牛客高频

2022年5月9日　　21:28

1. 快速排序代码

    -
      ```
      class Solution(object):
          # 主函数
          def Quicksort1(self, nums):
              self.Qsort(nums, 0, len(nums)-1)
              return nums

          # 核心函数
          # 对数组nums[low, high]进行快速排序
          def Qsort(self, nums, low, high):
              if low >= high:
                  return
              # 通过交换元素构造分界点索引
              pivot = self.Partition(nums, low, high)
              # 现在nums[low, pivot-1]都小于nums[pivot]
              # nums[pivot+1, high]都大于nums[pivot]
              self.Qsort(nums, low, pivot-1)
              self.Qsort(nums, pivot+1, high)

          # 交换元素获得分界点索引pivot
          def Partition(self, nums, low, high):
              if low == high:
                  return low
              p = nums[low]
              while low < high:
                  while low < high and nums[high] >= p:
                      high -= 1
                  self.swap(nums, low, high)
                  while low < high and nums[low] < p:
                      low += 1
                  # 因为p不在low的位置就在high的位置
                  self.swap(nums, low, high)
              return low

          # 将nums[i]和nums[j]的顺序交换
          def swap(self, nums, i, j):
              nums[i], nums[j] = nums[j], nums[i]
      ```

    -
      ```
      def Quicksort2(self, nums):
          # 当nums中只有一个元素，不用排序
          if len(nums) < 2:
              return nums
          # nums的中间元素作为基准
          pivot = nums[len(nums)//2]
          del nums[len(nums)//2]
          nums_left = []  # 记录比pivot小的元素
          nums_right = []  # 记录比pivot大的元素
          for item in nums:
              if item > pivot:
                  nums_right.append(item)
              elif item <= pivot:
                  nums_left.append(item)
          return self.Quicksort2(nums_left) + [pivot] + self.Quicksort2(nums_right)
      ```
2. 归并排序代码

    -
      ```
      class Solution:
          def MySort(self , arr: List[int]) -> List[int]:
              # write code here
              if len(arr) <= 1:
                  return arr
              self.Mergesort(arr, 0, len(arr)-1)
              return arr
          def Mergesort(self, nums, low, high):
              if low >= high:
                  return
              mid = (high+low) // 2
              self.Mergesort(nums, low, mid)
              self.Mergesort(nums, mid+1, high)
      ```

```
        self.Merge(nums, low, mid, high)
    def Merge(self, nums, low, mid, high):
        total = []
        i = low
        j = mid+1
        while i <= mid and j <= high:
            if nums[i] <= nums[j]:
                total.append(nums[i])
                i += 1
            else:
                total.append(nums[j])
                j += 1
        if j > high:
            total.extend(nums[i:mid+1])
        else:
            total.extend(nums[j:high])
        for k in range(len(total)):
            nums[low+k] = total[k]
```

-
  ```
  class Solution(object):
      def Mergesort1(self, nums):
          if len(nums) <= 1:
              return nums
          mid = len(nums) // 2
          left = self.Mergesort1(nums[:mid])
          right = self.Mergesort1(nums[mid:])
          return self.Merge1(left, right)

      def Merge1(self, left, right):
          merge = []
          l = r = 0
          while l < len(left) and r < len(right):
              if left[l] <= right[r]:
                  merge.append(left[l])
                  l += 1
              else:
                  merge.append(right[r])
                  r += 1
          if l == len(left):
              merge.extend(right[r:])
          else:
              merge.extend(left[l:])
          return merge
  ```

3. 合并两个排序链表
   - 定义伪头结点+while循换+两链表值比较

   -
     ```
     class Solution:
         def Merge(self , pHead1: ListNode, pHead2: ListNode) -> ListNode:
             # write code here
             if not pHead1:
                 return pHead2
             if not pHead2:
                 return pHead1
             dummy = ListNode(0)
             p = dummy
             while pHead1 and pHead2:
                 if pHead1.val <= pHead2.val:
                     p.next = pHead1
                     pHead1 = pHead1.next
                 else:
                     p.next = pHead2
                     pHead2 = pHead2.next
                 p = p.next
             p.next = pHead1 if pHead1 else pHead2
             return dummy.next
     ```
   - 递归实现

   -
     ```
     class Solution:
         def Merge(self , pHead1: ListNode, pHead2: ListNode) -> ListNode:
             # write code here
             if not pHead1:
                 return pHead2
             if not pHead2:
                 return pHead1
             if pHead1.val <= pHead2.val:
                 pHead1.next = self.Merge(pHead1.next, pHead2)
                 return pHead1
             else:
                 pHead2.next = self.Merge(pHead1, pHead2.next)
                 return pHead2
     ```
4. 最长公共子串
   - 滑动窗口比较法

-
  ```
  class Solution:
      def LCS(self , str1 , str2 ):
          # write code here
          res = ''
          left = 0
          for i in range(1, len(str1)+1):
              if str1[left:i] in str2:
                  if len(str1[left:i]) > len(res):
                      res = str1[left:i]
              else:
                  left += 1
          return res
  ```
- 动态规划法

-
  ```
  class Solution:
      def LCS(self , str1: str, str2: str) -> str:
          # write code here
          max_len = 0
          dp = [[0 for _ in range(len(str2)+1)] for _ in range(len(str1)+1)]
          for i in range(1, len(str1)+1):
              for j in range(1, len(str2)+1):
                  if str1[i-1] == str2[j-1]:
                      dp[i][j] = dp[i-1][j-1] + 1
                      if dp[i][j] > max_len:
                          max_len = dp[i][j]
                          end = j
                  else:
                      dp[i][j] = 0
          return str2[end-max_len:end]
  ```

