Assignment #6: 回溯、树、双向链表和哈希表

Updated 1526 GMT+8 Mar 22, 2025

2025 spring, Complied by 金俊毅，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

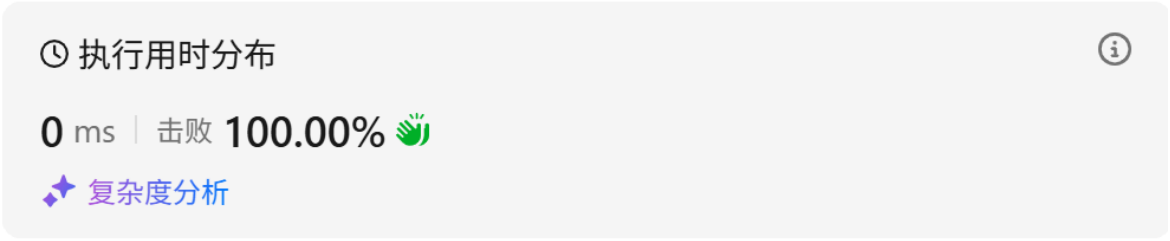
LC46.全排列

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

代码：

```
class Solution(object):
    def permute(self, nums):
        """
        :type nums: List[int]
        :rtype: List[List[int]]
        """
        def insert(k):
            if k == 1:
                return [[nums[0]]]
            mid = []
            last = insert(k-1)
            for a in last:
                for i in range(k):
                    mid.append(a[:i]+[nums[k-1]]+a[i:])
            return mid

        return insert(len(nums))
```

代码运行截图 （至少包含有"Accepted"）



LC79: 单词搜索

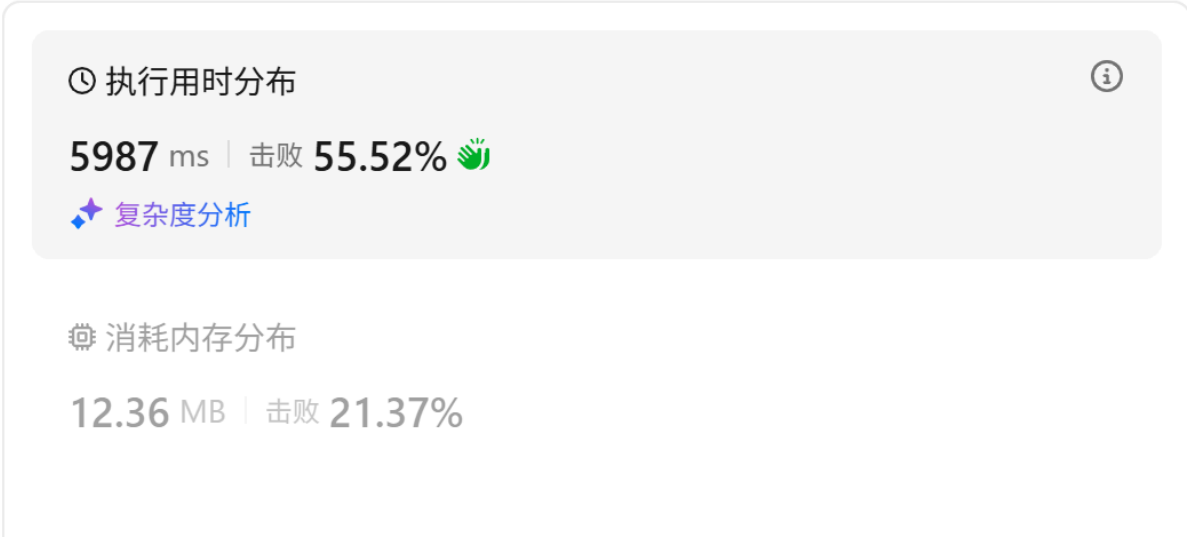
backtracking, https://leetcode.cn/problems/word-search/

