Assignment #6: 回溯、树、双向链表和哈希表

Updated 1526 GMT+8 Mar 22, 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

LC46.全排列

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

思路：

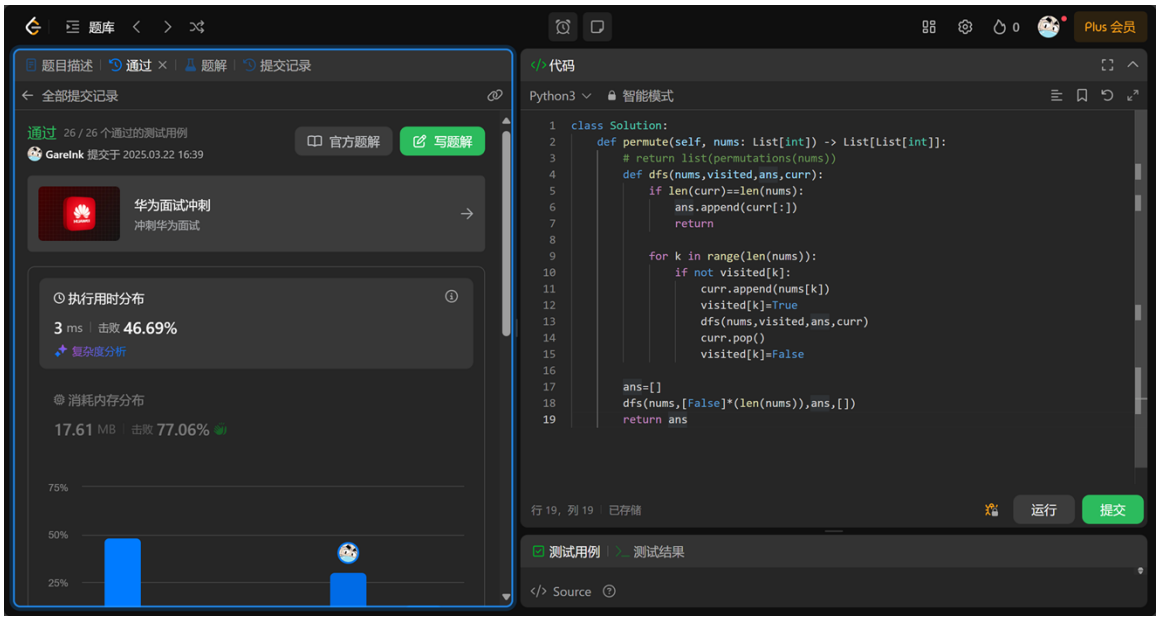
很经典很经典的 dfs，启蒙来自八皇后，坑点是 `ans.append(curr[:])`而不是 `ans.append(curr)`。

代码：

```
class Solution:
    def permute(self, nums: List[int]) -> List[List[int]]:
        # return list(permutations(nums))
        def dfs(nums,visited,ans,curr):
            if len(curr)==len(nums):
                ans.append(curr[:])
                return

            for k in range(len(nums)):
                if not visited[k]:
                    curr.append(nums[k])
                    visited[k]=True
                    dfs(nums,visited,ans,curr)
                    curr.pop()
                    visited[k]=False
```

```
        ans=[]
        dfs(nums,[False]*(len(nums)),ans,[])
        return ans
```

代码运行截图 <mark>（至少包含有"Accepted"）</mark>



LC79: 单词搜索

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

思路：

遍历棋盘的每一个位置，如果是单词的第一个字母就进入 dfs 的搜索，在 dfs 中设置一个 idx 索引来跟踪 word 的字母，之后就是很正常的搜索了。这道题能学到的东西是题解里面的两个优化：

一是统计棋盘所有字母的个数，如果其中出现在 word 中字母的个数少于 word 中需求的字母数量，那么可以直接返回 `False`；二是统计棋盘中的单词首字母和尾字母的个数，从个数少的一端进行 dfs。