5. 三数之和
   - 循换+双指针

   -
     ```
     class Solution:
         def threeSum(self , num: List[int]) -> List[List[int]]:
             # write code here
             if len(num) <= 2:
                 return []
             res = []
             num.sort()
             for i in range(len(num)):
                 if i > 0 and num[i] == num[i-1]:
                     continue
                 two_res = self.twoSum(num[i+1:], -num[i])
                 if two_res:
                     for item in two_res:
                         item.append(num[i])
                         res.append(item)
             return res
         def twoSum(self, num, target):
             res = []
             left = 0
             right = len(num) - 1
             while left < right:
                 cur_left = num[left]
                 cur_right = num[right]
                 if cur_left + cur_right == target:
                     res.append([cur_left, cur_right])
                     while left < right and num[left] == cur_left:
                         left += 1
                     while left < right and num[right] == cur_right:
                         right -= 1
                 elif cur_left + cur_right > target:
                     while left < right and num[right] == cur_right:
                         right -= 1
                 elif cur_left + cur_right < target:
                     while left < right and num[left] == cur_left:
                         left += 1
             return res
     ```
6. 最长上升子序列（三）
   - 动态规划+贪心算法+反推

   -
     ```
     class Solution:
         def LIS(self , arr ):
             # write code here
             vec = []
             dp = [1 for _ in range(len(arr))]
             import bisect
             for i in range(len(arr)):
                 idx = bisect.bisect_left(vec, arr[i])
                 if idx >= len(vec):
                     vec.append(arr[i])
                 else:
                     vec[idx] = arr[i]
     ```

```
            dp[i] = idx + 1
        L = max(dp)
        res = []
        for j in range(len(arr)-1, -1, -1):
            if dp[j] == L:
                res.append(arr[j])
                L -= 1
        return res[::-1]
```

7. 求平方根
    - 拟牛顿法
    - class Solution:

```
class Solution:
    def sqrt(self , x ):
        # write code here
        if x <= 0:
            return 0
        ori = x
        while ori > x / ori:
            ori = (ori + x / ori) / 2
        return ori
```

8. 在旋转过的有序数组中寻找目标值
    - 双指针+mid与右指针+判断target属于那个区间
    - class Solution:

```
class Solution:
    def search(self , nums , target ):
        # write code here
        if not nums:
            return -1
        left = 0
        right = len(nums) - 1
        while left <= right:
            mid = left + (right - left) // 2
            if nums[mid] == target:
                return mid
            # 判断mid属于那个区间
                # mid属于左区间
            if nums[mid] > nums[right]:
                # 判断target属于哪个区间
                if nums[left] <= target < nums[mid]:
                    right = mid - 1
                else:
                    left = mid + 1
                # mid属于右区间
            else:
                if nums[mid] < target <= nums[right]:
                    left = mid + 1
                else:
                    right = mid - 1
        return -1
```

9. 合并K个已排序链表
    - 归并排序
    - class Solution:

```
class Solution:
    def mergeKLists(self , lists: List[ListNode]) -> ListNode:
        # write code here
        if not lists:
            return
        if len(lists) == 1:
            return lists[0]
        mid = len(lists) // 2
        left_list = self.mergeKLists(lists[:mid])
        right_list = self.mergeKLists(lists[mid:])
        res = self.Merge(left_list, right_list)
        return res
    def Merge(self, head1, head2):
        if not head1:
            return head2
        if not head2:
            return head1
        total = ListNode(0)
        p = total
        while head1 and head2:
            if head1.val <= head2.val:
                p.next = head1
                head1 = head1.next
            else:
                p.next = head2
                head2 = head2.next
            p = p.next
        p.next = head1 if head1 else head2
        return total.next
```

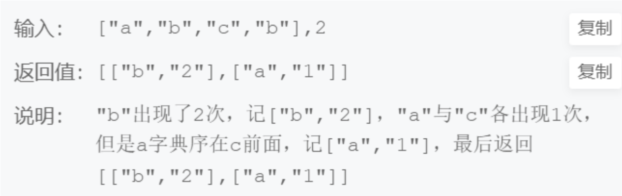
10. 单链表排序
    - 归并排序
    -
```
class Solution:
    def sortInList(self, head):
        # write code here
        if not head or not head.next:
            return head
        dummy = self.Mergesort(head)
        return dummy

    def Mergesort(self, head):
        if not head or not head.next:
            return head
        slow = fast = head
        while fast.next and fast.next.next:
            slow = slow.next
            fast = fast.next.next
        temp = slow.next
        slow.next = None
        left = self.Mergesort(head)
        right = self.Mergesort(temp)
        rel = self.Merge(left, right)
        return rel

    def Merge(self, h1, h2):
        if not h1:
            return h2
        if not h2:
            return h1
        res = ListNode(0)
        p = res
        while h1 and h2:
            if h1.val <= h2.val:
                p.next = h1
                h1 = h1.next
            else:
                p.next = h2
                h2 = h2.next
            p = p.next
        p.next = h1 if h1 else h2
        return res.next
```

