Assignment #5: 链表、栈、队列和归并排序

Updated 1348 GMT+8 Mar 17, 2025

2025 spring, Complied by <mark>汤伟杰，信息管理系</mark>

> **说明：**
>
> 1. **解题与记录：**
>
>   对于每一个题目，请提供其解题思路（可选），并附上使用 Python 或 C++编写的源代码（确保已在 OpenJudge， Codeforces，LeetCode 等平台上获得 Accepted）。请将这些信息连同显示“Accepted”的截图一起填写到下方的作业模板中。（推荐使用 Typora https://typoraio.cn 进行编辑，当然你也可以选择 Word。）无论题目是否已通过，请标明每个题目大致花费的时间。
>
> 2. **提交安排：**提交时，请首先上传 PDF 格式的文件，并将.md 或.doc 格式的文件作为附件上传至右侧的“作业评论”区。确保你的 Canvas 账户有一个清晰可见的头像，提交的文件为 PDF 格式，并且“作业评论”区包含上传的.md 或.doc 附件。
>
> 3. **延迟提交：**如果你预计无法在截止日期前提交作业，请提前告知具体原因。这有助于我们了解情况并可能为你提供适当的延期或其他帮助。
>
> 请按照上述指导认真准备和提交作业，以保证顺利完成课程要求。

1. 题目

LC21.合并两个有序链表

linked list, https://leetcode.cn/problems/merge-two-sorted-lists/

思路：

是从合并 K 个有序链表中学习的两种思路：

第一种是用**归并排序**的第二个函数 merge 的思路，题目给出的两个链表实际上就相当于是归并排序中要进行合并的 l 列表和 r 列表，于是只需要逐个比较加入答案即可；

第二种是用**堆**的思路，先存入头节点的数值，然后不断弹出最小元素形成答案。

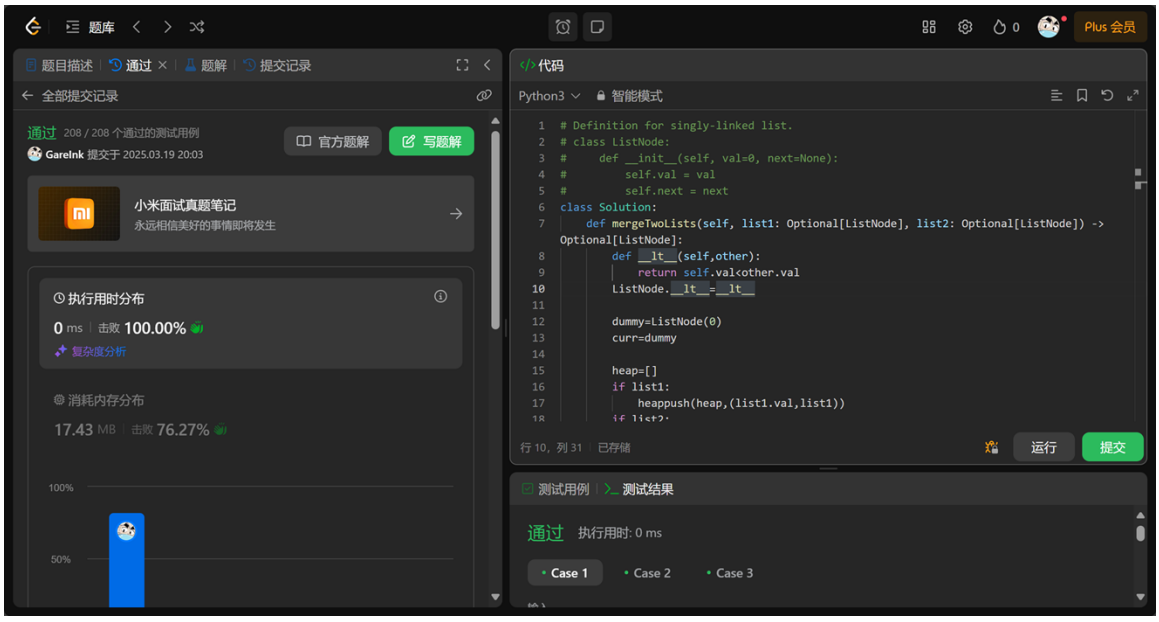
两种都用到设置 dummy 头节点的小技巧。不过第一种对于链表数目较多时有时间优势（如合并 K 个链表），第二种对于链表数目少且链表长度较长时有时间优势（比如本题用堆的时间比归并排序的时候短）。

