Assignment #6: 回溯、树、双向链表和哈希表

Updated 1526 GMT+8 Mar 22, 2025

2025 spring, Complied by 任宇桐 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：

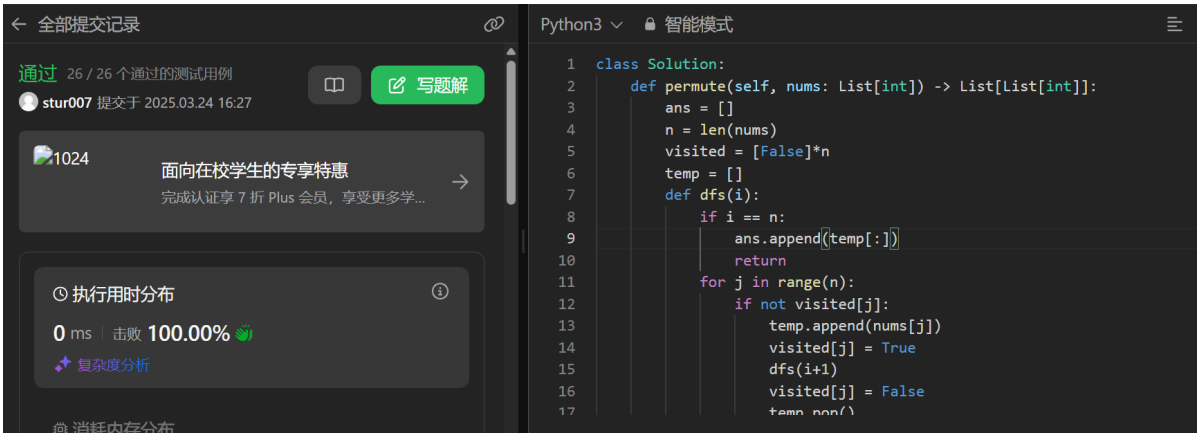
dfs实现

代码：

```
class Solution:
    def permute(self, nums: List[int]) -> List[List[int]]:
        ans = []
        n = len(nums)
        visited = [False]*n
        temp = []
        def dfs(i):
            if i == n:
                ans.append(temp[:])
                return
            for j in range(n):
                if not visited[j]:
                    temp.append(nums[j])
                    visited[j] = True
                    dfs(i+1)
```

```
                    visited[j] = False
                    temp.pop()
        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：

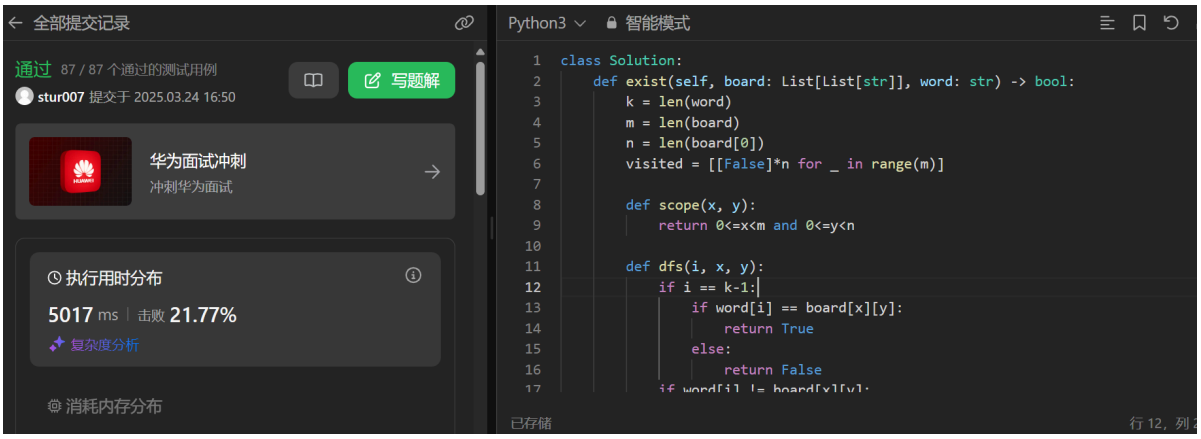
使用dfs实现即可，注意找到一组解以后就可以直接退出递归了。

代码：

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

        def scope(x, y):
            return 0<=x<m and 0<=y<n

        def dfs(i, x, y):
            if i == k-1:
                if word[i] == board[x][y]:
                    return True
                else:
                    return False
            if word[i] != board[x][y]:
                return False

            visited[x][y] = True
            for dx, dy in [(0, 1), (0, -1), (1, 0), (-1, 0)]:
                nx = x+dx
                ny = y+dy
                if scope(nx, ny) and not visited[nx][ny]:
```

```
                    if dfs(i+1, nx, ny):
                        return True
            visited[x][y] = False
            return False

        for i in range(m):
            for j in range(n):
                if dfs(0, i, j):
                    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：

直接递归实现即可。

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]:
        def inorder(node):
            if node is None:
                return []
            return inorder(node.left)+[node.val]+inorder(node.right)
        return inorder(root)
```