代码：

```
class Solution(object):
    def exist(self, board, word):
        """
        :type board: List[List[str]]
        :type word: str
        :rtype: bool
        """
        direction = [(1, 0), (-1, 0), (0, 1), (0, -1)]
        head = word[0]
        stor = []
        m = len(board)
        n = len(board[0])
        for i in range(m):
            for j in range(n):
                if board[i][j] == head:
                    stor.append((i, j))
        arbit = [[False for _ in range(n)] for _ in range(m)]

        def dfs(x, y, step):
            if step == len(word)-1:
                return True

            for p in range(4):
                dx, dy = direction[p]
                if 0 <= x+dx < n and 0 <= y+dy < m:
                    if not arbit[y+dy][x+dx] and board[y+dy][x+dx] ==
word[step+1]:
                        arbit[y+dy][x+dx] = True
                        if dfs(x+dx, y+dy, step+1):
                            return True
                        arbit[y+dy][x+dx] = False

            return False

        for st in stor:
            arbit = [[False for _ in range(n)] for _ in range(m)]
            i1, i2 = st
            arbit[i1][i2] = True
```

```
            if dfs(i2, i1, 0):
                return True

        return False
```

代码运行截图（至少包含有"Accepted"）



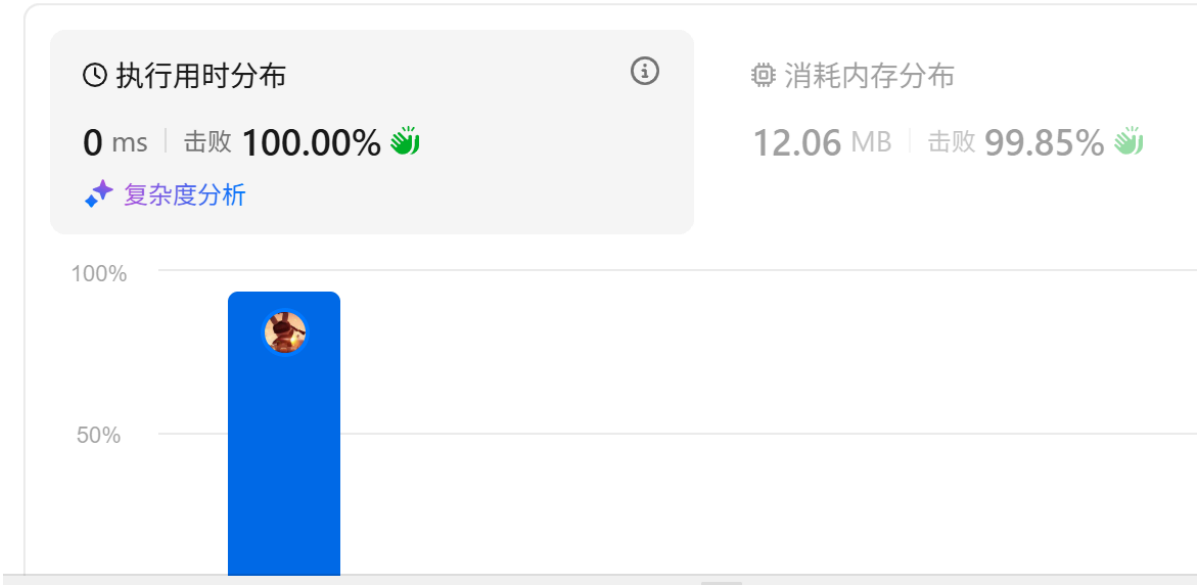
LC94.二叉树的中序遍历

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

代码：

```
# Definition for a binary tree node.
# class TreeNode(object):
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution(object):
    def inorderTraversal(self, root):
        """
        :type root: Optional[TreeNode]
        :rtype: List[int]
        """
        result = []

        def dfs(x):
            if not x:
                return
            dfs(x.left)
            result.append(x.val)
            dfs(x.right)

        dfs(root)
        return result
```