代码：

```
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def mergeTwoLists(self, list1: Optional[ListNode], list2:
```

```
Optional[ListNode]) -> Optional[ListNode]:
        def __lt__(self,other):
            return self.val<other.val
        ListNode.__lt__=__lt__

        dummy=ListNode(0)
        curr=dummy

        heap=[]
        if list1:
            heappush(heap,(list1.val,list1))
        if list2:
            heappush(heap,(list2.val,list2))

        while heap:
            v,l=heappop(heap)
            curr.next=ListNode(v)
            if l.next:
                heappush(heap,(l.next.val,l.next))
            curr=curr.next
        return dummy.next

        # dummy=ListNode(0)
        # curr=dummy
        # while list1 and list2:
        #     if list1.val<=list2.val:
        #         curr.next,list1=list1,list1.next
        #     else:
        #         curr.next,list2=list2,list2.next
        #     curr=curr.next

        # curr.next=list1 if list1 else list2
        # return dummy.next
```

代码运行截图 <mark>（至少包含有"Accepted"）</mark>



LC234.回文链表

linked list, https://leetcode.cn/problems/palindrome-linked-list/

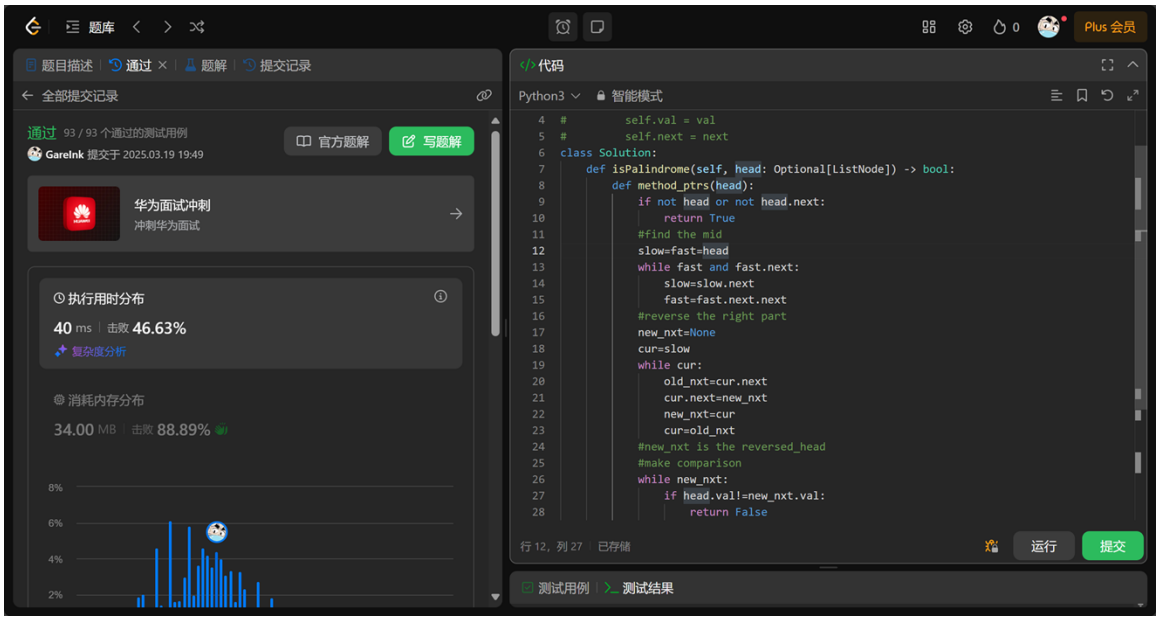
<mark>请用快慢指针实现。</mark>

思路：用快慢指针找到中间节点，然后反转右边的链表，再依次与左边的链表元素一一比较。

代码：

```
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def isPalindrome(self, head: Optional[ListNode]) -> bool:
        def method_ptrs(head):
            if not head or not head.next:
                return True
            #find the mid
            slow=fast=head
            while fast and fast.next:
                slow=slow.next
                fast=fast.next.next
            #reverse the right part
            new_nxt=None
            cur=slow
            while cur:
                old_nxt=cur.next
                cur.next=new_nxt
                new_nxt=cur
                cur=old_nxt
            #new_nxt is the reversed_head
            #make comparison
            while new_nxt:
                if head.val!=new_nxt.val:
                    return False
                head=head.next
                new_nxt=new_nxt.next
            return True

        # return method_stack(head)
        return method_ptrs(head)
```

