Assignment #6: 回溯、树、双向链表和哈希表

Updated 1526 GMT+8 Mar 22, 2025

2025 spring, Complied by 郑涵予 物理学院

> **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

LC46.全排列

backtracking, https://leetcode.cn/problems/permutations/

思路：

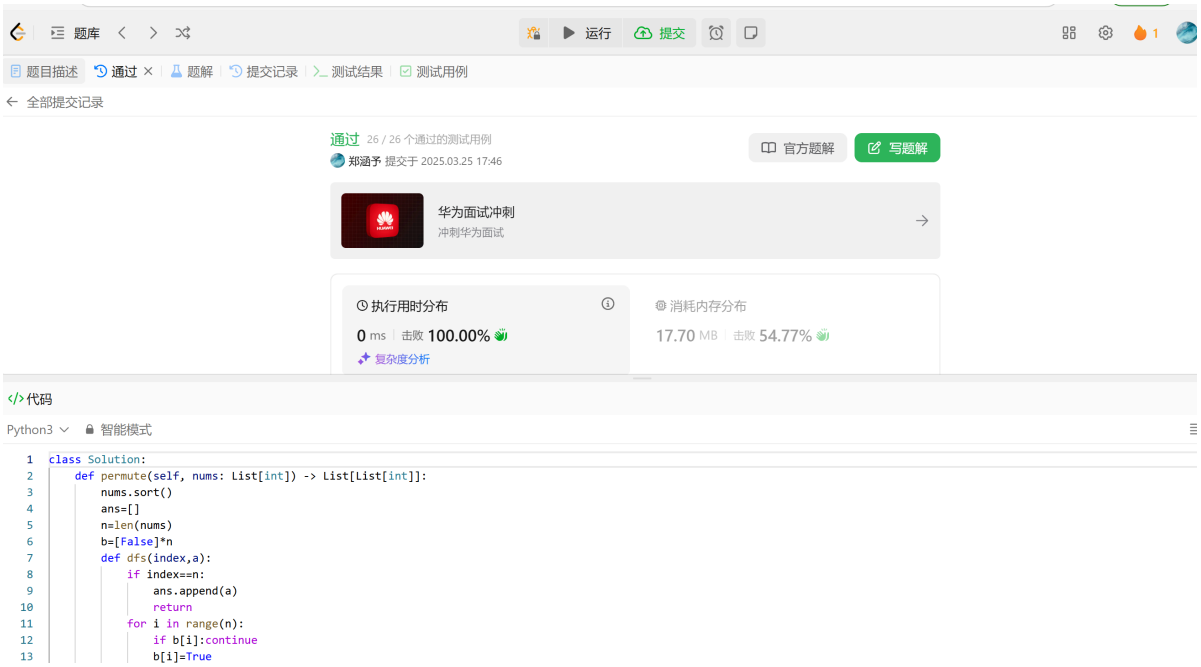
直接回溯搜索即可（用时约5min)

代码：

```
class Solution:
    def permute(self, nums: List[int]) -> List[List[int]]:
        nums.sort()
        ans=[]
        n=len(nums)
        b=[False]*n
        def dfs(index,a):
```

```
            if index==n:
                ans.append(a)
                return
            for i in range(n):
                if b[i]:continue
                b[i]=True
                a.append(nums[i])
                dfs(index+1,a.copy())
                a.pop()
                b[i]=False
        dfs(0,[])
        return ans
```