11. 表达式求值
    - 字符串表达式+实现加减乘+递归处理括号里的内容
    -
```
class Solution:
    def solve(self , s: str) -> int:
        # write code here
        s = s.strip()
        nums = []
        val = 0
        i = 0
        sign = '+'
        while i < len(s):
            if '0' <= s[i] <= '9':
                val = val * 10 + int(s[i])
                i += 1
            if i < len(s) and s[i] == '(':
                partation = 1
                j = i + 1
                i += 1
                while partation != 0:
                    if s[i] == '(':
                        partation += 1
                    elif s[i] == ')':
                        partation -= 1
                    i += 1
                val = self.solve(s[j:i-1])
            if i == len(s) or s[i] in ['+', '-', '*']:
                if sign == '+':
                    nums.append(val)
                elif sign == '-':
                    nums.append(-val)
                elif sign == '*':
                    nums.append(nums.pop()*val)
                val = 0
                if i != len(s):
                    sign = s[i]
                i += 1
        return sum(nums)
```

12. 字符串出现次数的TopK问题

- 

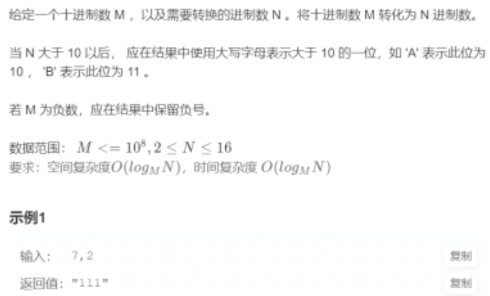
  输入： ["a","b","c","b"],2

  返回值：[["b","2"],["a","1"]]

  说明： "b"出现了2次，记["b","2"]，"a"与"c"各出现1次，但是a字典序在c前面，记["a","1"]，最后返回[["b","2"],["a","1"]]

- 哈希表记录每个元素出现次数+sorted排序+先字母后数据

-
  ```
  class Solution:
      def topKstrings(self , strings: List[str], k: int) -> List[List[str]]:
          # write code here
          if not strings:
              return []
          hash_dict = {}
          for item in strings:
              if item in hash_dict:
                  hash_dict[item] += 1
              else:
                  hash_dict[item] = 1
          import collections
          hash_dict = sorted(hash_dict.items(), key=lambda x:x[0], reverse=False)
          hash_dict = sorted(hash_dict, key=lambda x:x[1], reverse=True)
          if len(hash_dict) < k:
              return hash_dict
          else:
              return hash_dict[:k]
  ```

13. 进制转换
    - 

      给定一个十进制数 M ，以及需要转换的进制数 N 。将十进制数 M 转化为 N 进制数。

      当 N 大于 10 以后， 应在结果中使用大写字母表示大于 10 的一位，如 'A' 表示此位为 10 ， 'B' 表示此位为 11 。

      若 M 为负数，应在结果中保留负号。

      数据范围： $M <= 10^8, 2 \le N \le 16$
      要求：空间复杂度$O(log_M N)$，时间复杂度 $O(log_M N)$

      示例1

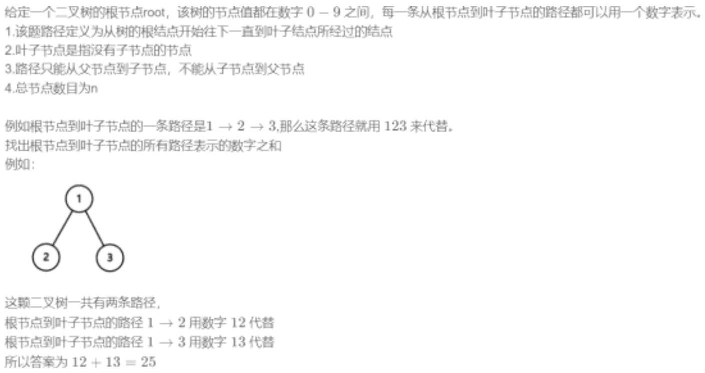
      输入： 7,2

      返回值："111"

    - 10进制转其他进制+取整取余

    -
      ```
      class Solution:
          def solve(self , M: int, N: int) -> str:
              # write code here
              temp = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E', 'F']
              sign = ''
              res = []
              if M < 0:
                  sign = '-'
                  M = -M
              while M > 0:
                  cur = M % N
                  M = M // N
                  res.append(temp[cur])
              res.append(sign)
              return ''.join(res[::-1])
      ```
14. 编辑距离（二）
    - 动态规划+二维dp+左上角+左边+上边

    -
      ```
      class Solution:
          def minEditCost(self , str1: str, str2: str, ic: int, dc: int, rc: int) -> int:
              # write code here
              dp = [[0 for _ in range(len(str2)+1)] for _ in range(len(str1)+1)]
              for i in range(1, len(str2)+1):
                  dp[0][i] = dp[0][i-1] + ic
              for j in range(1, len(str1)+1):
                  dp[j][0] = dp[j-1][0] + dc
              for i in range(1, len(str1)+1):
                  for j in range(1, len(str2)+1):
                      if str1[i-1] == str2[j-1]:
                          dp[i][j] = dp[i-1][j-1]
                      else:
                          dp[i][j] = min(dp[i-1][j-1]+rc, dp[i-1][j]+dc, dp[i][j-1]+ic)
              return dp[-1][-1]
      ```
15. 二叉树根节点到所有叶子结点的路径总和



