def numDistinct(self, s: str, t: str) -> int:
        if s is None or t is None:
            return 0
        ls = len(s)
        lt = len(t)
        dp = [ [0 for _ in range(lt+1) ] for _ in range(ls+1)]

        # init
        # 当母串子串都是0长度时，次数是1
        # 当子串长度为0时，所有次数都是1
        # 当母串长度为0时，所有次数都是0 (默认是0,不用重复了)
        for i in range(ls+1):
            dp[i][0] = 1

        for i in range(1,ls+1):
            for j in range(1,lt+1):
                dp[i][j] = dp[i-1][j]
                if s[i-1] == t[j-1]:
                    dp[i][j] += dp[i - 1][j - 1]

        return dp[-1][-1]
```

```python
#
# @lc app=leetcode.cn id=116 lang=python3
#
# [116] 填充每个节点的下一个右侧节点指针
#
"""
# Definition for a Node.
class Node:
    def __init__(self, val: int = 0, left : 'Node' = None, right: 'Node' = None, next: 'Node' = None):
        self.val = val
        self.left = left
```

```python
           self.right = right
           self.next = next
"""
class Solution:
    def connect(self, root: 'Node') -> 'Node':
        if root is None or root.left is None:
            return root
        # 左右链接
        root.left.next = root.right
        if root.next:
            root.right.next = root.next.left
        else:
            root.right.next = None

        self.connect(root.left)
        self.connect(root.right)

        return root
```

```python
#
# @lc app=leetcode.cn id=117 lang=python3
#
# [117] 填充每个节点的下一个右侧节点指针 II
#
"""
# Definition for a Node.
class Node:
    def __init__(self, val: int = 0, left: 'Node' = None, right: 'Node' = None, next: 'Node' = None):
        self.val = val
        self.left = left
        self.right = right
        self.next = next
"""
class Solution:
    def connect(self, root: 'Node') -> 'Node':
        head = root
        dummyHead = Node(-1)
        pre = dummyHead
        # dummyHead 当前行的最左端节点
        while root:
            if root.left:
                pre.next = root.left
                pre = pre.next
            if root.right:
                pre.next = root.right
                pre = pre.next
```

```
28          root = root.next
29          # 行的尾部
30          if root is None:
31              # pre值新的
32              pre = dummyHead
33              # dummyHead.next为前面pre.next 第一次赋值的节点
34              root = dummyHead.next
35              #前面链接断开,开始新的一行
36              dummyHead.next = None
37      return head
```

```
1  #
2  # @lc app=leetcode.cn id=118 lang=python3
3  #
4  # [118] 杨辉三角
5  #
6  class Solution:
7      def generate(self, numRows: int) -> List[List[int]]:
8          # 全部都用1先填充
9          out = [[1]*(i+1) for i in range(numRows)]
10         for r in range(numRows):
11             for col in range(1,r):
12                 out[r][col] = out[r-1][col-1] + out[r-1][col]
13         return out
```

```
1  #
2  # @lc app=leetcode.cn id=119 lang=python3
3  #
4  # [119] 杨辉三角 II
5  #
6  class Solution:
7      def getRow(self, rowIndex: int) -> List[int]:
8          '''
9          if rowIndex == 0:
10             return [1]
11         rowIndex += 1
12         # 全部都用1先填充
13         out = [[1]*(i+1) for i in range(rowIndex)]
14         for r in range(rowIndex):
15             for col in range(1,r):
16                 out[r][col] = out[r-1][col-1] + out[r-1][col]
17         return out[-1]
18         '''
19         # 先用1填充
20         res = [1]*(rowIndex+1)
21         # 从后往前,从上往下覆盖
```

```
22        for r in range(2,rowIndex+1):
23            for col in range(r−1, 0  ,−1):# 逆序
24                res[col] += res[col−1]
25        return res
```

```
1  #
2  # @lc app=leetcode.cn id=120 lang=python3
3  #
4  # [120] 三角形最小路径和
5  #
6  class Solution:
7      def minimumTotal(self, triangle: List[List[int]]) −> int:
8          if not triangle:
9              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):
```

68

```
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
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
17          def maxend(root):
18              if root is None:
19                  return 0
20              left = maxend(root.left)
21              right = maxend(root.right)
22              self.res = max(self.res, left + root.val + right)
23              return max(root.val + max(left, right), 0)
24
25          maxend(root)
26          return self.res
```

```
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
```

```
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  #
2  # @lc app=leetcode.cn id=128 lang=python3
3  #
4  # [128] 最长连续序列
5  #
6
7  class Solution:
8      def longestConsecutive(self, nums: List[int]) -> int:
9          maxLen = 0
10         while nums:
11             n = nums.pop()
12             # 往大处搜索
13             i1 = n + 1
14             while i1 in nums:
15                 nums.remove(i1)
16                 i1 += 1
17             # 往小处搜索
18             i2 = n - 1
19             while i2 in nums:
20                 nums.remove(i2)
21                 i2 -= 1
22             maxLen = max(maxLen, i1 - i2 - 1)
23         return maxLen
```

```
1  #
2  # @lc app=leetcode.cn id=129 lang=python3
3  #
4  # [129] 求根到叶子节点数字之和
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 sumNumbers(self, root: TreeNode) -> int:
15          return self.sum_tree(root,0)
16
17      def sum_tree(self,root,sum):
18          if root is None:
19              return 0
20          if root.left is None and root.right is None:
21              return sum*10+root.val
22
23          return self.sum_tree(root.left,sum*10+root.val)+ self.sum_tree(root.right,sum*10+root.val)
```