代码运行截图 （至少包含有"Accepted"）



LC79: 单词搜索

backtracking, https://leetcode.cn/problems/word-search/

思路：

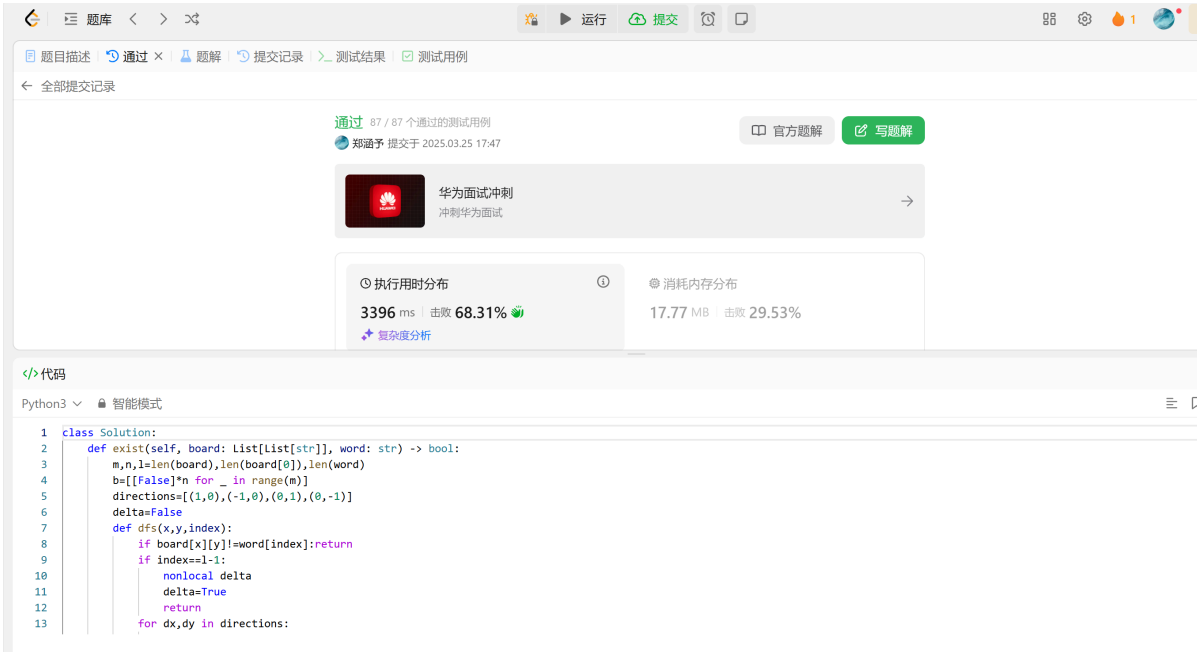
可以使用dfs进行搜索，在搜索过程中储存当前需要的字母在word中的索引（用时约10min)

代码：

```
class Solution:
    def exist(self, board: List[List[str]], word: str) -> bool:
        m,n,l=len(board),len(board[0]),len(word)
        b=[[False]*n for _ in range(m)]
```

```
        directions=[(1,0),(-1,0),(0,1),(0,-1)]
        delta=False
        def dfs(x,y,index):
            if board[x][y]!=word[index]:return
            if index==l-1:
                nonlocal delta
                delta=True
                return
            for dx,dy in directions:
                nx,ny=x+dx,y+dy
                if 0<=nx<m and 0<=ny<n and not b[nx][ny]:
                    b[nx][ny]=True
                    dfs(nx,ny,index+1)
                    b[nx][ny]=False
        for i in range(m):
            for j in range(n):
                if board[i][j]==word[0]:
                    b[i][j]=True
                    dfs(i,j,0)
                    b[i][j]=False
                if delta: return True
        return False
```

代码运行截图 （至少包含有"Accepted"）



LC94.二叉树的中序遍历

dfs, https://leetcode.cn/problems/binary-tree-inorder-traversal/

思路：

经典递归（用时约2min)

代码：

```
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def inorderTraversal(self, root: Optional[TreeNode]) -> List[int]:
        a=[]
        def dfs(root):
            if not root:return
            dfs(root.left)
            a.append(root.val)
            dfs(root.right)
        dfs(root)
        return a
```