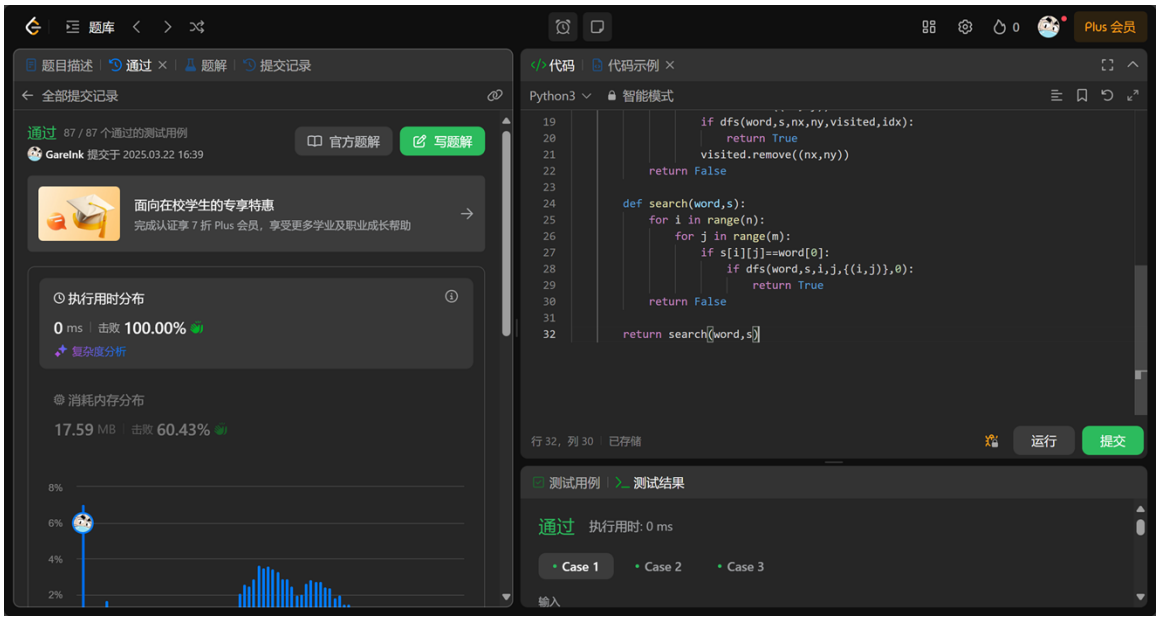
这道题由于只需要返回“能不能找到单词”，因此设置的 dfs 的返回值是布尔值，那么在每次调用函数本身的时候可以写成 `if dfs(word, s, nx, ny, visited, idx): return True`，这样的好处是：如果 dfs 到了单词末尾，那么会进入 if 语句的 return True，从而逐层返回 True，就不会进行 visited 的状态恢复了。很方便，这个写法也很巧妙。

代码：

```
class Solution:
    def exist(self, s: List[List[str]], word: str) -> bool:
        cnt = Counter(c for row in s for c in row)
        if not cnt >= Counter(word):  # 优化一
            return False
        if cnt[word[-1]] < cnt[word[0]]:  # 优化二
```

```
            word = word[::-1]

        n,m=len(s),len(s[0])
        dx,dy=[0,-1,1,0],[-1,0,0,1]
        def dfs(word,s,x,y,visited,idx):
            idx+=1
            if idx==len(word):
                return True
            for i in range(4):
                nx,ny=x+dx[i],y+dy[i]
                if 0<=nx<n and 0<=ny<m and word[idx]==s[nx][ny] and (nx,ny)
not in visited:
                    visited.add((nx,ny))
                    if dfs(word,s,nx,ny,visited,idx):
                        return True
                    visited.remove((nx,ny))
            return False

        def search(word,s):
            for i in range(n):
                for j in range(m):
                    if s[i][j]==word[0]:
                        if dfs(word,s,i,j,{(i,j)},0):
                            return True
            return False

        return search(word,s)
```