```
1   #
2   # @lc app=leetcode.cn id=130 lang=python3
3   #
4   # [130] 被围绕的区域
5   #
6   class Solution:
7       def solve(self, board: List[List[str]]) -> None:
8           """
9           Do not return anything, modify board in-place instead.
10          """
11          if len(board) <= 2 or len(board[0])<=2:
12              return
13          row , col = len(board) , len(board[0])
14          # 对边界上的所有点分别进行深度遍历
15          # 第一列和最后一列
16          for i in range(row):
17
18              self.dfs(board,i,0,row,col)
19              self.dfs(board,i,col-1,row,col)
20          # 第一行和最后一行
21          for j in range(1,col-1):
22              self.dfs(board,0,j,row,col)
23              self.dfs(board,row-1,j,row,col)
24
25          for i in range(row):
26              for j in range(col):
27                  if board[i][j] == "O":
28                      board[i][j] = "X"
29                  if board[i][j] == "T":
30                      board[i][j] = "O"
```

```python
            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]]:
        result = []
        self.recurPartition(s,result, [],  0)
        return result

    def recurPartition(self,s, result, curr, start):
        if start == len(s):
            result.append(list(curr))
            return
        for i in range(start, len(s)):
            if self.isPalindrome(s, start, i):
                curr.append(s[start:i+1])
                self.recurPartition(s,result,curr ,i + 1)
                curr.pop()
    # 判断回文
    def isPalindrome(self, s, begin, end):
        while begin < end :
            if s[begin] != s[end]:
                return False
            begin += 1
            end -= 1
        return True
```

```python
#
# @lc app=leetcode.cn id=132 lang=python3
#
# [132] 分割回文串 II
#
```

```python
class Solution:
    def minCut(self, s: str) -> int:
        n = len(s)
        dp = [[False for _ in range(n)] for _ in range(n)]
        # f[0->n](共n+1个) f[n]=-1
        # f(i) [i, n-1]最小裁剪数
        f = list(reversed(range(-1, n)))
        # f 从右往左更新
        # dp (i 往左更新,j往右更新)
        for i in range(n-1,-1,-1):
            for j in range(i,n):
                if (s[i] == s[j] and (j - i < 2 or dp[i + 1][j - 1])) :
                    dp[i][j] = True
                    # 如果满足回文的条件
                    # f 选取裁剪更少的方案
                    f[i] = min(f[i], f[j + 1] + 1)
        return f[0]
```

```python
#
# @lc app=leetcode.cn id=133 lang=python3
#
# [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)
```

```python
                    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)

        return copy_node
```

```python
#
# @lc app=leetcode.cn id=134 lang=python3
#
# [134] 加油站
#
class Solution:
    def canCompleteCircuit(self, gas: List[int], cost: List[int]) -> int:
        sumGas = sumCost = 0
        start = 0
        diff = 0
        for i in range(len(gas)):
            sumGas += gas[i]
            sumCost += cost[i]
            diff += gas[i] - cost[i]
            if diff < 0:
                start = i + 1 ## 下一个开始
                diff = 0
        return start if sumGas - sumCost >= 0 else -1
```

```python
#
# @lc app=leetcode.cn id=135 lang=python3
#
# [135] 分发糖果
```

75

```python
#
class Solution:
    def candy(self, ratings: List[int]) -> int:
        if len(ratings) == 0:
            return 0
        leng = len(ratings)
        res = [1 for _ in range(leng)]
        for i in range(1,leng):
            if ratings[i] > ratings[i-1]:
                res[i] = res[i-1] + 1
        for i in range(leng-1 , 0 , -1 ):
            if ratings[i-1] > ratings[i]:
                res[i-1] = max(res[i]+1 ,res[i-1])
        return sum(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
```

```
23              headcopy.next = node
24              headcopy = node.next
25          # 复制random部分
26          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          pnew = res = head.next
34          src = head
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)
9           dp = [False for _ in range(n+1)]
10          dp[0] = True
11          # prev true 表示s[j,i)是一个合法单词,从j处切开
12          prev = [[False for _ in range(n)] for _ in range(n+1) ]
13
14          for i in range(n+1):
15              for j in range(i-1,-1,-1):
16                  if dp[j] and s[j:i] in wordDict:
17                      dp[i] = True
18                      prev[i][j] = True
19
20          res = []
21          self.dfs(s,prev,n ,[], res)
22          return res
23
24
25      def dfs( self ,s,prev,cur,path,res):
26          if cur == 0:
27              # 终止条件
28              temp = "␣".join(list (reversed(path)))
29              res.append(temp)
30              return
31
32          for i in range(cur-1,-1,-1):
33              if prev[cur][i]:
34                  path.append(s[i:cur])
35                  self.dfs(s,prev,i,path,res)
36                  path.pop()
```

```
1   #
2   # @lc app=leetcode.cn id=141 lang=python3
3   #
4   # [141] 环形链表
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 hasCycle(self, head: ListNode) -> bool:
```

```python
        '''
        try:
            slow = head
            fast = head.next
            while slow is not fast:
                slow = slow.next
                fast = fast.next.next
            return True
        except:
            return False
        '''
        fast = slow = head
        while fast and fast.next:
            fast = fast.next.next
            slow = slow.next
            if slow == fast:
                return True
        return False
```

```python
#
# @lc app=leetcode.cn id=142 lang=python3
#
# [142] 环形链表 II
#
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution:
    def detectCycle(self, head: ListNode) -> ListNode:
        fast = slow = head
        while fast and fast.next:
            slow = slow.next
            fast = fast.next.next
            if slow == fast:
                #相遇了
                res = head
                while res != slow:
                    slow = slow.next
                    res = res.next
                return res
        return None
```

```python
#
```

```python
# @lc app=leetcode.cn id=143 lang=python3
#
# [143] 重排链表
#
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution:
    def reorderList(self, head: ListNode) -> None:
        """
        Do not return anything, modify head in-place instead.
        """
        if head is None or head.next is None:
            return head
        p1, p2 = head, head
        while p2 and p2.next:
            p1 = p1.next
            p2 = p2.next.next
        # head2 是后面半部分
        head2 = p1.next
        p1.next = None
        # head head2 对应前后两部分

        cur = head2
        rever = None
        # 反转
        while cur:
            temp = cur.next
            cur.next = rever
            rever = cur
            cur = temp