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/

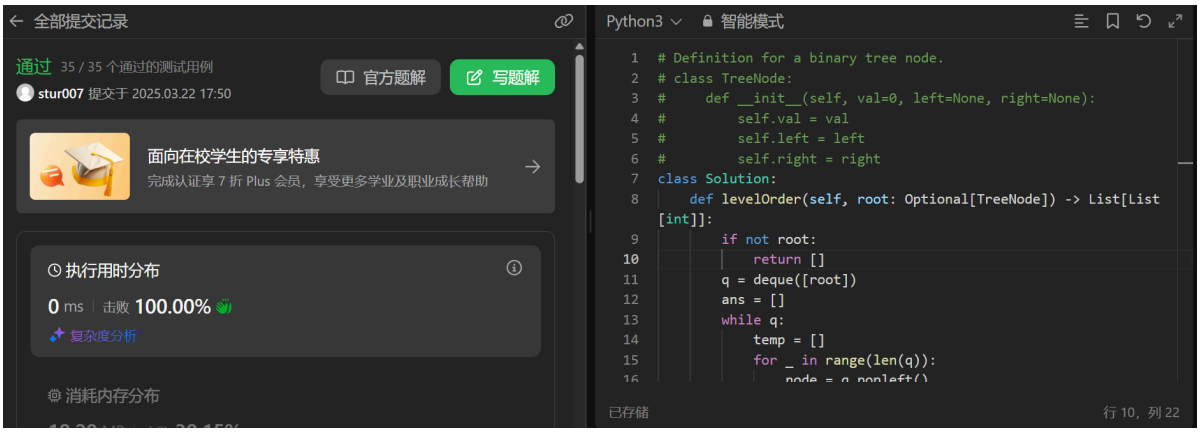
思路:

直接按照bfs的方式遍历即可。

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]]:
        if not root:
            return []
        q = deque([root])
        ans = []
        while q:
            temp = []
            for _ in range(len(q)):
                node = q.popleft()
                if node.left:
                    q.append(node.left)
                if node.right:
                    q.append(node.right)
                temp.append(node.val)
            ans.append(temp[:])
        return ans
```

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：

字符串较短，似乎不用在回文上下太多功夫，直接dfs实现即可。

代码：

```
class Solution:
    def partition(self, s: str) -> List[List[str]]:
        n = len(s)

        def check_palindrome(temp):
            return temp == temp[::-1]

        ans = []
        palindromes = []

        def dfs(i):
            if i == n:
                ans.append(palindromes[:])
                return

            temp = ''
            for j in range(i, n):
                temp += s[j]
                if check_palindrome(temp):
                    palindromes.append(temp)
                    dfs(j + 1)
                    palindromes.pop()
        dfs(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:

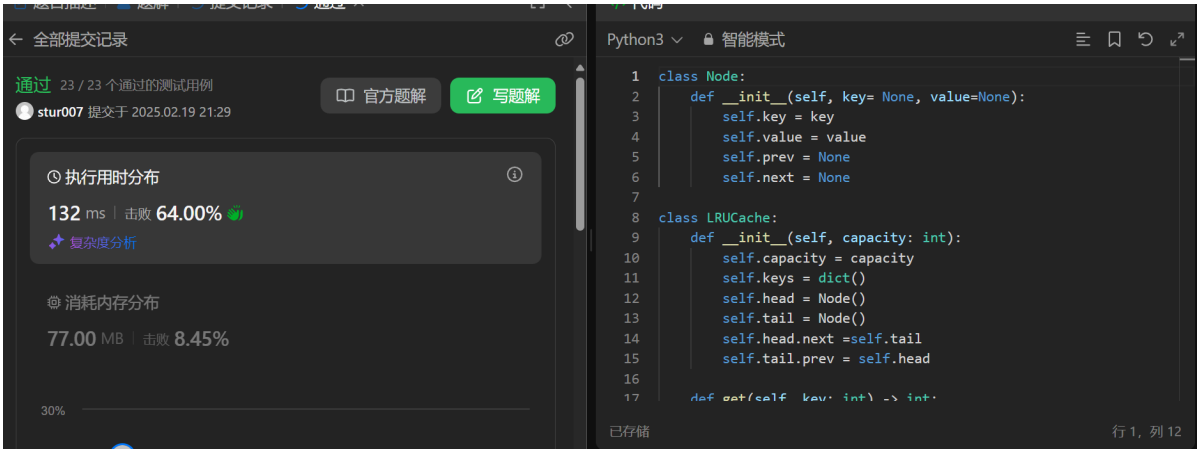
使用双链表快速添加与删除元素。

代码:

```
class Node:
    def __init__(self, key= None, value=None):
        self.key = key
        self.value = value
        self.prev = None
        self.next = None

class LRUCache:
    def __init__(self, capacity: int):
        self.capacity = capacity
        self.keys = dict()
        self.head = Node()
        self.tail = Node()
        self.head.next =self.tail
        self.tail.prev = self.head

    def get(self, key: int) -> int:
        if key not in self.keys:
            return -1
        current = self.keys[key]
        current.prev.next =current.next
        current.next.prev = current.prev
        current.next = self.head.next
        self.head.next.prev = current
        self.head.next = current
        current.prev = self.head
        return current.value

    def put(self, key: int, value: int) -> None:
        if key in self.keys:
            current = self.keys[key]
            current.prev.next =current.next
            current.next.prev = current.prev
```

```
            current.next = self.head.next
            self.head.next.prev = current
            self.head.next = current
            current.prev = self.head
            current.value = value
            return
        if len(self.keys) == self.capacity:
            deletekey = self.tail.prev.key
            self.keys.pop(deletekey)
            self.tail.prev.prev.next = self.tail
            self.tail.prev = self.tail.prev.prev
        self.keys[key] = Node(key, value)
        current =self.keys[key]
        self.head.next.prev = current
        current.next = self.head.next
        self.head.next = current
        current.prev = self.head


# 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。

感觉回溯部分就是dfs复习课?