给定一个二叉树的根节点root，该树的节点值都在数字 $0-9$ 之间，每一条从根节点到叶子节点的路径都可以用一个数字表示。
1.该题路径定义为从树的根结点开始往下一直到叶子结点所经过的结点
2.叶子节点是指没有子节点的节点
3.路径只能从父节点到子节点，不能从子节点到父节点
4.总节点数目为n

例如根节点到叶子节点的一条路径是$1 \to 2 \to 3$,那么这条路径就用 123 来代替。
找出根节点到叶子节点的所有路径表示的数字之和
例如：

这颗二叉树一共有两条路径，
根节点到叶子节点的路径 $1 \to 2$ 用数字 12 代替
根节点到叶子节点的路径 $1 \to 3$ 用数字 13 代替
所以答案为 $12+13=25$

- 深度优先搜索+保存路径

-
  ```
  class Solution:
      def sumNumbers(self , root: TreeNode) -> int:
          # write code here
          if not root:
              return 0
          res = []
          cur = []
          def recur(root):
              if not root:
                  return
              cur.append(str(root.val))
              if not root.left and not root.right:
                  res.append(int(''.join(cur)))
              if root.left:
                  recur(root.left)
              if root.right:
                  recur(root.right)
              cur.pop()
          recur(root)
          return sum(res)
  ```

16. 二叉树中和为某一值的路径（二）

- 深度优先搜索+保存路径

-
  ```
  class Solution:
      def FindPath(self , root: TreeNode, target: int) -> List[List[int]]:
          # write code here
          if not root:
              return []
          res = []
          cur = []
          def recur(root, num):
              if not root:
                  return
              cur.append(root.val)
              if root.val == num and not root.left and not root.right:
                  res.append(list(cur))
              else:
                  if root.left:
                      recur(root.left, num-root.val)
                  if root.right:
                      recur(root.right, num-root.val)
              cur.pop()
          recur(root, target)
          return res
  ```

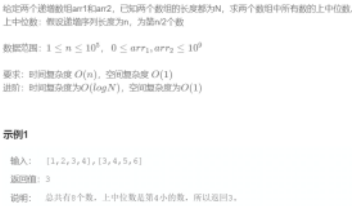
17. 链表内指定区间反转

- 找区间+反转+链接回原链表

-
  ```
  class Solution:
      def reverseBetween(self , head: ListNode, m: int, n: int) -> ListNode:
          # write code here
          if not head or not head.next:
              return head
          dummy = ListNode(0)
          dummy.next = head
          left = right = dummy
          pre = None
          for _ in range(m-1):
              left = left.next
          pre = left
          left = left.next
          for _ in range(n):
              right = right.next
          temp = right.next
          left, right = self.reverses(left, right)
          right.next = temp
          pre.next = left
  ```

```
        return dummy.next
    def reverses(self, start, end):
        pre = None
        cur = start
        while pre != end:
            temp = cur.next
            cur.next = pre
            pre = cur
            cur = temp
        return end, start
```

18. 在两个长度相等的排序数组中找到上中位数

- 

  给定两个递增数组arr1和arr2，已知两个数组的长度都为N，求两个数组中所有数的上中位数。
  上中位数：假设递增序列长度为n，为第n/2个数

  数据范围：$1 \leq n \leq 10^5$，$0 \leq arr_1, arr_2 \leq 10^9$

  要求：时间复杂度 $O(n)$，空间复杂度 $O(1)$
  进阶：时间复杂度为$O(logN)$，空间复杂度为$O(1)$

  示例1

  输入：　[1,2,3,4],[3,4,5,6]
  返回值：3
  说明：　总共有8个数，上中位数是第4小的数，所以返回3。

- 双指针+记录已排序个数

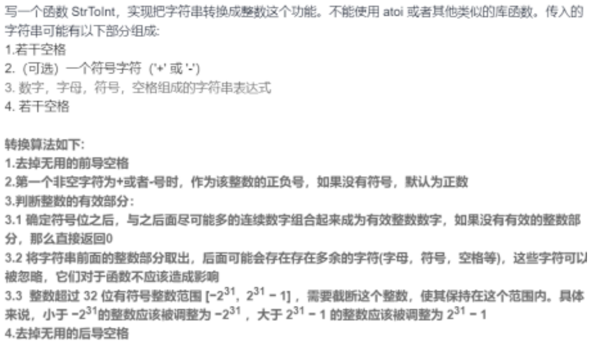
-
  ```
  class Solution:
      def findMedianinTwoSortedAray(self , arr1: List[int], arr2: List[int]) -> int:
          # write code here
          if not arr1:
              return arr2[-1]
          if not arr2:
              return arr1[-1]
          res = self.Mid(arr1, arr2, len(arr1))
          return res
      def Mid(self, arr1, arr2, N):
          res = 0
          p1 = 0
          p2 = 0
          num = 0
          while p1 < len(arr1) and p2 < len(arr2):
              if arr1[p1] <= arr2[p2]:
                  res = arr1[p1]
                  p1 += 1
              else:
                  res = arr2[p2]
                  p2 += 1
              num += 1
              if num == N:
                  break
          return res
  ```