        # head rever 两个合并
        p1 = head
        while rever:
            # 两个链的下一个
            temp = p1.next
            temp2 = rever.next
            # 链接好
            p1.next = rever
            rever.next = temp
            # 下一个循环
            rever = temp2
```

```
48            p1 = temp
49
50        return head
```

```
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:
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 None
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
```

```
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 None
17            result = []
18            stack = []
19            stack.append(root)
20            while stack:
21                p = stack.pop()
22                if p :
23                    result .append(p.val)
24                    stack.append(p.left)
25                    stack.append(p.right)
26
27            return result [::-1]
```

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

```
32
33              self.cache[key] = value
34              self.queue.insert(0,key)
35
36
37  # Your LRUCache object will be instantiated and called as such:
38  # obj = LRUCache(capacity)
39  # param_1 = obj.get(key)
40  # obj.put(key,value)
```

```
1   #
2   # @lc app=leetcode.cn id=147 lang=python3
3   #
4   # [147] 对链表进行插入排序
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 insertionSortList(self, head: ListNode) -> ListNode:
14          p = dummy = ListNode(-1000)
15          dummy.next = head
16          cur = head
17          while cur and cur.next:
18              val = cur.next.val
19              # 顺序的
20              if cur.val < val:
21                  cur = cur.next
22                  continue
23              # 找到p(小于的最后一个节点)
24              if p.next.val > val:
25                  p = dummy
26              while p.next.val < val:
27                  p = p.next
28              # 交换节点
29              next_step = cur.next
30              cur.next = next_step.next
31              next_step.next = p.next
32              p.next = next_step
33          return dummy.next
```

```
1   #
2   # @lc app=leetcode.cn id=148 lang=python3
```

```python
#
# [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:
        if head is None or head.next is None:
            return head
        fast = slow = head
        pre = None
        while fast and fast.next:
            fast = fast.next.next
            pre = slow
            slow = slow.next
        pre.next = None
        return self.mergeTwoLists(self.sortList(head), self.sortList(slow))

    def mergeTwoLists(self, l1, l2):
        res = now = ListNode(-1000)
        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 res.next  # 把0去掉
```

```python
#
# @lc app=leetcode.cn id=149 lang=python3
#
# [149] 直线上最多的点数
#
class Solution:
    def maxPoints(self, points: List[List[int]]) -> int:
        if points is None:
            return 0
        res = 0
        # 两重循环
```

```python
        # 双重字典
        for i in range(len(points)):
            line_map = {}
            same = max_point_num = 0
            for j in range(i + 1, len(points)):
                dx, dy = points[j][0] - points[i][0], points[j][1] - points[i][1]
                # 同一个点
                if dx == 0 and dy == 0:
                    same += 1
                    continue
                # 去除最大公约数部分
                gcd = self.generateGCD(dx, dy)
                if gcd != 0:
                    dx //= gcd
                    dy //= gcd

                if dx in line_map:
                    if dy in line_map[dx]:
                        line_map[dx][dy] += 1
                    else:
                        line_map[dx][dy] = 1
                else:
                    line_map[dx] = {}
                    line_map[dx][dy] = 1
                max_point_num = max(max_point_num, line_map[dx][dy])
            res = max(res, max_point_num + same + 1)
        return res

    # 辗转相除法求最大公约数
    def generateGCD(self, x, y):
        if y == 0:
            return x
        else:
            return self.generateGCD(y, x % y)
```

```python
#
# @lc app=leetcode.cn id=150 lang=python3
#
# [150] 逆波兰表达式求值
#
class Solution:
    def evalRPN(self, tokens: List[str]) -> int:
        nums = []
        for t in tokens:
            if t not in ['+','-','*','/']:
                nums.append(int(t))
```

```python
            else:
                r = nums.pop()
                l = nums.pop()
                if t == '+':
                    temp = l+r
                elif t == '-':
                    temp = l-r
                elif t == '*':
                    temp = l*r
                elif t == '/':
                    if l*r < 0 and l%r != 0:
                        temp = l//r + 1
                    else:
                        temp = l//r
                nums.append(temp)
        return nums.pop()
```

```python
#
# @lc app=leetcode.cn id=151 lang=python3
#
# [151] 翻转字符串里的单词
#

class Solution:
    def reverseWords(self, s: str) -> str:
        if len(s) == 0:
            return s


        '''
        temp = s.split(' ')
        temp = [t for t in temp if len(t) > 0]
        temp.reverse()
        return ' '.join(temp)
        '''
        s = s + " "
        l = 0
        stack = []
        for i in range(l,len(s)):
            if s[i] == " ":
                if l != i:
                    stack.append(s[l:i])
                l = i + 1

        stack.reverse()
        return " ".join(stack)
```

```python
#
# @lc app=leetcode.cn id=152 lang=python3
#
# [152] 乘积最大子序列
#
class Solution:
    def maxProduct(self, nums: List[int]) -> int:
        if nums is None:
            return 0
        maxtmp = mintmp = maxval = nums[0]
        for i in range(1,len(nums)):
            mx ,mn = maxtmp ,mintmp
            maxtmp = max(nums[i] , nums[i]*mx ,nums[i]*mn)
            mintmp = min(nums[i] , nums[i]*mx ,nums[i]*mn)
            maxval = max(maxtmp,maxval)
        return maxval
```

```python
#
# @lc app=leetcode.cn id=153 lang=python3
#
# [153] 寻找旋转排序数组中的最小值
#

class Solution:
    def findMin(self, nums: List[int]) -> int:
        if len(nums) == 1 or nums[0] < nums[-1]: # 升序
            return nums[0]

        l , r = 0, len(nums)-1
        while l < r :
            mid = (l+r)//2
            # 左边
            if nums[0] <= nums[mid]:
                l = mid + 1
            # 在右边
            else :
                r = mid

        return nums[l]
```

```python
#
# @lc app=leetcode.cn id=154 lang=python3
#
# [154] 寻找旋转排序数组中的最小值 II
#

```

