Assignment #6: 回溯、树、双向链表和哈希表

Updated 1526 GMT+8 Mar 22, 2025

2025 spring, Complied by **袁奕 2400010766 数院**

> **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

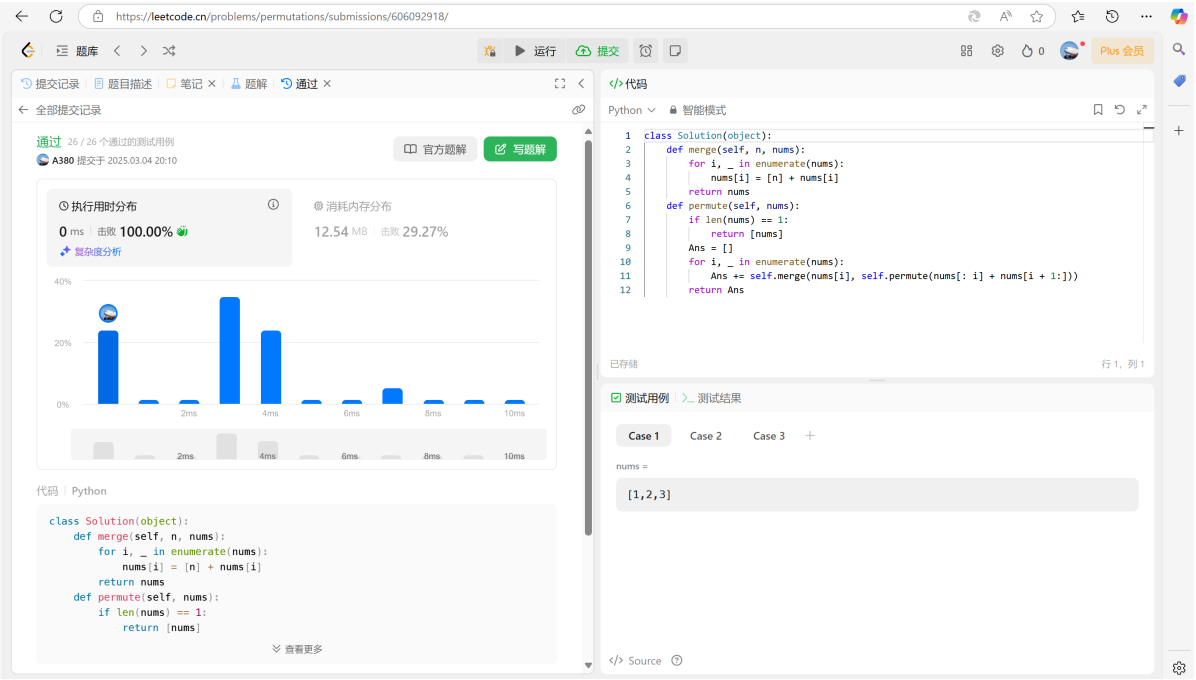
LC46.全排列

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

思路：

代码：

```
class Solution(object):
    def merge(self, n, nums):
        for i, _ in enumerate(nums):
            nums[i] = [n] + nums[i]
        return nums
    def permute(self, nums):
        if len(nums) == 1:
            return [nums]
        Ans = []
        for i, _ in enumerate(nums):
            Ans += self.merge(nums[i], self.permute(nums[: i] + nums[i + 1:]))
        return Ans
```