19. 判断一棵树是否为二叉搜索树和二叉完全树

- 二叉搜索树+DFS+最大最小值+二叉完全树+BFS+看'#'存在情况

-
  ```
  class Solution:
      def judgeIt(self , root: TreeNode) -> List[bool]:
          # write code here
          max_val = float('inf')
          min_val = float('-inf')
          rel1 = self.BST(root, max_val, min_val)
          rel2 = self.BCT(root)
          return [rel1, rel2]
      def BST(self, root, max_val, min_val):
          if not root:
              return True
          if min_val < root.val < max_val:
              return self.BST(root.left, root.val, min_val) and self.BST(root.right, max_val, root.val)
          else:
              return False
      def BCT(self, root):
          if not root:
              return True
          queue = [root]
          temp = []
          while queue:
              cur = queue.pop(0)
              if cur:
                  temp.append(cur.val)
                  queue.append(cur.left)
                  queue.append(cur.right)
              else:
                  temp.append('#')
          while temp[-1] == '#':
              temp.pop()
          return not '#' in temp
  ```

20. 把字符串翻译成整数

- 

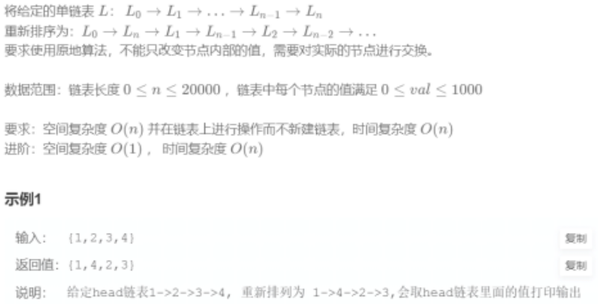
  写一个函数 StrToInt，实现把字符串转换成整数这个功能。不能使用 atoi 或者其他类似的库函数。传入的字符串可能有以下部分组成:
  1.若干空格
  2.（可选）一个符号字符（'+' 或 '-'）
  3. 数字，字母，符号，空格组成的字符串表达式
  4. 若干空格

  转换算法如下:
  1.去掉无用的前导空格
  2.第一个非空字符为+或者-号时，作为该整数的正负号，如果没有符号，默认为正数
  3.判断整数的有效部分：
  3.1 确定符号位之后，与之后面尽可能多的连续数字组合起来成为有效整数数字，如果没有有效的整数部分，那么直接返回0
  3.2 将字符串前面的整数部分取出，后面可能会存在存在多余的字符(字母，符号，空格等)，这些字符可以被忽略，它们对于函数不应该造成影响
  3.3 整数超过 32 位有符号整数范围 $[-2^{31}, 2^{31}-1]$，需要截断这个整数，使其保持在这个范围内。具体来说，小于 -2^{31}的整数应该被调整为 -2^{31}，大于 $2^{31}-1$ 的整数应该被调整为 $2^{31}-1$
  4.去掉无用的后导空格
- 循换遍历+判断边界
-
  ```
  class Solution:
      def StrToInt(self , s: str) -> int:
          # write code here
          s = s.strip()
          if not s:
              return 0
          res = 0
          sign = 1
          start = 0
          MAX_INT = 2**31-1
          MIN_INT = -1 * 2**31
          bound = MAX_INT // 10
          if s[0] == '-':
              sign = -1
          if s[0] == '+' or s[0] == '-':
              start = 1
          for item in s[start:]:
              if not '0' <= item <= '9':
                  break
              else:
                  if res > bound or res == bound and item > '7':
                      if sign == 1:
                          return MAX_INT
                      else:
                          return MIN_INT
                  else:
                      res = res * 10 + int(item)
          if sign == 1:
              return res
          else:
              return -1 * res
  ```

21. 反转数字
    - 转换成字符串+反转+判断越界
    -
      ```
      class Solution:
          def reverse(self , x: int) -> int:
              # write code here
              x = list(str(x))
              sign = 1
              INT_MAX = 2**31-1
              if x[0] == '-':
                  sign = -1
                  x = x[1:]
              x.reverse()
              if len(x) > 1 and x[0] == '0':
                  del x[0]
              if int(''.join(x)) > INT_MAX:
                  return 0
              else:
                  return sign * int(''.join(x))
      ```
22. 二叉树的最大路径和
    - 能否向上合并+能合并（根，根左，根右）+不能合并（根左右，左，右）+能合并递归+不能合并（全局变量）
    -
      ```
      class Solution:
          def maxPathSum(self , root: TreeNode) -> int:
              # write code here
              self.rel1 = float('-inf')
              def recur(root):
                  if not root:
                      return float('-inf')
                  left_val = recur(root.left)
                  right_val = recur(root.right)
                  self.rel1 = max(self.rel1, left_val, right_val, root.val+left_val+right_val)
                  return max(root.val, root.val+left_val, root.val+right_val)
              rel2 = recur(root)
              return max(self.rel1, rel2)
      ```
23. 括号生成