代码运行截图 <mark>（至少包含有"Accepted"）</mark>



LC94.二叉树的中序遍历

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

思路：

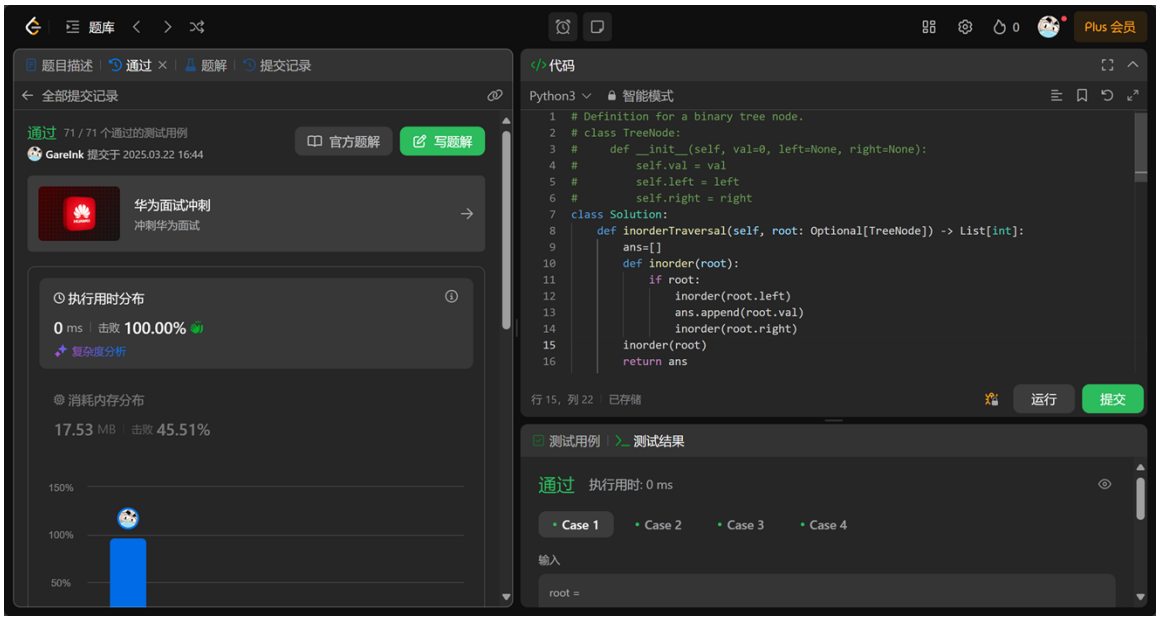
看了题解学了用 stack 模拟的“颜色填充法”，和递归的思路其实很相似。

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]:
        # 颜色填充法，新的是白色，放入栈时变为灰色，弹出时若为灰色就输出，白色说明的新的，重复前面步骤
        white, gray = 0, 1
        res = []
        stack = [(white, root)]
        while stack:
            color, node = stack.pop()
            if node is None: continue
            if color == white:
                stack.append((white, node.right))
                stack.append((gray, node))
                stack.append((white, node.left))
            else:
                res.append(node.val)
        return res
```