```python
class Solution:
    def findMin(self, nums: List[int]) -> int:
        if len(nums) == 1 or nums[0] < nums[-1]: # 升序
            return nums[0]

        l, r = 0, len(nums)-1
        while l < r:
            mid = (l+r)//2
            # 左边
            if nums[mid] > nums[r]:
                l = mid + 1
            # 在右边
            elif nums[mid] < nums[r]:
                r = mid
            # nums[mid] == nums[r]情况
            else:
                r -= 1
        return nums[l]
```

```python
#
# @lc app=leetcode.cn id=155 lang=python3
#
# [155] 最小栈
#
class MinStack:

    def __init__(self):
        """
        initialize your data structure here.
        """
        self.stack = []
        self.min_stack = []

    def push(self, x: int) -> None:
        self.stack.append(x)
        if len(self.min_stack) == 0:
            self.min_stack.append(x)
            return
        if x <= self.min_stack[-1]:
            self.min_stack.append(x)
        else:
            self.min_stack.append(self.min_stack[-1])

    def pop(self) -> None:
        if len(self.stack)>0:
            self.min_stack.pop()
```

```python
28              self.stack.pop()
29
30      def top(self) -> int:
31          if len(self.stack)>0:
32              return self.stack[-1]
33          return None
34
35      def getMin(self) -> int:
36          if len(self.min_stack)>0:
37              return self.min_stack[-1]
38          return None
39
40
41  # Your MinStack object will be instantiated and called as such:
42  # obj = MinStack()
43  # obj.push(x)
44  # obj.pop()
45  # param_3 = obj.top()
46  # param_4 = obj.getMin()
```

```python
1   #
2   # @lc app=leetcode.cn id=160 lang=python3
3   #
4   # [160] 相交链表
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 getIntersectionNode(self, headA: ListNode, headB: ListNode) -> ListNode:
14          p1, p2 = headA, headB
15          # 初始化两个运动结点p1和p2
16          while p1 != p2:
17              # 只要两个结点还未相遇
18              p1 = headB if p1 is None else p1.next
19              # 如果p1走到了链表A的末尾，则换到链表B上
20              p2 = headA if p2 is None else p2.next
21              # 如果p2走到了链表B的末尾，则换到链表A上
22
23          return p1
24          # 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
1   #
```

```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:
                l = mid + 1
```

```python
#
# @lc app=leetcode.cn id=167 lang=python3
#
# [167] 两数之和 II - 输入有序数组
#
class Solution:
    def twoSum(self, numbers: List[int], target: int) -> List[int]:
        l = 0
        r = len(numbers)-1
        while l <= r:
            tmp = numbers[l]+numbers[r]
            if tmp == target:
                return [l+1 , r+1]
            elif tmp < target :
                l += 1
            elif tmp> target:
                r -= 1
```

```python
#
# @lc app=leetcode.cn id=168 lang=python3
#
# [168] Excel表列名称
#
class Solution:
    def convertToTitle(self, n: int) -> str:
```

```python
        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] 求众数
#
class Solution:
    def majorityElement(self, nums: List[int]) -> int:
        return sorted(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=174 lang=python3
#
# [174] 地下城游戏
```

```python
#
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)]

        res[m-1][n-1] = -min(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=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:
        p_len = len(prices)

        #交易次数太多，用贪心
        if k >= p_len//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
```

```
28          for i in range(1, len(prices)):
29              if prices[i] > prices[i−1]:
30                  res += prices[i] − prices[i−1]
31
32          return res
```

```
1   #
2   # @lc app=leetcode.cn id=189 lang=python3
3   #
4   # [189] 旋转数组
5   #
6   class Solution:
7       def rotate(self, nums: List[int], k: int) −> None:
8           """
9           Do not return anything, modify nums in−place instead.
10          """
11          tmp = [0] * len(nums)
12          for i in range(len(nums)):
13              tmp[(i+k)%len(nums)] = nums[i] #recycle
14
15          for i in range(len(nums)):
16              nums[i] = tmp[i]
```

```
1   #
2   # @lc app=leetcode.cn id=198 lang=python3
3   #
4   # [198] 打家劫舍
5   #
6   class Solution:
7       def rob(self, nums: List[int]) −> int:
8           if not nums:
9               return 0
10          f_1 = 0
11          f_2 = 0
12          for i in nums:
13              f_i = max(f_2+i,f_1)
14              f_1 ,f_2 = f_i ,f_1
15          return f_1
```

```
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  # 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 removeElements(self, head: ListNode, val: int) -> ListNode:
14
15         prehead = ListNode(-1)
16         prehead.next = head
17         last, pos = prehead, head
18         while pos is not None:
19             if pos.val == val:
20                 # last 跟上了pos
21                 last.next = pos.next
22             else:
23                 last = pos
24             pos = pos.next
25         return prehead.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
```

```python
        res = [0,0]+ [1]*(n-2)
        for i in range(2,n):
            # 这些没改过
            if res[i] == 1:
                for j in range(2,(n-1)//i+1):
                    res[i*j] = 0
        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] * 127
        mapTtoS = [0] * 127
        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
        point = head # 他来往后走
        prev = None # 新的反转的
```

```
18          while point:
19              # 下一步先保存下来
20              nextpoint = point.next
21              # 反转的接上去
22              point.next = prev
23              prev  = point
24              # 下一步
25              point = nextpoint
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         return max(
13             self.robb(nums[0:-1]),
14             self.robb(nums[len(nums) != 1:])
15         )
16
17     def robb(self ,nums):
18         now = prev = 0
19         for num in nums:
20             now , prev = max(now , prev+num) , now
21         return now
```

```
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            '''
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)
```