- 当前位置+左括号个数+右括号个数
-
```
class Solution:
    def generateParenthesis(self , n: int) -> List[str]:
        # write code here
        res = []
        cur = []
        self.left = 0
        self.right = 0
        def recur(x, l):
            if x == l:
                res.append(''.join(cur))
                return
            if self.left < n:
                cur.append('(')
                self.left += 1
                recur(x+1, l)
                cur.pop()
                self.left -= 1
            if self.right < self.left:
                cur.append(')')
                self.right += 1
                recur(x+1, l)
                cur.pop()
                self.right -= 1
        recur(0, 2*n)
        return res
```

24. 重排链表

- 

将给定的单链表 L： $L_0 \to L_1 \to \ldots \to L_{n-1} \to L_n$
重新排序为： $L_0 \to L_n \to L_1 \to L_{n-1} \to L_2 \to L_{n-2} \to \ldots$
要求使用原地算法，不能只改变节点内部的值，需要对实际的节点进行交换。

数据范围：链表长度 $0 \le n \le 20000$ ，链表中每个节点的值满足 $0 \le val \le 1000$

要求：空间复杂度 $O(n)$ 并在链表上进行操作而不新建链表，时间复杂度 $O(n)$
进阶：空间复杂度 $O(1)$ ， 时间复杂度 $O(n)$

示例1

输入： {1,2,3,4}
返回值：{1,4,2,3}
说明： 给定head链表1->2->3->4， 重新排列为 1->4->2->3,会取head链表里面的值打印输出

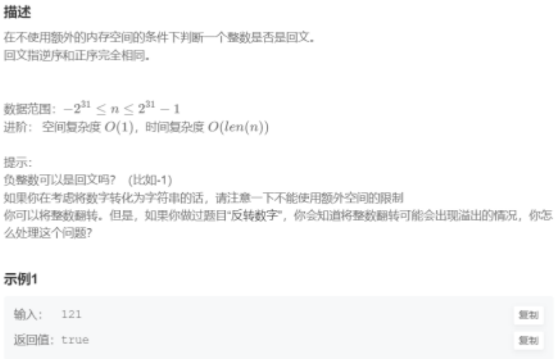
- 快慢指针将链表分为两部分+反转后半部分+交替合并两个链表
-
```
class Solution:
    def reorderList(self , head ):
        # write code here
        if not head or not head.next:
            return head
        slow = fast = head
        while fast and fast.next:
            slow = slow.next
            fast = fast.next.next
        temp = slow.next
        slow.next = None
        temp = self.reverses(temp)
        dummy = ListNode(0)
        dummy.next = head
        p = dummy
        while head and temp:
            p.next = head
            head = head.next
            p = p.next
            p.next = temp
            temp = temp.next
            p = p.next
        p.next = head if head else temp
        return dummy.next
    def reverses(self, head):
        pre = None
        cur = head
        while cur:
            temp = cur.next
            cur.next = pre
            pre = cur
            cur = temp
        return pre
```

25. 加起来和为目标值的组合（二）

- 回溯法+def(start, diff)
-
```
class Solution:
    def combinationSum2(self , num: List[int], target: int) -> List[List[int]]:
        # write code here
        if not num:
            return []
        num.sort()
        res = []
        cur = []
        def recur(start, diff):
            if diff < 0:
                return
            if diff == 0:
                res.append(cur[:])
                return
            for i in range(start, len(num)):
                if i > start and num[i] == num[i-1]:
                    continue
                cur.append(num[i])
                recur(i+1, diff-num[i])
                cur.pop()
        recur(0, target)
    return res
```

26. 最长公共前缀
    - 找最短字符串+判断前缀是否在其他字符串中

    -
      ```
      class Solution:
          def longestCommonPrefix(self , strs: List[str]) -> str:
              # write code here
              if not strs:
                  return ''
              if len(strs) == 1:
                  return strs[0]
              res = ''
              min_str = strs[0]
              for item in strs:
                  if len(item) < len(min_str):
                      min_str = item
              for i in range(1, len(min_str)+1):
                  cur = min_str[:i]
                  for item in strs:
                      if cur not in item:
                          return res
                  res = cur
              return res
      ```

27. 回文数字
    - 

      描述

      在不使用额外的内存空间的条件下判断一个整数是否是回文。
      回文指逆序和正序完全相同。

      数据范围：$-2^{31} \le n \le 2^{31}-1$
      进阶： 空间复杂度 $O(1)$，时间复杂度 $O(len(n))$

      提示:
      负整数可以是回文吗？ (比如-1)
      如果你在考虑将数字转化为字符串的话，请注意一下不能使用额外空间的限制
      你可以将整数翻转。但是，如果你做过题目“反转数字”，你会知道将整数翻转可能会出现溢出的情况，你怎么处理这个问题?