```
# 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]:
        ans=[]
        def inorder(root):
            if root:
                inorder(root.left)
                ans.append(root.val)
                inorder(root.right)
        inorder(root)
        return ans
```

代码运行截图 <mark>（至少包含有"Accepted"）</mark>



LC102.二叉树的层序遍历

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

思路：

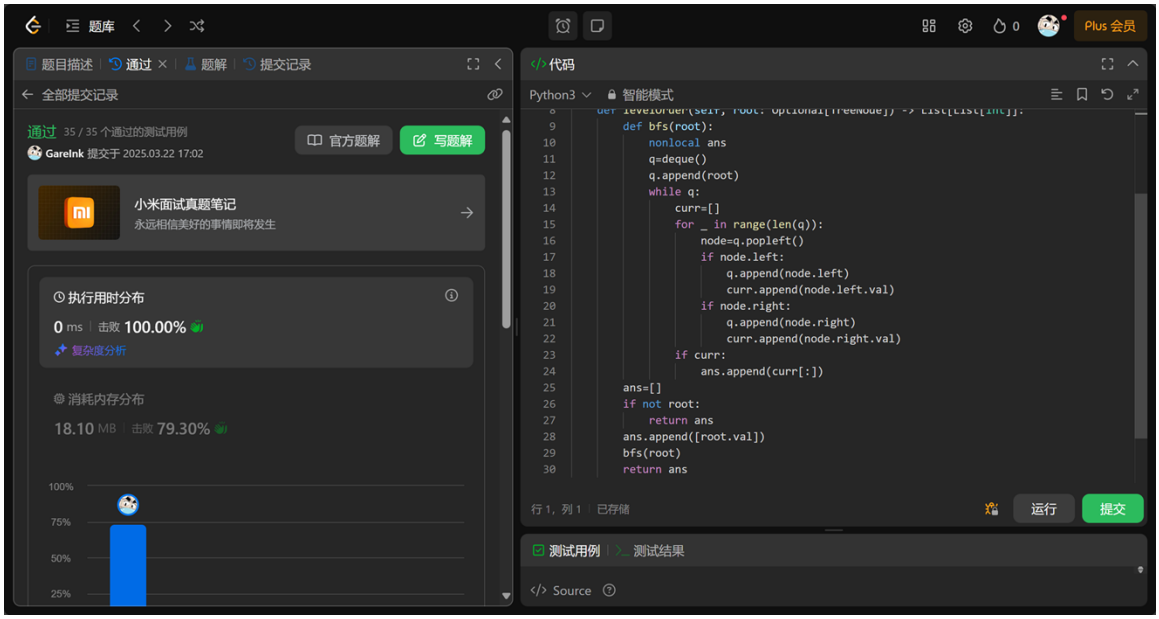
纯正的 bfs，典中典中典的模板了（和第一题差不多的感觉）。为了保证层次遍历，每次都 `if root.left` 和 `if root.right`，保证不漏掉某一侧。

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]]:
        def bfs(root):
            nonlocal ans
            q=deque()
            q.append(root)
            while q:
                curr=[]
                for _ in range(len(q)):
                    node=q.popleft()
                    if node.left:
                        q.append(node.left)
                        curr.append(node.left.val)
                    if node.right:
                        q.append(node.right)
                        curr.append(node.right.val)
                if curr:
                    ans.append(curr[:])
```

```
        ans=[]
        if not root:
            return ans
        ans.append([root.val])
        bfs(root)
        return ans
```