代码运行截图 （至少包含有"Accepted"）

LC102.二叉树的层序遍历

bfs, https://leetcode.cn/problems/binary-tree-level-order-traversal/

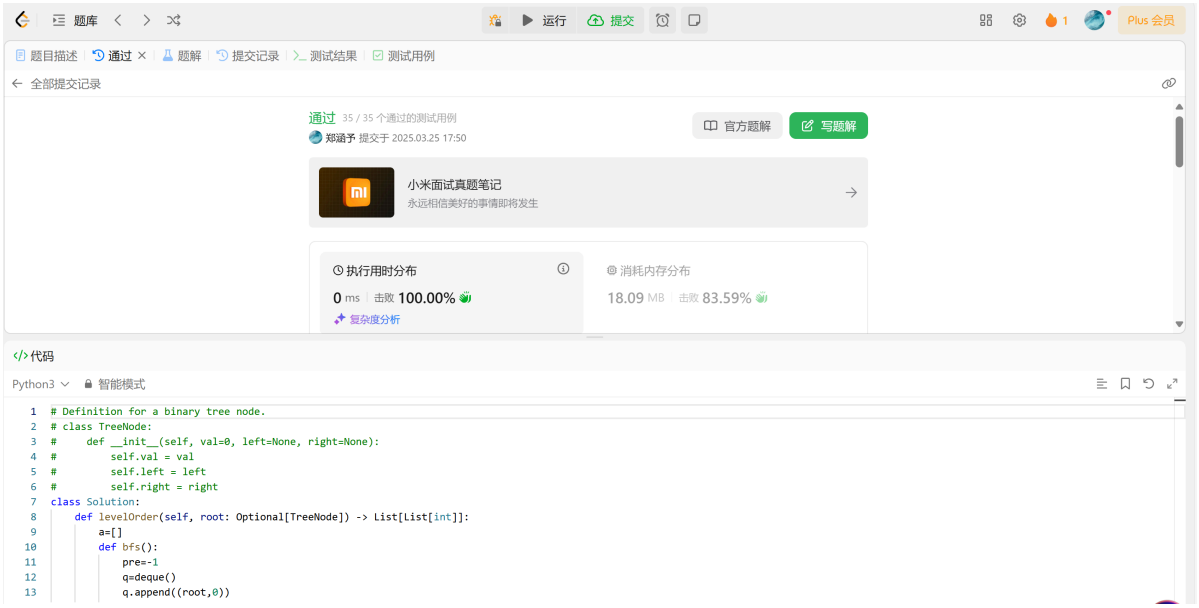
思路：

也是一道经典的题目，利用队列存储节点，标记层数即可（用时约5min）

代码：

```
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def levelOrder(self, root: Optional[TreeNode]) -> List[List[int]]:
        a=[]
        def bfs():
            pre=-1
            q=deque()
            q.append((root,0))
            while q:
                p=q.popleft()
                if not p[0]:continue
                if pre==p[1]:a[pre].append(p[0].val)
                else:
                    a.append([p[0].val])
                    pre+=1
                q.append((p[0].left,p[1]+1))
                q.append((p[0].right,p[1]+1))
        bfs()
        return a
```

代码运行截图 （至少包含有"Accepted"）



LC131.分割回文串

dp, backtracking, https://leetcode.cn/problems/palindrome-partitioning/

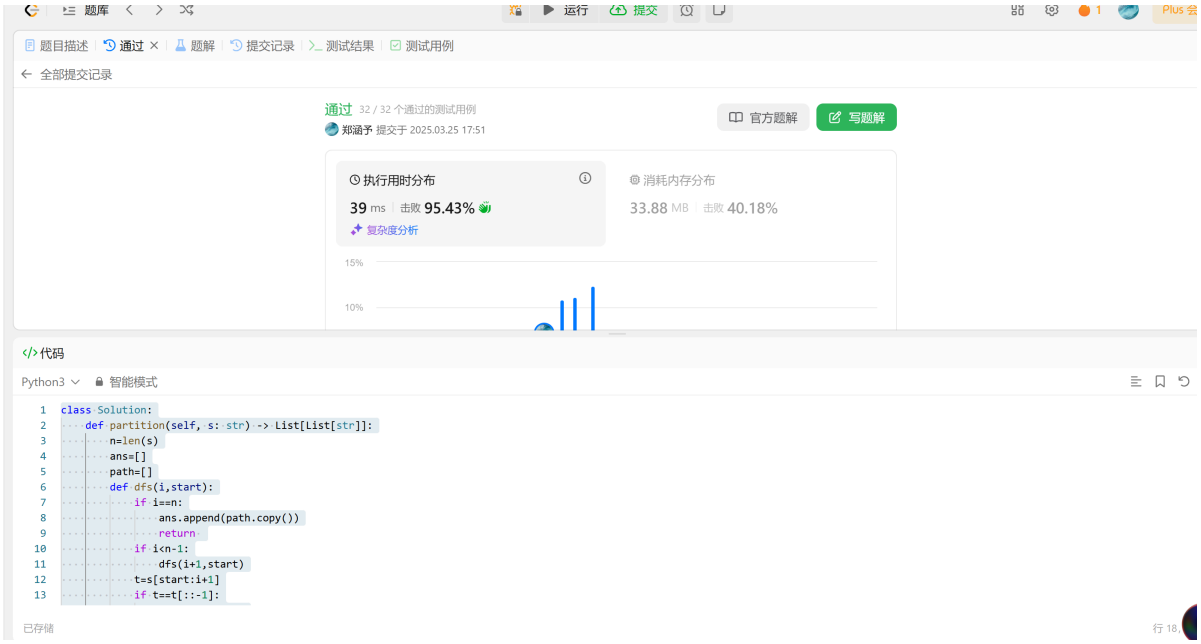
思路：

可以直接利用dfs,存储当前位置的索引以及上一个分割位置的索引，利用t==t[::-1]判断当前的索引能不能作为一个分割点。（用时约15min）

代码：

```
class Solution:
    def partition(self, s: str) -> List[List[str]]:
        n=len(s)
        ans=[]
        path=[]
        def dfs(i,start):
            if i==n:
                ans.append(path.copy())
                return
            if i<n-1:
                dfs(i+1,start)
            t=s[start:i+1]
            if t==t[::-1]:
                path.append(t)
                dfs(i+1,i+1)
                path.pop()
        dfs(0,0)
        return ans
```