      示例1

      输入： 121
      返回值：true

    - 只反转一半+取余求和

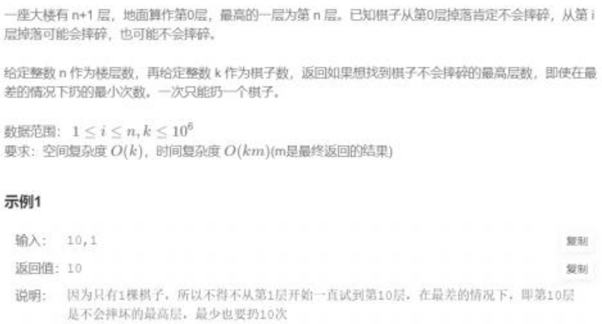
    -
      ```
      class Solution:
          def isPalindrome(self , x: int) -> bool:
              # write code here
              if x < 0:
                  return False
              if x == 0:
                  return True
              temp = 0
              while temp < x:
                  cur = x % 10
                  temp = temp * 10 + cur
                  x = x // 10
              if x == temp or x == temp // 10:
                  return True
              else:
                  return False
      ```

28. 二分查找-II
    - 找目标的左边界

    -
      ```
      class Solution:
          def search(self , nums: List[int], target: int) -> int:
              # write code here
              if not nums:
                  return -1
              left = 0
              right = len(nums) - 1
      ```

```
        while left <= right:
            mid = left + (right - left) // 2
            if nums[mid] > target:
                right = mid - 1
            elif nums[mid] < target:
                left = mid + 1
            elif nums[mid] == target:
                right = mid - 1
        if left >= len(nums) or nums[left] != target:
            return -1
        else:
            return left
```

29. 丢棋子问题

- 

一座大楼有 n+1 层，地面算作第0层，最高的一层为第 n 层。已知棋子从第0层掉落肯定不会摔碎，从第 i 层掉落可能会摔碎，也可能不会摔碎。

给定整数 n 作为楼层数，再给定整数 k 作为棋子数，返回如果想找到棋子不会摔碎的最高层数，即使在最差的情况下扔的最小次数。一次只能扔一个棋子。

数据范围：$1 \le i \le n, k \le 10^6$
要求：空间复杂度 $O(k)$，时间复杂度 $O(km)$(m是最终返回的结果)

示例1

输入： 10,1 复制

返回值： 10 复制

说明： 因为只有1棵棋子，所以不得不从第1层开始一直试到第10层，在最差的情况下，即第10层是不会摔坏的最高层，最少也要扔10次

- 看左边和左上角+dp[i][j]表示i个棋子扔j次，最多测多少层楼

-
  ```
  class Solution:
      def solve(self , n: int, k: int) -> int:
          # write code here
          if n == 1:
              return 1
          # dp[i][j]表示i次操作，j个棋子可以验证的最高楼层数
          dp = [[0 for _ in range(k+1)] for _ in range(n+1)]
          for j in range(1, k+1):
              dp[1][j] = 1
          res = -1
          for i in range(2, n+1):
              for j in range(1, k+1):
                  dp[i][j] = dp[i-1][j-1] + dp[i-1][j] + 1
              if dp[i][j] >= n:
                  res = i
                  break
          return res
  ```

-
  ```
  class Solution:
      def solve(self , n: int, k: int) -> int:
          # write code here
          if n < 1 or k < 1:
              return 0
          if k == 1:
              return n
          import math
          b = math.log2(n) + 1
          if k >= b:
              return int(b)
          dp = [0] * k
          res = 0
          while True:
              res += 1
              pre = 0
              for i in range(k):
                  tmp = dp[i]
                  dp[i] = dp[i] + pre + 1
                  pre = tmp
                  if dp[i] >= n:
                      return res
  ```

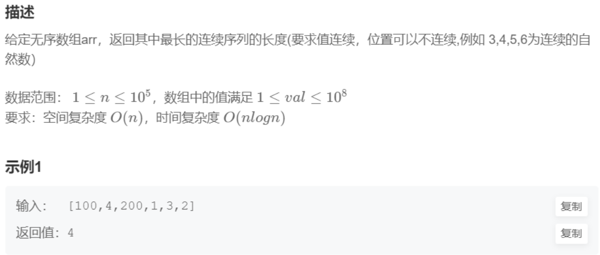
30. 二叉搜索树的第K个节点

- 中序搜索+返回第K小

-
  ```
  class Solution:
      def KthNode(self , proot: TreeNode, k: int) -> int:
          # write code here
          if not proot:
              return -1
          rel = self.inorder(proot)
          if len(rel) < k or k <= 0:
              return -1
          return rel[k-1]
      def inorder(self, root):
          if not root:
              return []
  ```

```
        stack = [root]
        rel = []
        while stack:
            cur = stack.pop()
            if isinstance(cur, TreeNode):
                stack.append(cur.right)
                stack.append(cur.val)
                stack.append(cur.left)
            elif isinstance(cur, int):
                rel.append(cur)
        return rel
```

31. 数组中的最长连续子序列

- 

  描述

  给定无序数组arr，返回其中最长的连续序列的长度(要求值连续，位置可以不连续,例如 3,4,5,6为连续的自然数)

  数据范围：$1 \le n \le 10^5$，数组中的值满足 $1 \le val \le 10^8$
  要求：空间复杂度 $O(n)$，时间复杂度 $O(nlogn)$