代码运行截图 <mark>（至少包含有"Accepted"）</mark>



LC131.分割回文串

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

思路：

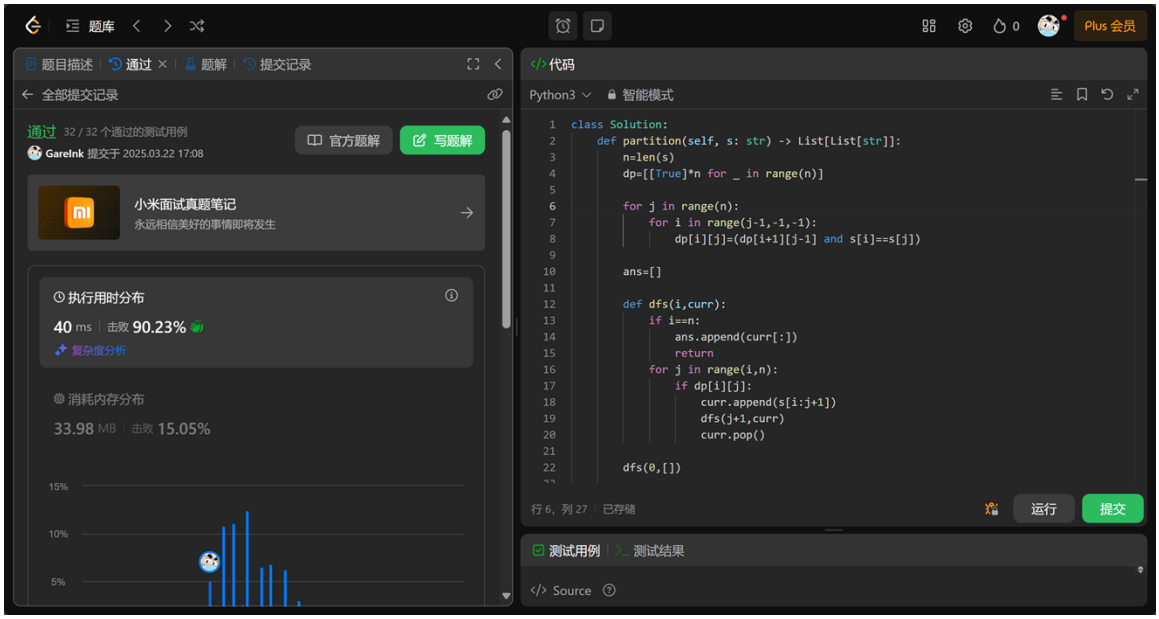
个人感觉这道题挺难的，第一部分的判断某一段子串是不是回文串的 dp 写法；第二部分是 dfs 找切片。其中第一部分的 dp 的值都是布尔值，这样方便后续判断某一个子串是不是回文串；第二部分应该是双指针的思路，用 i 来遍历所有起点，用 j 来从每一个起点开始遍历第一处断点，这种写法也值得积累。

代码：

```
class Solution:
    def partition(self, s: str) -> List[List[str]]:
        n=len(s)
        dp=[[True]*n for _ in range(n)]

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

        ans=[]

        def dfs(i,curr):
            if i==n:
```

```
                ans.append(curr[:])
                return
            for j in range(i,n):
                if dp[i][j]:
                    curr.append(s[i:j+1])
                    dfs(j+1,curr)
                    curr.pop()

        dfs(0,[])

        return ans
```

代码运行截图 <mark>（至少包含有"Accepted"）</mark>



LC146.LRU 缓存

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

思路：

先参考了官方题解的思路然后自己完成的代码。。。首先设置 dummy 头节点和尾节点。当 get 元素时，如果存在就返回值，并将该节点移到头节点；如果不存在就返回-1。当 put 元素时，如果存在就值原地修改，并将该节点移到头节点；如果不存在就在头节点直接插入这个新节点，并判断是否超过了容量，如果超过了就把尾节点删掉。

把节点移到头部分为两步：第一步，获得这个节点的 val 值，把它当作一个新节点插入头部；第二步，在链表中删除原来节点。因此只需要两个辅助函数：placeToHead 和 deleteNode。同时还要注意在超出容量删除尾端元素时要同时把字典里面的这个键值对删去。