LC79: 单词搜索

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

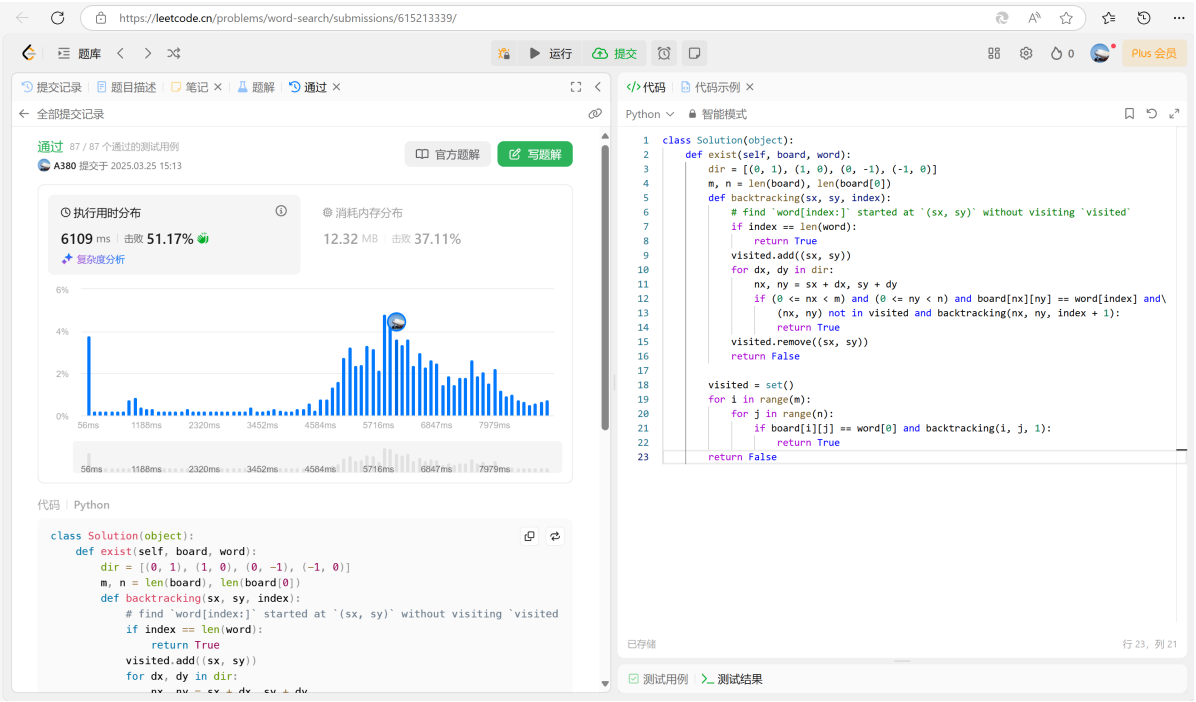
惨痛教训：

1. 老老实实用 `0 <= x < n`，不要偷懒耍滑用 `x in range(n)`，后者是在 `list` 中遍历，会极大的提高复杂度
2. 为了节省时间，将 `visited` 设为全局变量，但是回溯后记得还原（`remove` 过程）

代码：

```
class Solution(object):
    def exist(self, board, word):
        dir = [(0, 1), (1, 0), (0, -1), (-1, 0)]
        m, n = len(board), len(board[0])
        def backtracking(sx, sy, index):
            # find `word[index:]` started at `(sx, sy)` without visiting `visited`
            if index == len(word):
                return True
            visited.add((sx, sy))
            for dx, dy in dir:
                nx, ny = sx + dx, sy + dy
                if (0 <= nx < m) and (0 <= ny < n) and board[nx][ny] == word[index] and\
                        (nx, ny) not in visited and backtracking(nx, ny, index + 1):
                    return True
            visited.remove((sx, sy))
            return False

```

```
        visited = set()
        for i in range(m):
            for j in range(n):
                if board[i][j] == word[0] and backtracking(i, j, 1):
                    return True
        return False
```