  示例1

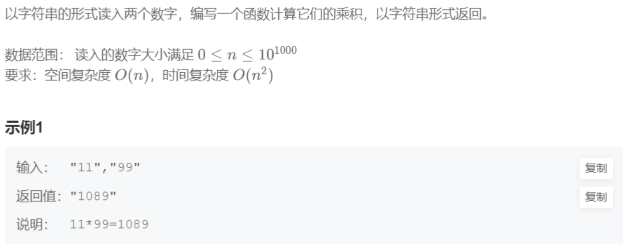
  输入： [100,4,200,1,3,2]
  返回值：4

- 数组set去重+排序+滑动窗口判断是否为连续子序列
-
```
class Solution:
    def MLS(self , arr: List[int]) -> int:
        # write code here
        if not arr:
            return 0
        res = 0
        left = 0
        right = 0
        arr = list(set(arr))
        arr.sort()
        while right < len(arr):
            if arr[right]-arr[left] == right - left:
                res = max(res, right-left+1)
                right += 1
            else:
                left += 1
        return res
```

32. 最大数

- 字符串快排
-
```
class Solution:
    def solve(self , nums: List[int]) -> str:
        # write code here
        if not nums:
            return ''
        for i in range(len(nums)):
            nums[i] = str(nums[i])
        if nums.count('0') == len(nums):
            return '0'
        def recur(arr):
            if len(arr) < 2:
                return arr
            pivot = arr[0]
            left = []
            right = []
            for i in range(1, len(arr)):
                if arr[i]+pivot > pivot+arr[i]:
                    right.append(arr[i])
                else:
                    left.append(arr[i])
            return recur(left) + [pivot] + recur(right)
        nums = recur(nums)
        return ''.join(nums[::-1])
```

33. 大数乘法



以字符串的形式读入两个数字，编写一个函数计算它们的乘积，以字符串形式返回。

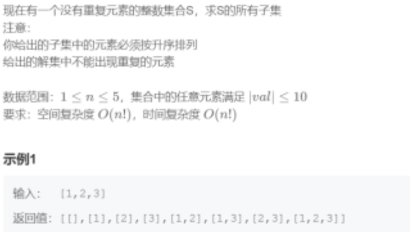
数据范围： 读入的数字大小满足 $0 \le n \le 10^{1000}$
要求：空间复杂度 $O(n)$，时间复杂度 $O(n^2)$

- **示例1**

  输入： "11","99"
  返回值："1089"
  说明： 11*99=1089

- 字符串乘法+拆分求和
-
```
class Solution:
    def solve(self, s, t):
        if s == '0' or t == '0':
            return '0'
        # 11*99:可以拆解为从末尾开始 1*99 + 10 * 99    123*99 = 3*99+20*99+100*99
        dig = 0
        res = 0
        i = len(s) - 1
        while i >= 0:
            temp = int(s[i]) * (10 ** dig) * int(t)
            res += temp
            dig += 1
            i -= 1
        return str(res)
```

34. 集合的所有子集（一）

- 

  现在有一个没有重复元素的整数集合S，求S的所有子集
  注意：
  你给出的子集中的元素必须按升序排列
  给出的解集中不能出现重复的元素

  数据范围：$1 \le n \le 5$，集合中的任意元素满足 $|val| \le 10$
  要求：空间复杂度 $O(n!)$，时间复杂度 $O(n!)$

  示例1

  ```
  输入：  [1,2,3]
  返回值：[[],[1],[2],[3],[1,2],[1,3],[2,3],[1,2,3]]
  ```

- 回溯树+有顺序之分+start表示当前遍历树的第start层

```
class Solution:
    def subsets(self , S: List[int]) -> List[List[int]]:
        # write code here
        if not S:
            return []
        res = []
        cur = []
        def recur(start):
            res.append(cur[:])
            if len(cur) == len(S):
                return
            for i in range(start, len(S)):
                cur.append(S[i])
                recur(i + 1)
                cur.pop()
        recur(0)
        return res
```

35. 链表中倒数第K个节点

- 快慢指针

```
class Solution:
    def FindKthToTail(self , pHead: ListNode, k: int) -> ListNode:
        # write code here
        if not pHead:
            return
        slow = fast = pHead
        for _ in range(k):
            if not fast:
                return None
            fast = fast.next
        while fast:
            slow = slow.next
            fast = fast.next
        return slow
```

36. 多叉树的直径

- 思路
- DFS
  - 先从初始点出发，一直探索到最底层，也就是离初始点最远的一个端点。
  - 这个端点必定是我们最后所求的最长距离的其中一个端点
  - 找到这个端点，然后从这个端点出发，探索距离此点最远的点

```
class Solution:
    def solve(self , n: int, Tree_edge: List[Interval], Edge_value: List[int]) -> int:
        # write code here
        import collections
        graph = collections.defaultdict(dict)
        for i in range(len(Tree_edge)):
            s = Tree_edge[i].start
            e = Tree_edge[i].end
            graph[s][e] = Edge_value[i]
            graph[e][s] = Edge_value[i]
        self.max_dis = 0
        self.first = 0
        self.second = 0
        def recur(start, pre, dis):
```

```
    if len(graph[start]) == 1 and pre in graph[start]:
        if self.max_dis < dis:
            self.max_dis = dis
            self.second = start
    else:
        for item in graph[start]:
            if item != pre:
                recur(item, start, dis+graph[start][item])
recur(self.first, -1, 0)
recur(self.second, -1, 0)
return self.max_dis
```