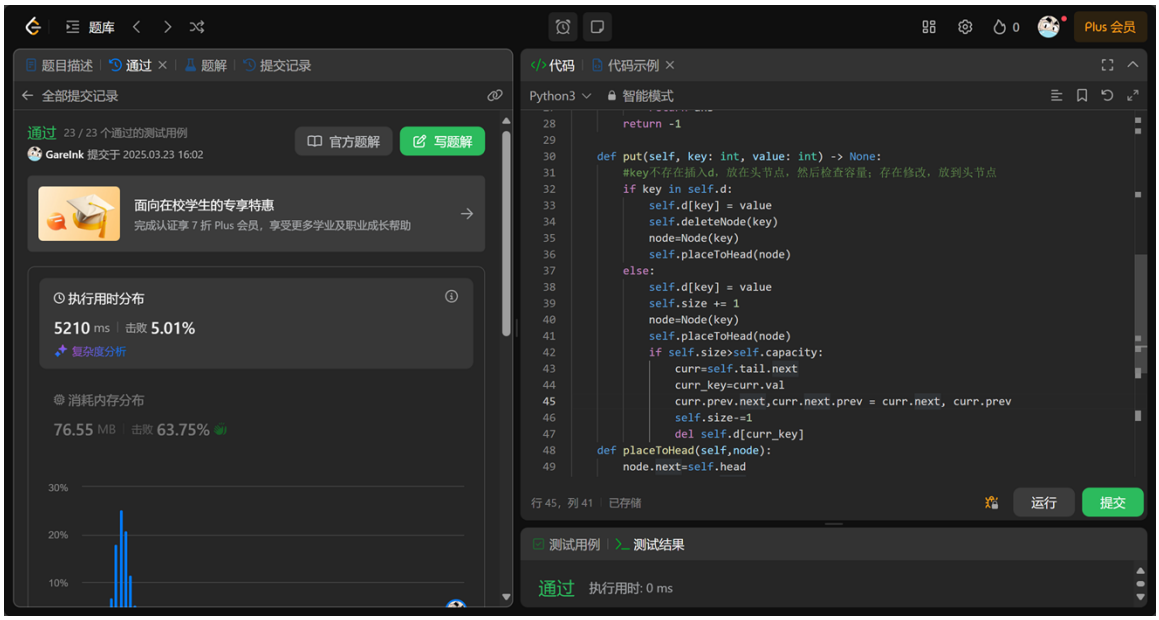
代码：

```
class Node:
    def __init__(self,val,next=None,prev=None):
```

```
        self.val=val
        self.next=next
        self.prev=prev

class LRUCache:

    def __init__(self, capacity: int):
        self.capacity = capacity
        self.size = 0
        self.d = dict()
        self.tail = Node(-1)
        self.head = Node(-1)
        self.head.prev=self.tail
        self.tail.next=self.head
        #tail <-> head

    def get(self, key: int) -> int: #key 不存在-1，存在返回 d[key]并移动到头节点
        if key in self.d:
            ans=self.d[key]
            #删除这个节点
            self.deleteNode(key)
            #头节点设置为它
            node=Node(key)
            self.placeToHead(node)
            return ans
        return -1

    def put(self, key: int, value: int) -> None:
        #key 不存在插入 d，放在头节点，然后检查容量；存在修改，放到头节点
        if key in self.d:
            self.d[key] = value
            self.deleteNode(key)
            node=Node(key)
            self.placeToHead(node)
        else:
            self.d[key] = value
            self.size += 1
            node=Node(key)
            self.placeToHead(node)
            if self.size>self.capacity:
                curr=self.tail.next
                curr_key=curr.val
                curr.prev.next,curr.next.prev = curr.next, curr.prev
                self.size-=1
                del self.d[curr_key]
    def placeToHead(self,node):
        node.next=self.head
        self.head.prev.next=node
        node.prev=self.head.prev
        self.head.prev=node

    def deleteNode(self,key):
        curr=self.tail
        while curr and curr.val!=key:
```

```
            curr=curr.next
        curr.prev.next, curr.next.prev=curr.next, curr.prev


# 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 <mark>（至少包含有"Accepted"）</mark>



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>

第六题卡了我 3 个小时，先是自己写没写出来，参考题解思路写又 debug 半天，难受。

除了第二题和第六题都是寒假做过的，每日选做的解数独是个很硬核的回溯题，最近看到了一个类似的 五子棋 的回溯问题也是相当硬核的，和数独有相似之处：https://www.lanqiao.cn/problems/19694/learning/。课下总结了一下回文串、回文序列问题的模板，在这次作业的分割回文串中很好用。每日选做还在补。。