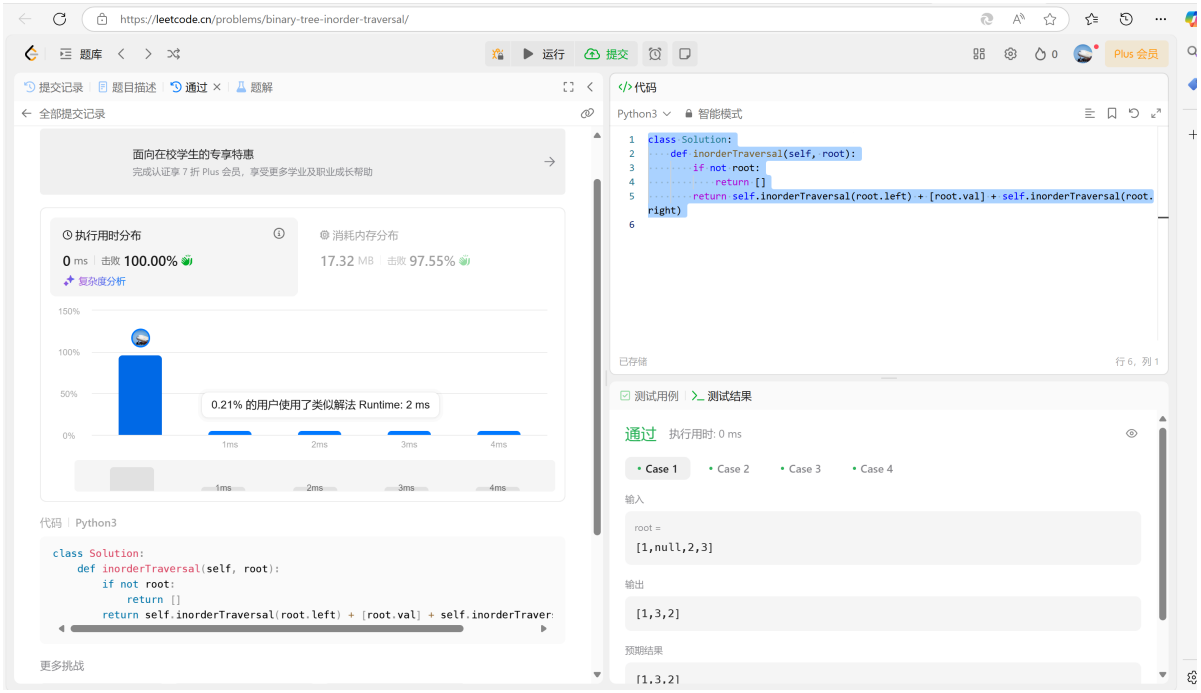
LC94.二叉树的中序遍历

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

思路：

代码：

```
class Solution:
    def inorderTraversal(self, root):
        if not root:
            return []
        return self.inorderTraversal(root.left) + [root.val] +
self.inorderTraversal(root.right)
```