代码运行截图 （至少包含有"Accepted"）



LC146.LRU缓存

hash table, doubly-linked list, https://leetcode.cn/problems/lru-cache/

思路:

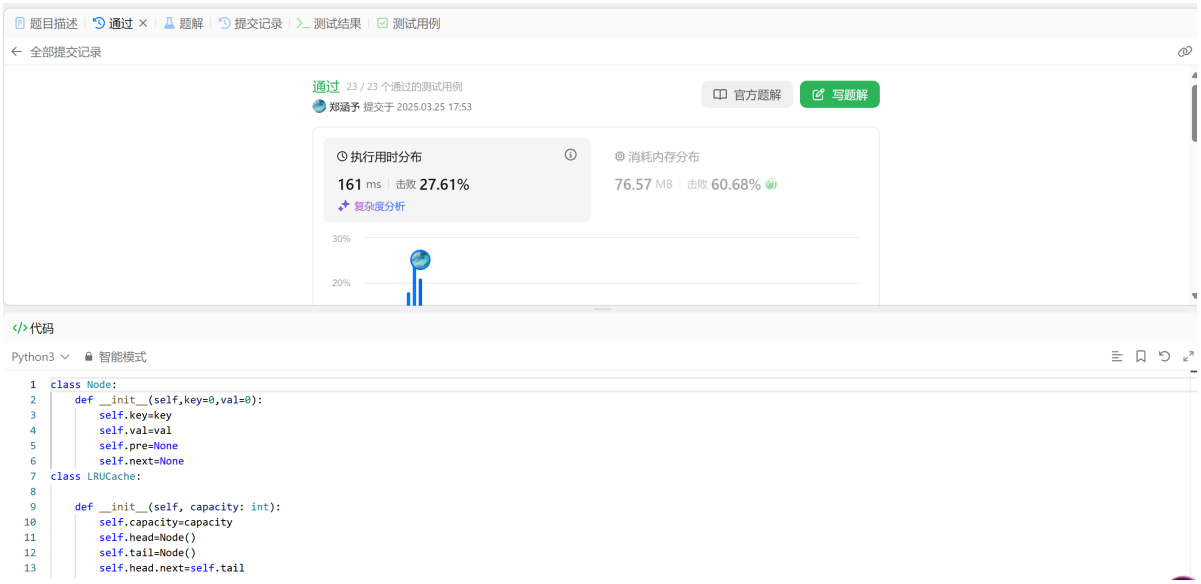
这次作业里最麻烦也是最难的一题，虽然看了提示知道要使用双向链表，但是在实现的时候还是会遇到各种问题。debug用了好长时间才AC.(用时约30min)

代码:

```
class Node:
    def __init__(self,key=0,val=0):
        self.key=key
        self.val=val
        self.pre=None
        self.next=None
class LRUCache:

    def __init__(self, capacity: int):
        self.capacity=capacity
        self.head=Node()
        self.tail=Node()
        self.head.next=self.tail
        self.tail.pre=self.head
        self.mp={}
        self.size=0

    def get(self, key: int) -> int:
        if key not in self.mp:
            return -1
        temp=self.mp[key]
```

```
        self.move_to_head(temp)
        return temp.val

    def put(self, key: int, value: int) -> None:
        if key in self.mp:
            self.mp[key].val=value
            temp=self.mp[key]
            self.move_to_head(temp)
        else:
            temp=Node(key,value)
            self.add_to_head(temp)
            self.mp[key]=temp
            self.size+=1
            if self.size>self.capacity:
                self.size-=1
                temp=self.remove_tail()
                del self.mp[temp.key]

    def add_to_head(self,temp):
        temp.pre=self.head
        temp.next=self.head.next
        self.head.next.pre=temp
        self.head.next=temp

    def remove_node(self,temp):
        temp.pre.next=temp.next
        temp.next.pre=temp.pre

    def move_to_head(self,temp):
        self.remove_node(temp)
        self.add_to_head(temp)

    def remove_tail(self):
        temp=self.tail.pre
        self.remove_node(temp)
        return temp



# Your LRUCache object will be instantiated and called as such:
# obj = LRUCache(capacity)
# param_1 = obj.get(key)
# obj.put(key,value)
```

代码运行截图 （至少包含有"Accepted"）



2. 学习总结和收获

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。

这周的力扣周赛比前几次的要简单不少（感觉这次的3还不如以前的2，4还不如以前的3），运气好AK了，但是并不能说明水平有多大提升（这次AK的共300多人，和以前AC3的人数差不多）。另外写作业题LRU缓存时发现对面向对象编程的方式还是不太熟练，比如经常忘记加self,而力扣又没有自动的语法检查，导致排查这个问题还花了不少时间，还要再练练。