代码运行截图 <mark>（至少包含有"Accepted"）</mark>



LC1472.设计浏览器历史记录

doubly-lined list, https://leetcode.cn/problems/design-browser-history/

<mark>请用双链表实现。</mark>

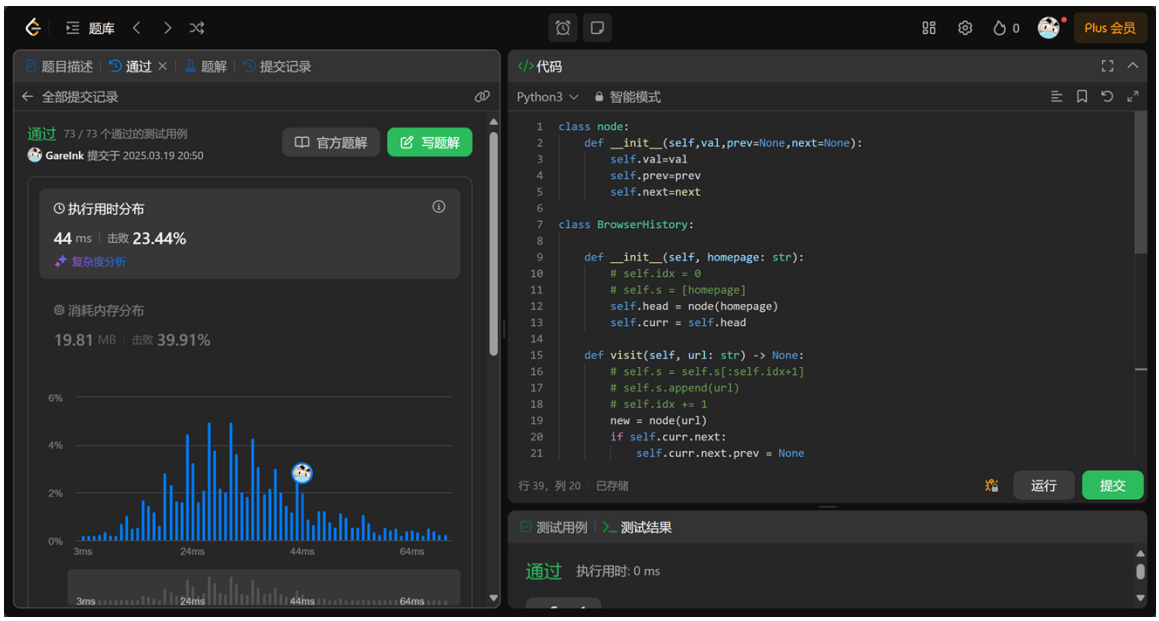
思路：构造一个双端链表，每次 visit 就把 curr 节点后方全部断掉连上这个新的 url，每次 back 或者 forward 就直接用 curr.prev 和 curr.next 即可，很方便。

代码：

```
# 注释部分是未使用链表的做法

class node:
    def __init__(self,val,prev=None,next=None):
        self.val=val
        self.prev=prev
        self.next=next

class BrowserHistory:

    def __init__(self, homepage: str):
        # self.idx = 0
        # self.s = [homepage]
        self.head = node(homepage)
        self.curr = self.head

    def visit(self, url: str) -> None:
        # self.s = self.s[:self.idx+1]
        # self.s.append(url)
        # self.idx += 1
        new = node(url)
        if self.curr.next:
            self.curr.next.prev = None
        self.curr.next = new
```

```
        new.prev = self.curr
        self.curr = new

    def back(self, steps: int) -> str:
        # self.idx = max(0, self.idx - steps)
        # return self.s[self.idx]
        while steps and self.curr.prev:
            self.curr = self.curr.prev
            steps -= 1
        return self.curr.val

    def forward(self, steps: int) -> str:
        # self.idx = min(len(self.s) - 1, self.idx + steps)
        # return self.s[self.idx]
        while steps and self.curr.next:
            self.curr = self.curr.next
            steps -= 1
        return self.curr.val


# Your BrowserHistory object will be instantiated and called as such:
# obj = BrowserHistory(homepage)
# obj.visit(url)
# param_2 = obj.back(steps)
# param_3 = obj.forward(steps)
```

代码运行截图 <mark>（至少包含有"Accepted"）</mark>



24591: 中序表达式转后序表达式

stack, http://cs101.openjudge.cn/practice/24591/

思路：

设置运算符等级，遇到运算符要把 stack 中等级高于等于其的运算符弹出，遇到右括号要一直匹配到左括号。

代码：

```
d={'+':1,'-':1,'*':2,'/':2,'(':0}
for _ in range(int(input())):
    s=input()
    stack=[]
    ans=[]
    i=0
    while i<len(s):
        # print(stack)
        if s[i] in '+-*/':
            while stack and d[stack[-1]]>=d[s[i]]:
                ans.append(stack.pop())
            stack.append(s[i])
            i+=1
        elif s[i]=='(':
            stack.append('(')
            i+=1
        elif s[i]==')':
            while stack and stack[-1]!='(':
                ans.append(stack.pop())
            stack.pop()
            i+=1
        else:
            curr=s[i]
            i+=1
            while i<len(s) and s[i] not in '+-*/()':
                curr+=s[i]
                i+=1
            ans.append(curr)
    while stack:
        ans.append(stack.pop())
    print(*ans)
```