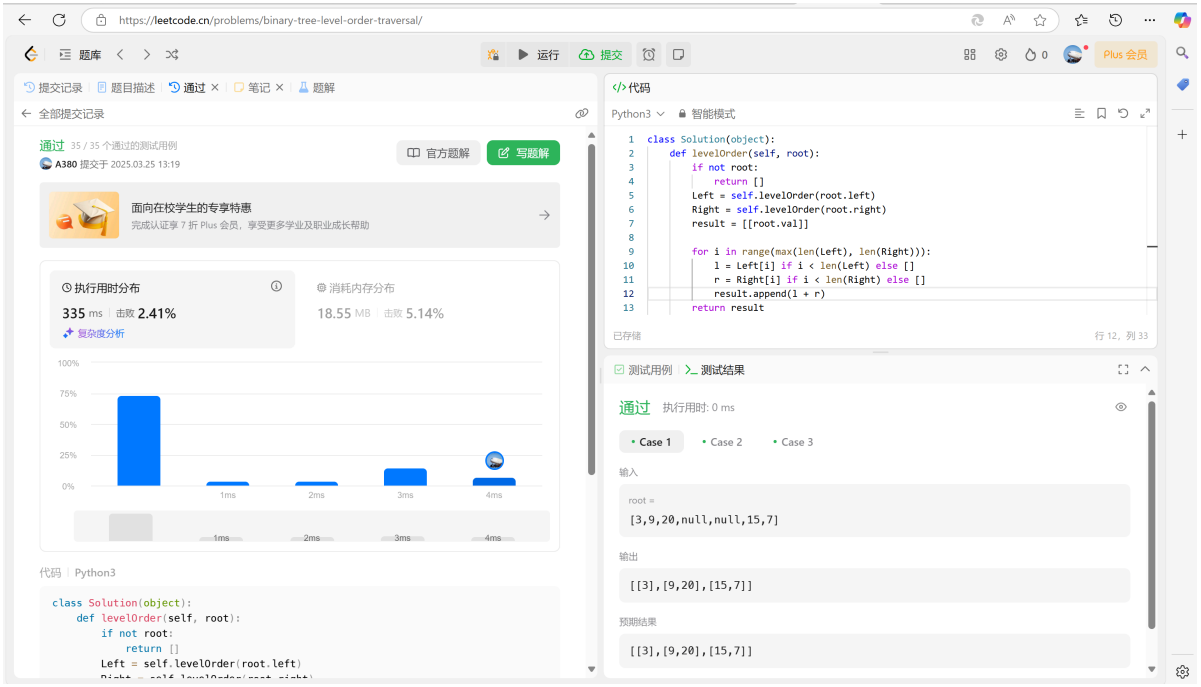
LC102.二叉树的层序遍历

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

思路：

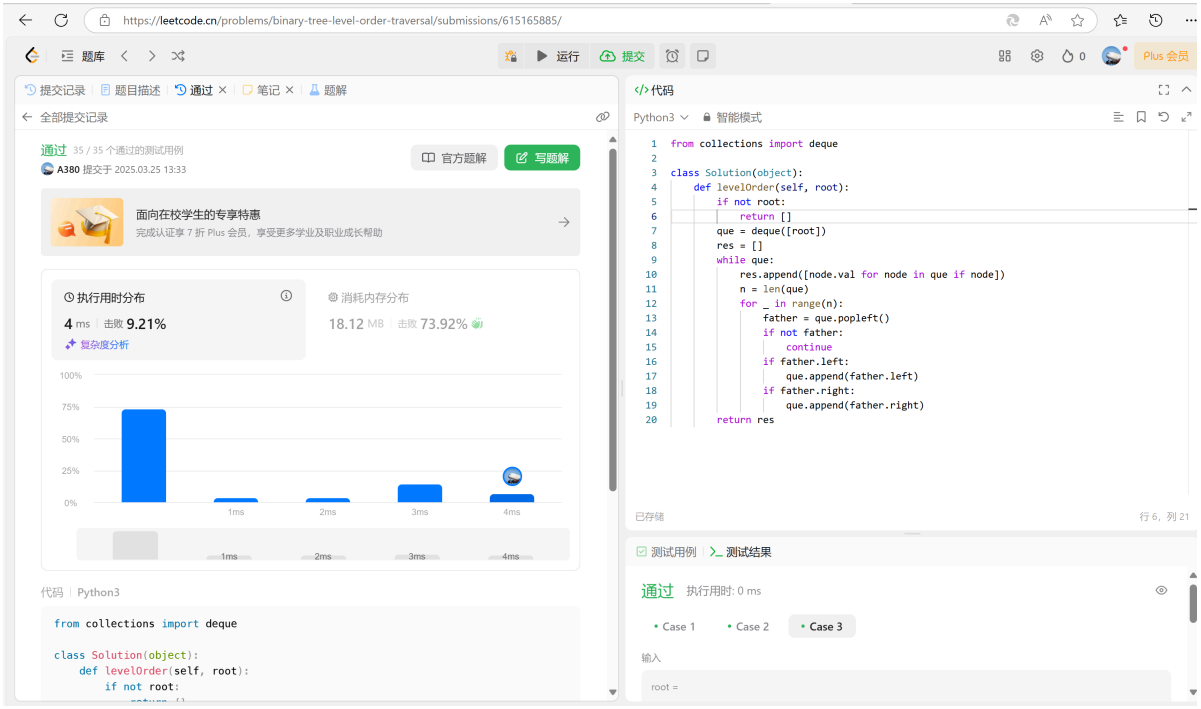
方法1：

```python
class Solution(object):
    def levelOrder(self, root):
        if not root:
            return []
        Left = self.levelOrder(root.left)
        Right = self.levelOrder(root.right)
        result = [root.val]

        for i in range(max(len(Left), len(Right))):
            l = Left[i] if i < len(Left) else []
            r = Right[i] if i < len(Right) else []
            result.append(l + r)
        return result
```



方法2:

```python
from collections import deque

class Solution(object):
    def levelOrder(self, root):
        que = deque([root])
        res = []
        while que:
            res.append([node.val for node in que if node])
            n = len(que)
            for _ in range(n):
                father = que.popleft()
                if father.left:
                    que.append(father.left)
                if father.right:
                    que.append(father.right)
        return res
```