```
1   #
2   # @lc app=leetcode.cn id=217 lang=python3
3   #
4   # [217] 存在重复元素
5   #
6   class  Solution:
7       def containsDuplicate( self , nums: List[int ])  −> bool:
8           return  len(nums) != len(set(nums))
```

```
1   #
2   # @lc app=leetcode.cn id=219 lang=python3
3   #
4   # [219] 存在重复元素 II
5   #
6   class  Solution:
7       def containsNearbyDuplicate(self, nums: List[int ],  k:  int )  −> bool:
8           dic  = {}
9           for  key ,val  in  enumerate(nums):
10              if  val  in  dic  and key  − dic[val]<=k :
11                  return  True
12              dic [val ]  = key
```

```python
13          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] * (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=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
#
# [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):
```

```python
            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):
        """
        Initialize your data structure here.
        """
        self.list = []

    def push(self, x: int) -> None:
        """
        Push element x onto stack.
        """
        # 尾部压入
        self.list.append(x)

    def pop(self) -> int:
```

```python
        """
        Removes the element on top of the stack and returns that element.
        """
        # 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:
        """
        Get the top element.
        """
        if len( self . list ) == 0:
            return
        else :
            return  self . list [−1]


    def empty(self) −> bool:
        """
        Returns whether the stack is empty.
        """
        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
```

```python
 10    #         self.left  = None
 11    #         self.right  = None
 12
 13    class Solution:
 14        def invertTree( self , root: TreeNode) -> TreeNode:
 15            if root is None:
 16                return None
 17            root. left  ,root. right  = self.invertTree(root. right )  , self .invertTree(root. left )
 18            return root
```

```python
  1    #
  2    # @lc app=leetcode.cn id=229 lang=python3
  3    #
  4    # [229] 求众数 II
  5    #
  6
  7    class Solution:
  8        def majorityElement(self, nums: List[int ])  -> List[int]:
  9            # 摩尔投票法得到两个大多数
 10            result1  , result2  = -1, -1
 11            score1  , score2  = 0 , 0
 12            for i in range(len(nums)):
 13                if ( result1==nums[i]):
 14                    score1+=1
 15                elif ( result2==nums[i]):
 16                    score2+=1
 17                elif (score1==0):
 18                    result1=nums[i]
 19                    score1=1
 20                elif (score2==0):
 21                    result2=nums[i]
 22                    score2=1
 23                else :
 24                    score1  -= 1
 25                    score2  -= 1
 26
 27            # 统计两个大多数的出现次数
 28            time1,time2 = 0 , 0
 29            for i in range(len(nums)):
 30                if    (nums[i]==result1): time1+=1
 31                elif (nums[i]==result2): time2+= 1
 32
 33            # 得到结果
 34            result  = []
 35            if (time1>len(nums)/3): result.append(result1)
 36            if (time2>len(nums)/3): result.append(result2)
```

```
37            return  result
```

```
1   #
2   # @lc app=leetcode.cn id=230 lang=python3
3   #
4   # [230] 二叉搜索树中第K小的元素
5   #
6
7   # Definition for a binary tree node.
8   # class TreeNode:
9   #     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  root  :
41              return  self .count(root. left ) + self.count(root.right) + 1
42          else :
43              return  0
```

```python
#
# @lc app=leetcode.cn id=231 lang=python3
#
# [231] 2的幂
#
class Solution:
    def isPowerOfTwo(self, n: int) -> bool:
        while n > 1:
            n /= 2
        if n == 1:
            return True
        else:
            return False
```

```python
#
# @lc app=leetcode.cn id=232 lang=python3
#
# [232] 用栈实现队列
#
class MyQueue:

    def __init__(self):
        """
        Initialize  your data structure here.
        """
        self.stack1 = []


    def push(self, x: int) -> None:
        """
        Push element x to the back of queue.
        """
        # 尾部加入
        self.stack1.append(x)


    def pop(self) -> int:
        """
        Removes the element from in front of queue and returns that element.
        """
        temp = self.stack1[0]
        self.stack1.pop(0)
        return temp


```

```python
    def peek(self) -> int:
        """
        Get the front element.
        """
        return self.stack1[0]


    def empty(self) -> bool:
        """
        Returns whether the queue is empty.
        """
        return len(self.stack1) == 0



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

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

```python
#
# @lc app=leetcode.cn id=234 lang=python3
#
```

```python
# [234] 回文链表
#
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution:
    def isPalindrome(self, head: ListNode) -> bool:
        if head is None:
            return True
        rev = None
        slow = fast = head
        # fast 到尾部
        # slow 到中部
        # rev 前半部分的反向
        while fast and fast.next:
            fast = fast.next.next
            rev, rev.next ,slow = slow, rev , slow.next
        # 奇
        if fast:
            slow = slow.next
        # 一个向左,一个向右
        while rev :
            if rev.val != slow.val:
                return False
            slow = slow.next
            rev = rev.next
        return True
```

```python
#
# @lc app=leetcode.cn id=235 lang=python3
#
# [235] 二叉搜索树的最近公共祖先
#
# 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':
        if p is None or q is None or root is None:
```

```python
            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 :
            return right
        # 同理
        if not right :
            return left
        return None
```

```python
#
# @lc app=leetcode.cn id=237 lang=python3
#
# [237] 删除链表中的节点
#
```

```python
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None

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

```python
#
# @lc app=leetcode.cn id=238 lang=python3
#
# [238] 除自身以外数组的乘积
#

class Solution:
    def productExceptSelf(self, nums: List[int]) -> List[int]:
        res = [1] * len(nums)
        right = 1
        for i in range(1, len(nums)):
            res[i] = res[i - 1] * nums[i - 1]

        for i in range(len(nums) - 1, -1, -1):
            res[i] *= right
            right *= nums[i]
        return res
```