代码运行截图 <mark>（至少包含有"Accepted"）</mark>

03253: 约瑟夫问题 No.2

queue, http://cs101.openjudge.cn/practice/03253/

<mark>请用队列实现。</mark>

用 deque 的 popleft 和 append 来不断循环，很方便。

代码：

```
from collections import deque
while True:
    n,p,m=map(int,input().split())
    if {n,p,m}=={0}:
        break
    ans=[]
    q=deque()
    for i in range(1,n+1):
        q.append(i)
    for _ in range(p-1):
        q.append(q.popleft())

    while q:
        for _ in range(m-1):
            q.append(q.popleft())
        ans.append(q.popleft())

    print(','.join(map(str,ans)))
```

代码运行截图 <mark>（至少包含有"Accepted"）</mark>

#48633409提交状态

查看 提交 统计 提问

状态: Accepted

源代码

```
from collections import deque
while True:
    n,p,m=map(int,input().split())
    if {n,p,m}=={0}:
        break
    ans=[]
    q=deque()
    for i in range(1,n+1):
        q.append(i)
    for _ in range(p-1):
        q.append(q.popleft())

    while q:
        for _ in range(m-1):
            q.append(q.popleft())
        ans.append(q.popleft())

    print(','.join(map(str,ans)))
```

基本信息

#: 48633409
题目: 03253
提交人: 24n2400016635
内存: 3648kB
时间: 35ms
语言: Python3
提交时间: 2025-03-19 20:35:51

English 帮助 关于

20018: 蚂蚁王国的越野跑

merge sort, http://cs101.openjudge.cn/practice/20018/

思路：

与每日选做的 oj_Ultra-QiuckSort_02299 的思路完全一致，一个比较巧妙的计算方法是：每当 left[i]>right[j]的时候，由于此时 left 和 right 都是已经升序了的列表，那么在 i 位置之后的所有 left 元素的值都会比当前的 right[j]的值大，那么 res 就直接加上 len(l)-i，这样可以避免每次只是加一次。

代码：

```
#pylint:skip-file
def mergeSort(s):
    n=len(s)
    if n<=1:
        return s
    mid=n//2
    left=mergeSort(s[:mid])
    right=mergeSort(s[mid:])
    return merge(left,right)

def merge(l,r):
    global res
    i,j=0,0
    ans=[]
    while i<len(l) and j<len(r):
        if l[i]<=r[j]:
            ans.append(l[i])
            i+=1
        else:
            ans.append(r[j])
            res+=len(l)-i
```

```
            j+=1
    ans.extend(l[i:])
    ans.extend(r[j:])
    return ans

n=int(input())
s=[]
for _ in range(n):
    s.append(int(input()))
s.reverse()
res=0
mergeSort(s)
print(res)
```

代码运行截图 <mark>（至少包含有"Accepted"）</mark>

状态: Accepted

源代码

```
#pylint:skip-file
def mergeSort(s):
    n=len(s)
    if n<=1:
        return s
    mid=n//2
    left=mergeSort(s[:mid])
    right=mergeSort(s[mid:])
    return merge(left,right)

def merge(l,r):
    global res
    i,j=0,0
    ans=[]
    while i<len(l) and j<len(r):
        if l[i]<=r[j]:
            ans.append(l[i])
            i+=1
        else:
            ans.append(r[j])
            res+=len(l)-i
            j+=1
    ans.extend(l[i:])
    ans.extend(r[j:])
    return ans

n=int(input())
s=[]
for _ in range(n):
    s.append(int(input()))
s.reverse()
res=0
mergeSort(s)
print(res)
```

基本信息

#: 48604300
题目: 20018
提交人: 24n2400016635
内存: 10012kB
时间: 717ms
语言: Python3
提交时间: 2025-03-17 15:03:02

English 帮助 关于

2. 学习总结和收获

<mark>如果发现作业题目相对简单，有否寻找额外的练习题目，如“数算 2025spring 每日选做”、LeetCode、Codeforces、洛谷等网站上的题目。</mark>

最近做题感觉要不是模板题，要不是一点都不会的题（如：合并 K 个有序链表），要不就是会但是做不出来的题（如：每日选做的 palindrome 那道 dp）。平常也在做 cf 和力扣的题，这种感觉更深了。但是好处是每次遇到一个不会的题，通过看题解的思路真的可以学到真正的新知识，很充实也很开心（比如 dp 把二维数组仅保留为两列数组的转换思路、

利用模板归并排序和堆的知识去解决有序链表合并……）每日选做题目难度有些好大，做起来很费时间，目前有一定差距要补齐了！

喜报：cf 绿名了（但是都是通过做 div3 或者 div4 的前几道简单题上分的。。力扣的周赛常常只能做出来 1 道，我好菜啊服了）