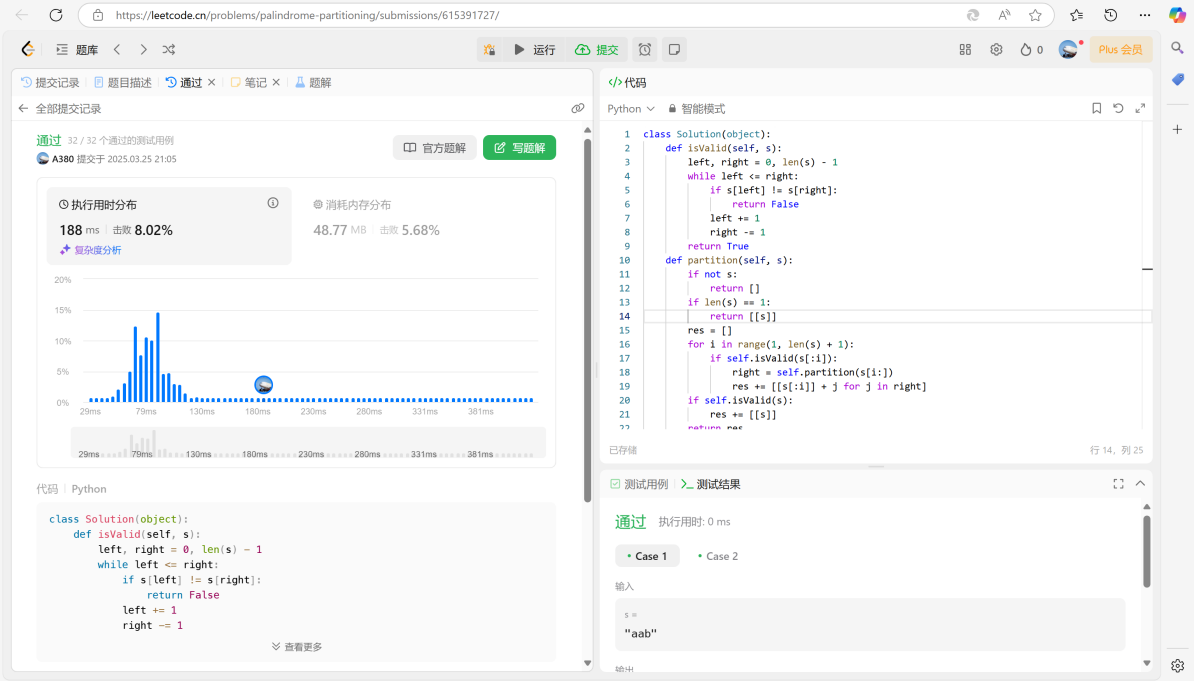


LC131.分割回文串

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

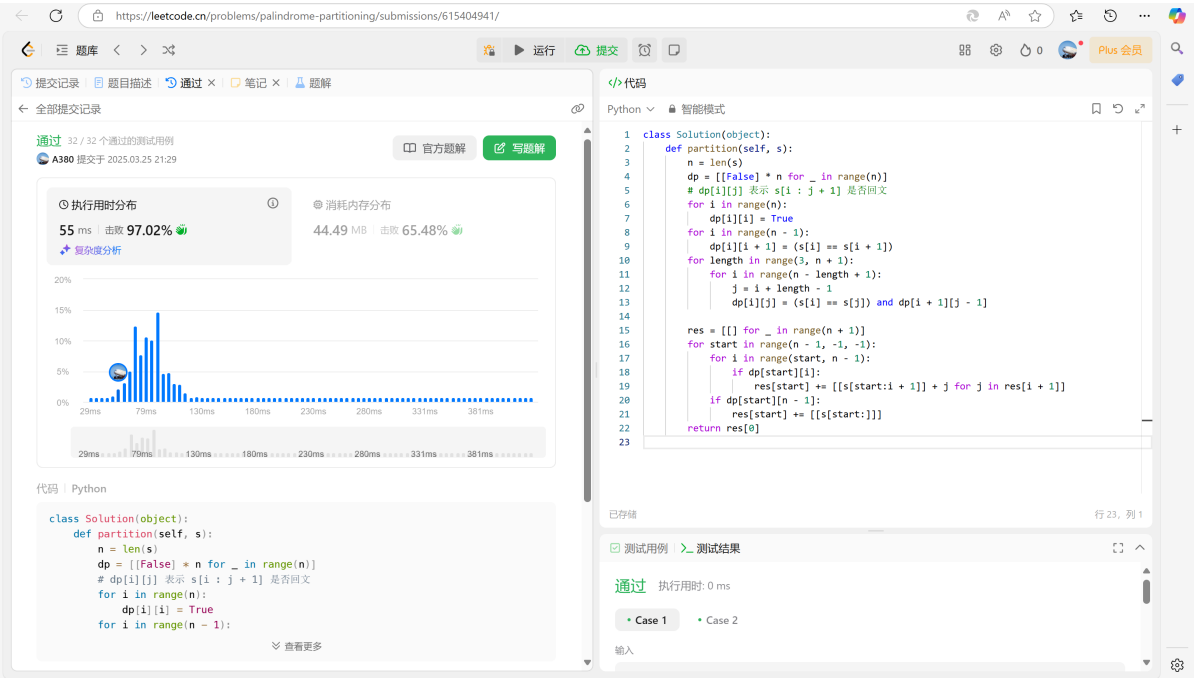
思路：长度小于 16, 性能问题不太需要关注 (哪怕 $O(n^3)$ 也能过) 考试肯定会这样写节省时间

```python
class Solution(object):
    def isValid(self, s):
        left, right = 0, len(s) - 1
        while left <= right:
            if s[left] != s[right]:
                return False
            left += 1
            right -= 1
        return True
    def partition(self, s):
        if not s:
            return []
        if len(s) == 1:
            return [[s]]
        res = []
        for i in range(1, len(s) + 1):
            if self.isValid(s[:i]):
                right = self.partition(s[i:])
                res += [[s[:i]] + j for j in right]
        if self.isValid(s):
            res += [[s]]
        return res
```