```python
#
# @lc app=leetcode.cn id=242 lang=python3
#
# [242] 有效的字母异位词
#
class Solution:
    def isAnagram(self, s: str, t: str) -> bool:
        dic1, dic2 = {}, {}
        for item in s:
            dic1[item] = dic1.get(item, 0) + 1
        for item in t:
            dic2[item] = dic2.get(item, 0) + 1
        return dic1 == dic2
```

```python
#
# @lc app=leetcode.cn id=257 lang=python3
#
# [257] 二叉树的所有路径
#
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution:
    def binaryTreePaths(self, root: TreeNode) -> List[str]:
        if root is None:
            return []
        paths = []
        # 路径  节点  现有的子路
        self.getpath(paths, root, [])
        res = ['->'.join(path) for path in paths]
        return res

    def getpath(self, res, node, path):
        # 没有左右子树
        if node.left is None and node.right is None:
            res.append(path + [str(node.val)])
            return
        # 否则
        path = path + [str(node.val)]
        if node.left is not None:
            self.getpath(res, node.left, path)
        if node.right is not None:
            self.getpath(res, node.right, path)
```

```python
#
# @lc app=leetcode.cn id=258 lang=python3
#
# [258] 各位相加
#
class Solution:
    def addDigits(self, num: int) -> int:
        t = num
        while t >= 10:
            t = sum([int(char) for char in str(t)])
        return t
```

```python
#
# @lc app=leetcode.cn id=263 lang=python3
#
# [263] 丑数
#
class Solution:
    def isUgly(self, num: int) -> bool:
        if num <= 0:
            return False

        divisors = [2, 3, 5]
        for d in divisors:
            while num % d == 0:
                num /= d
        return num == 1
```

```python
#
# @lc app=leetcode.cn id=264 lang=python3
#
# [264] 丑数 II
#
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:
```

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

```python
#
# @lc app=leetcode.cn id=278 lang=python3
#
# [278] 第一个错误的版本
#
# The isBadVersion API is already defined for you.
# @param version, an integer
# @return a bool
# def isBadVersion(version):

class Solution:
    def firstBadVersion(self, n):
        """
        :type n: int
        :rtype: int
        """
        l ,r = 0 , n−1
        while l <= r:
```

```
19            mid = (l+r)//2
20              if isBadVersion(0) == isBadVersion(mid):
21                  l = mid + 1
22              elif isBadVersion(n) == isBadVersion(mid):
23                  r = mid −1
24          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      _dp = [0]
8      def numSquares(self, n: int) −> int:
9          '''
10          # 性能测试过不了
11          # 先扣出来2的部分
12          while n % 4 == 0 :
13              n /= 4
14          if n %8 == 7:
15              return 4
16          n = int(n)
17          dp = [0]+[float('inf') for i in range(n)]
18          # 对每个数来说
19          for i in range(n + 1):
20              j = 1
21              while i >= j**2 :
22                  # 取小
23                  dp[i] = min(dp[i], dp[i− j**2] + 1)
24                  j += 1
25
26          return dp[n]
27          '''
28          dp = self._dp
29          while len(dp) <= n:
30              dp += min(dp[−i*i] for i in range(1, int(len(dp)**0.5+1))) + 1,
31          return dp[n]
```

```
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          """
 9          Do not return anything, modify nums in−place instead.
10          """
11          ling = []
12          for i in range(len(nums)):
13              if nums[i] == 0 :
14                  ling .append(i)
15
16          for i in ling [::−1]:
17              nums.pop(i)
18              nums.append(0)
19          return nums
```

```
 1  #
 2  # @lc app=leetcode.cn id=290 lang=python3
 3  #
 4  # [290] 单词规律
 5  #
 6  class Solution:
 7      def wordPattern(self, pattern: str, str: str) −> bool:
 8          word_list = str. split ( '␣')
 9          pattern_list = list (pattern)
10          if len(word_list) != len(pattern_list):
11              return False
12          for i, word in enumerate(word_list):
13              idx = word_list.index(word)
14              idx2 = pattern_list.index(pattern[i])
15              if idx != idx2:
16                  return False
17          return True
```

```
 1  #
 2  # @lc app=leetcode.cn id=292 lang=python3
 3  #
 4  # [292] Nim 游戏
 5  #
 6  class Solution:
 7      def canWinNim(self, n: int) −> bool:
 8          return n%4 != 0
```

```
 1  #
 2  # @lc app=leetcode.cn id=299 lang=python3
 3  #
 4  # [299] 猜数字游戏
 5  #
 6
 7  class Solution:
```

```python
8        def getHint(self, secret: str, guess: str) -> str:
9            a = b = 0
10           dic = {}
11           for i in range(len(secret)):
12               if secret[i] == guess[i]:
13                   a += 1
14               dic[secret[i]] = dic[secret[i]] + 1 if secret[i] in dic else 1
15           for i in range(len(guess)):
16               if guess[i] in dic and dic[guess[i]] > 0:
17                   b += 1
18                   dic[guess[i]] -= 1
19           b -= a
20           return f"{a}A{b}B"
```

```python
1   #
2   # @lc app=leetcode.cn id=300 lang=python3
3   #
4   # [300] 最长上升子序列
5   #
6
7   class Solution:
8       def lengthOfLIS(self, nums: List[int]) -> int:
9           if not nums:
10              return 0
11
12          '''
13          dp = [1] * len(nums)
14
15          for i in range(1,len(nums)):
16              for j in range(i):
17                  # 如果要求非严格递增,将此行 '<' 改为 '<=' 即可
18                  if(nums[j] < nums[i]):
19                      dp[i] = max(dp[i], dp[j] + 1)
20          return max(dp)
21          '''
22
23          up_list = []
24          for i in range(len(nums)):
25              # 二分查找
26              left, right = 0, len(up_list)-1
27              while left <= right:
28                  mid = (left+right)//2
29                  if up_list[mid] < nums[i]:
30                      left = mid+1
31                  else:
32                      right = mid-1
```