为了性能更优化, 可以采取类似[5. 最长回文子串 - 力扣（LeetCode）](#) 的办法：

```python
class Solution(object):
    def partition(self, s):
        n = len(s)
        dp = [[False] * n for _ in range(n)]
        # dp[i][j] 表示 s[i : j + 1] 是否回文
        for i in range(n):
            dp[i][i] = True
        for i in range(n - 1):
            dp[i][i + 1] = (s[i] == s[i + 1])
        for length in range(3, n + 1):
            for i in range(n - length + 1):
                j = i + length - 1
                dp[i][j] = (s[i] == s[j]) and dp[i + 1][j - 1]

        res = [[] for _ in range(n + 1)]
        for start in range(n - 1, -1, -1):
            for i in range(start, n - 1):
                if dp[start][i]:
                    res[start] += [[s[start:i + 1]] + j for j in res[i + 1]]
            if dp[start][n - 1]:
                res[start] += [[s[start:]]]
        return res[0]
```



LC146.LRU缓存

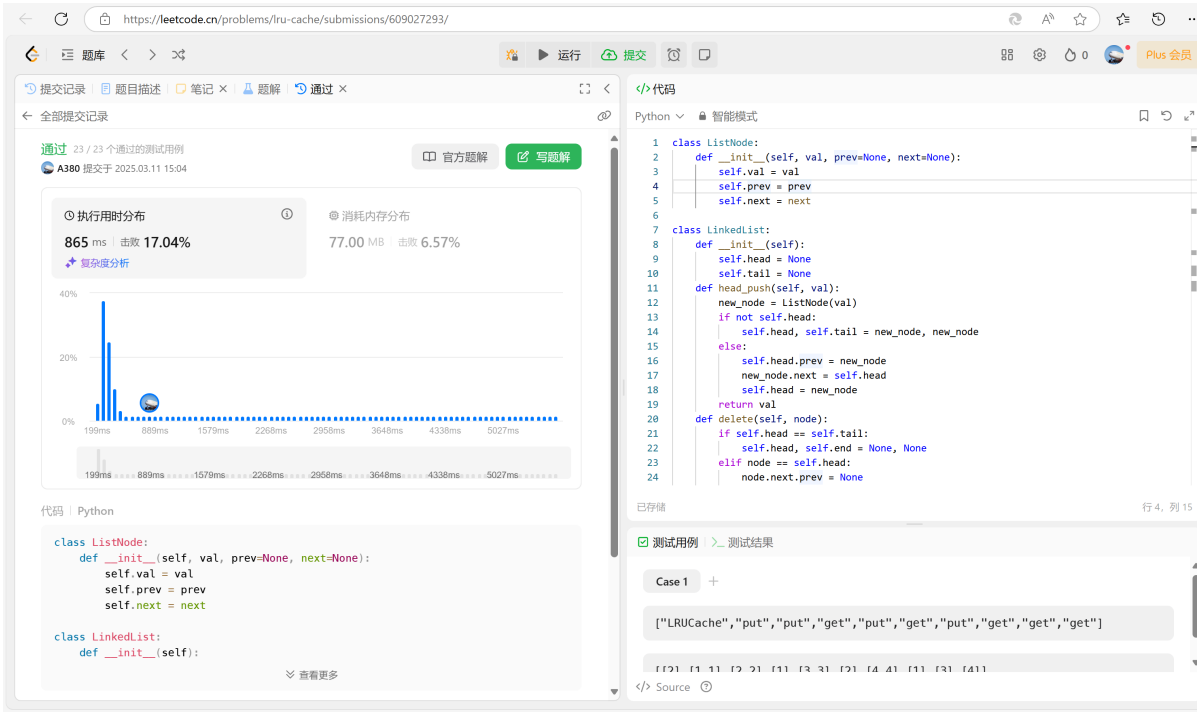
hash table, doubly-linked list, https://leetcode.cn/problems/lru-cache/