```
33              # 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
34              if left == len(up_list):
35                  up_list.append(nums[i])
36              else:
37                  up_list[left] = nums[i]
38          return len(up_list)
```

```
1  #
2  # @lc app=leetcode.cn id=303 lang=python3
3  #
4  # [303] 区域和检索 - 数组不可变
5  #
6  class NumArray:
7
8      def __init__(self, nums: List[int]):
9          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
14
15     def sumRange(self, i: int, j: int) -> int:
16         return self.list[j+1] - self.list[i]
17
18
19
20 # Your NumArray object will be instantiated and called as such:
21 # obj = NumArray(nums)
22 # 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
19             idx  += 1
20         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
7  class Solution:
8      def coinChange(self, coins: List[int], amount: int) −> int:
9          if amount == 0:
10             return 0
11         if coins is None or len(coins) == 0:
```

```
12                return −1
13
14        coins.sort()
15        dp = [float('inf')] * (amount + 1)
16        dp[0] = 0
17
18        for coin in coins:
19            for j in range(coin, amount+1):
20                dp[j] = min(dp[j], dp[j − coin] + 1)
21
22        return −1 if dp[−1] > amount else dp[−1]
```

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

```
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
```

```
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           """
```

```
 9            Do not return anything, modify s in−place instead.
10            """
11            n =len(s)
12            for i in range(n//2):
13                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 = []
11          pos1 = pos2 = 0
```

```
12          while pos1 < len(nums1) and pos2 < len(nums2):
13              if nums1[pos1] == nums2[pos2]:
14                  res.append(nums1[pos1])
15                  pos1 += 1
16                  pos2 += 1
17              elif nums1[pos1] < nums2[pos2]:
18                  pos1 += 1
19              else:
20                  pos2 += 1
21      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
# @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:
```

```python
class Solution:
    def guessNumber(self, n: int) -> int:
        start, end = 1, n
        while start <= end:
            mid = (start + end)//2
            if guess(mid) == 0:
                return mid
            elif guess(mid) == 1:
                start = mid + 1
            else:
                end = mid
```

```python
#
# @lc app=leetcode.cn id=383 lang=python3
#
# [383] 赎金信
#
class Solution:
    def canConstruct(self, ransomNote: str, magazine: str) -> bool:
        letter_map = {}
        for i in magazine:
            '''
            if i in letter_map:
                letter_map[i] += 1
            else:
                letter_map[i] = 1
            '''
            letter_map[i] = letter_map.get(i, 0) + 1
        for i in ransomNote:
            '''
            if i not in letter_map:
                return False
            else:
                letter_map[i] -= 1
            '''
            letter_map[i] = letter_map.get(i, 0) - 1
            if letter_map[i]<0:
                return False
        return True
```

```python
#
# @lc app=leetcode.cn id=387 lang=python3
#
# [387] 字符串中的第一个唯一字符
#
class Solution:
```