思路:

代码:

```python
class ListNode:
    def __init__(self, val, prev=None, next=None):
        self.val = val
        self.prev = prev
        self.next = next

class LinkedList:
    def __init__(self):
        self.head = None
        self.tail = None
    def head_push(self, val):
        new_node = ListNode(val)
        if not self.head:
            self.head, self.tail = new_node, new_node
        else:
            self.head.prev = new_node
            new_node.next = self.head
            self.head = new_node
        return val
    def delete(self, node):
        if self.head == self.tail:
            self.head, self.end = None, None
        elif node == self.head:
            node.next.prev = None
            self.head = node.next
        elif node == self.tail:
```

```
            node.prev.next = None
            self.tail = node.prev
        else:
            node.prev.next = node.next
            node.next.prev = node.prev
    def put_head(self, node):
        self.delete(node)
        self.head_push(node.val)


class LRUCache(object):
    def __init__(self, capacity):
        self.cache = dict() # key : node (type node : ListNode)
        self.pair = dict() # key : value
        self.capacity = capacity
        self.linked_list = LinkedList() # (val : key)
    def get(self, key):
        if key not in self.cache:
            return -1
        node = self.cache[key]
        self.linked_list.put_head(node)
        self.cache[key] = self.linked_list.head
        return self.pair[key]
    def put(self, key, value):
        if key in self.cache:
            node = self.cache[key]
            self.linked_list.put_head(node)
            self.cache[key] = self.linked_list.head
            self.pair[key] = value
        else:
            self.linked_list.head_push(key)
            self.cache[key] = self.linked_list.head
            self.pair[key] = value
            if len(self.cache) > self.capacity:
                tail_key = self.linked_list.tail.val
                self.cache.pop(tail_key)
                self.pair.pop(tail_key)
                self.linked_list.delete(self.linked_list.tail)
```


https://leetcode.cn/problems/lru-cache/submissions/609027293/
题库
运行
提交
Plus 会员
提交记录
题目描述
笔记
题解
通过
全部提交记录
通过 23 / 23 个通过的测试用例
A380 提交于 2025.03.11 15:04
官方题解
写题解
执行用时分布
865 ms | 击败 17.04%
复杂度分析
消耗内存分布
77.00 MB | 击败 6.57%
40%
20%
0%
199ms
889ms
1579ms
2268ms
2958ms
3648ms
4338ms
5027ms
代码 | Python
class ListNode:
def __init__(self, val, prev=None, next=None):
self.val = val
self.prev = prev
self.next = next
class LinkedList:
def __init__(self):
查看更多
代码
Python
智能模式
class ListNode:
def __init__(self, val, prev=None, next=None):
self.val = val
self.prev = prev
self.next = next
class LinkedList:
def __init__(self):
self.head = None
self.tail = None
def head_push(self, val):
new_node = ListNode(val)
if not self.head:
self.head, self.tail = new_node, new_node
else:
self.head.prev = new_node
new_node.next = self.head
self.head = new_node
return val
def delete(self, node):
if self.head == self.tail:
self.head, self.end = None, None
elif node == self.head:
node.next.prev = None
已存储
行 4，列 15
测试用例
测试结果
Case 1
["LRUCache","put","put","get","put","get","put","get","get","get"]
Source

2. 学习总结和收获