```python
    def firstUniqChar(self, s: str) -> int:
        letter_map = {}
        for i in s:
            letter_map[i] = letter_map.get(i, 0) + 1
        for i in range(len(s)):
            if letter_map[s[i]] == 1:
                return i
        return -1
```

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

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:
```

```python
        nums = list(set(nums))
        if len(nums)<3:
            return max(nums)
        nums.sort()
        return nums[-3]
```

```python
#
# @lc app=leetcode.cn id=434 lang=python3
#
# [434] 字符串中的单词数
#
class Solution:
    def countSegments(self, s: str) -> int:
        if not s:
            return 0
        segment_count = 0
        for i in range(len(s)):
            if i == 0 and s[i] != ' ':
                segment_count = 1
            elif s[i-1] == ' ' and s[i] != ' ':
                segment_count += 1

        return segment_count
```

```python
#
# @lc app=leetcode.cn id=442 lang=python3
#
# [442] 数组中重复的数据
#
class Solution:
    def findDuplicates(self, nums: List[int]) -> List[int]:
        returnlist =[]
        for x in nums:
            x = abs(x)
            if nums[x-1]<0:# 出现过了
                returnlist.append(x)
            else:
                nums[x-1]*=-1
        return returnlist
```

```python
#
# @lc app=leetcode.cn id=443 lang=python3
#
# [443] 压缩字符串
#
class Solution:
    def compress(self, chars: List[str]) -> int:
```

```python
        # 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
                    chars[walker+1] = c
                    walker += 1

            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
        out = []
        leng = len(nums)
        for i in range(leng):
            if i+1 not in nums:
                out.append(i+1)
        return out
        '''
        for num in nums:
            index = abs(num)-1
            if nums[index]>0:
                nums[index] *= -1
```

```
21
22          out = []
23          for i in range(len(nums)):
24              if nums[i]>0:
25                  out.append(i+1)
26          return out
```

```
1   #
2   # @lc app=leetcode.cn id=485 lang=python3
3   #
4   # [485] 最大连续1的个数
5   #
6   class Solution:
7       def findMaxConsecutiveOnes(self, nums: List[int]) -> int:
8           maxval = 0
9           tmp = 0
10          for i in range(len(nums)):
11              if nums[i] != 0 :
12                  tmp += 1
13              else :
14                  maxval = max(maxval,tmp)
15                  tmp = 0
16          maxval = max(maxval,tmp)
17          return maxval
```

```
1   #
2   # @lc app=leetcode.cn id=494 lang=python3
3   #
4   # [494] 目标和
5   #
6   class Solution:
7       def findTargetSumWays(self, nums: List[int], S: int) -> int:
8           sum_nums = sum(nums)
9           if sum_nums < S or (S + sum_nums)%2 != 0:
10              return 0
11
12          target = (S + sum_nums) >> 1
13          mem = [0]*(target + 1)
14          mem[0] = 1
15          for num in nums:
16              for i in range(target, num-1, -1):
17                  mem[i] += mem[i - num]
18          return mem[target]
```

```
1   #
2   # @lc app=leetcode.cn id=532 lang=python3
3   #
```

```python
# [532] 数组中的K-diff数对
#
class Solution:
    def findPairs(self, nums: List[int], k: int) -> int:

        c = collections.Counter(nums)
        dui = 0

        if k < 0:
            return 0
        elif k == 0:
            for i in c.keys():
                if c[i]>1:
                    dui += 1
        else:
            for i in c.keys():
                if i + k in c:
                    dui +=1
        return dui
```

```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)):
```

```
14              # 等于1表示被别的圈包进去了,等于0表示再开一个圈
15              if  visited[i] == 0:
16                  self.dfs(M, visited, i)
17                  count += 1
18          return count
19
20      # 判断和i认识的都是哪些人
21      def dfs(self, M, visited, i):
22          # 全1了
23          if sum(visited) == len(M):
24              return
25          for j in range(len(M)):
26              if j != i and visited[j] == 0 and M[i][j] == 1:
27                  visited[j] = 1
28                  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
7  class Solution:
8      def reverseWords(self, s: str) -> str:
9          return ' '.join([word[::-1] for word in s.split(' ')])
```

```
1  #
2  # @lc app=leetcode.cn id=561 lang=python3
```

```
#
# [561] 数组拆分 I
#
class Solution:
    def arrayPairSum(self, nums: List[int]) -> int:
        nums.sort()
        return sum(nums[::2])
```

```
#
# @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
```

```
#
# @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))
```

```
#
# @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()
```

```
10          n=len(nums)
11          i,j=0,n−1
12          while i<n and nums[i]==num_sort[i]:
13              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           count = 0
9           for i in range(len(flowerbed)):
10              curr = flowerbed[i]
11              if i >= 1:
12                  curr += flowerbed[i − 1]
13              if i < len(flowerbed)−1:
14                  curr += flowerbed[i + 1]
15              if curr == 0:
16                  count += 1
17                  flowerbed[i] = 1
18
19          return count>=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)
12          return maxmean/k
```

```
 1  #
 2  # @lc app=leetcode.cn id=661 lang=python3
 3  #
 4  # [661] 图片平滑器
 5  #
 6  class Solution:
 7      def imageSmoother(self, M: List[List[int]]) −> List[List[int]]:
 8      #def imageSmoother(self, M):
 9          r = len(M)
10          c = len(M[0])
11          res = []
12
13          for i in range(r):
14              tmp = []
15              for j in range(c):
16                  value,count=M[i][j],1
17                  if i−1>=0:
18                      value += M[i−1][j]
19                      count += 1
20                      if j−1>=0:
21                          value += M[i−1][j−1]
22                          count += 1
23                      if j+1<c:
24                          value += M[i−1][j+1]
25                          count += 1
26                  if i+1<r:
27                      value += M[i+1][j]
28                      count += 1
29                      if j−1>=0:
30                          value += M[i+1][j−1]
31                          count += 1
32                      if j+1<c:
33                          value += M[i+1][j+1]
34                          count += 1
35                  if j−1>=0:
36                      value += M[i][j−1]
37                      count += 1
38                  if j+1<c:
39                      value += M[i][j+1]
40                      count += 1
```

```
41            tmp.append(int(value/count))
42         res.append(tmp)
43     return res
```

```
1  #
2  # @lc app=leetcode.cn id=665 lang=python3
3  #
4  # [665] 非递减数列
5  #
6  class Solution:
7      def checkPossibility(self, nums: List[int]) -> bool:
8          count = 0
9          for i in range(len(nums)-1):
10             if nums[i]>nums[i+1]:
11                 count +=1
12                 #变相去掉nums[i]
13                 if i <1 or nums[i-1] <= nums[i+1]:
14                     nums[i] = nums[i+1]
15                 else:
16                     # 变相去掉nums[i+1]
17                     nums[i+1]=nums[i]
18         return count <= 1
```

```
1  #
2  # @lc app=leetcode.cn id=674 lang=python3
3  #
4  # [674] 最长连续递增序列
5  #
6  class Solution:
7      def findLengthOfLCIS(self, nums: List[int]) -> int:
8          if not nums:
9              return 0
10         count = 1
11         res = 0
12         for i in range(len(nums)-1):
13             if nums[i] < nums[i+1]:
14                 count += 1
15             else:
16                 res = max(res,count)
17                 count = 1
18         return max(res,count)
```

```
1  #
2  # @lc app=leetcode.cn id=680 lang=python3
3  #
4  # [680] 验证回文字符串
5  #
```

```python
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:
        ans = 0
        for i in range(len(grid)):
            for j in range(len(grid[0])):
                if grid[i][j] == 1:
                    grid[i][j] = 0
                    ans = max(self.dfs(grid, i, j), ans)
                    # ans = max(self.bfs(grid, i, j), ans)
        return ans

    def dfs(self, grid, i, j):
        # DFS based on stack
        stack = [(i, j)]
        area = 0
        # Stack for DFS
        while stack:
            r, c = stack.pop(-1)
            area += 1
            for nr, nc in ((r - 1, c), (r + 1, c), (r, c - 1), (r, c + 1)):
                if (0 <= nr < len(grid) and
                        0 <= nc < len(grid[0]) and grid[nr][nc]):
                    stack.append((nr, nc))
                    grid[nr][nc] = 0
        return area
```

```python
#
# @lc app=leetcode.cn id=836 lang=python3
#
# [836] 矩形重叠
#
class Solution:
```

```python
def isRectangleOverlap(self, rec1: List[int], rec2: List[int]) -> bool:
    return not (rec1[2] <= rec2[0] or  # rec1的右边在rec2的左边
                rec1[3] <= rec2[1] or  # rec1的上边在rec2的下边
                rec1[0] >= rec2[2] or  # rec1的左边在rec2的右边
                rec1[1] >= rec2[3])    # rec1的下边在rec2的上边
```