代码运行截图（至少包含有"Accepted"）



LC102.二叉树的层序遍历

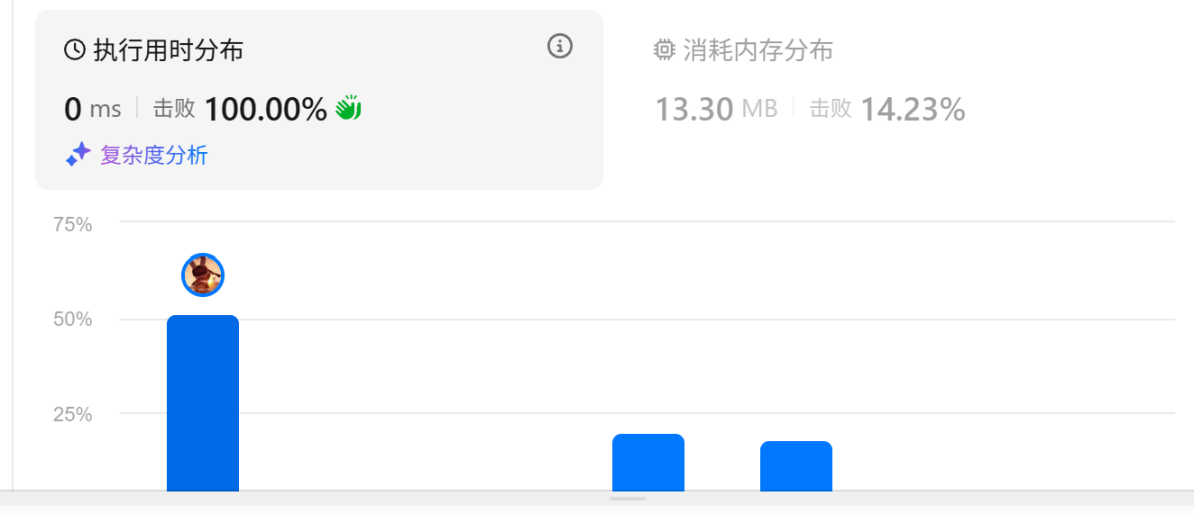
bfs, https://leetcode.cn/problems/binary-tree-level-order-traversal/

代码：

```
# Definition for a binary tree node.
# class TreeNode(object):
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
from collections import deque


class Solution(object):
    def levelOrder(self, root):
        """
        :type root: Optional[TreeNode]
        :rtype: List[List[int]]
        """
        q = deque()
        q.append((root, 1))
        ans = []
        visited = set()
        if root == None:
            return []
        while q:
            node, step = q.popleft()
            if node.left:
                q.append((node.left, step+1))
            if node.right:
                q.append((node.right, step+1))
            if step not in visited:
```

```
                ans.append([])
                visited.add(step)
            ans[-1].append(node.val)

        return ans
```

代码运行截图 （至少包含有"Accepted"）



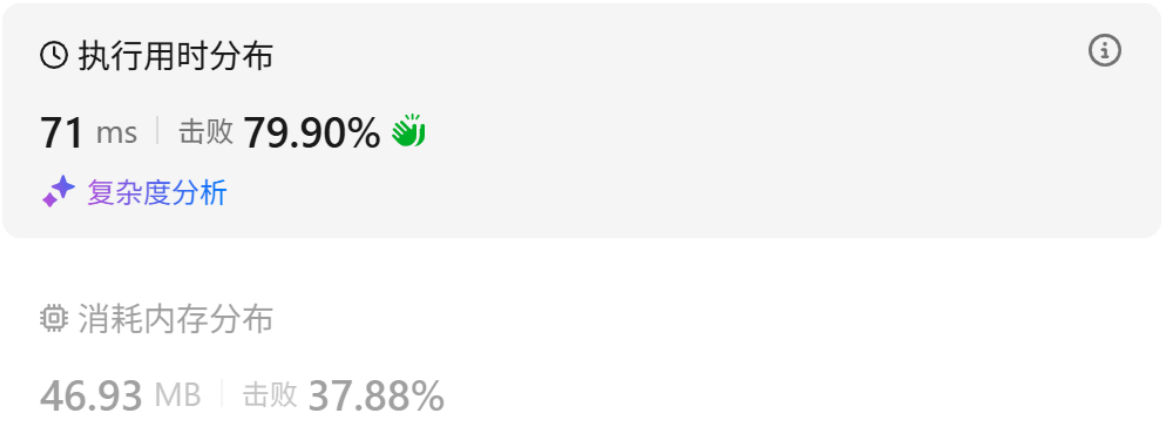
LC131.分割回文串

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

代码：

```
class Solution(object):
    def partition(self, s):
        """
        :type s: str
        :rtype: List[List[str]]
        """
        ans = []
        def part(fore, latt):
            if not latt:
                ans.append(fore)
            else:
                for i in range(1, len(latt)+1):
                    if latt[:i] == latt[:i][::-1]:
                        part(fore+[latt[:i]], latt[i:])
        part([], s)
        return ans
```

代码运行截图（至少包含有"Accepted"）



LC146.LRU缓存

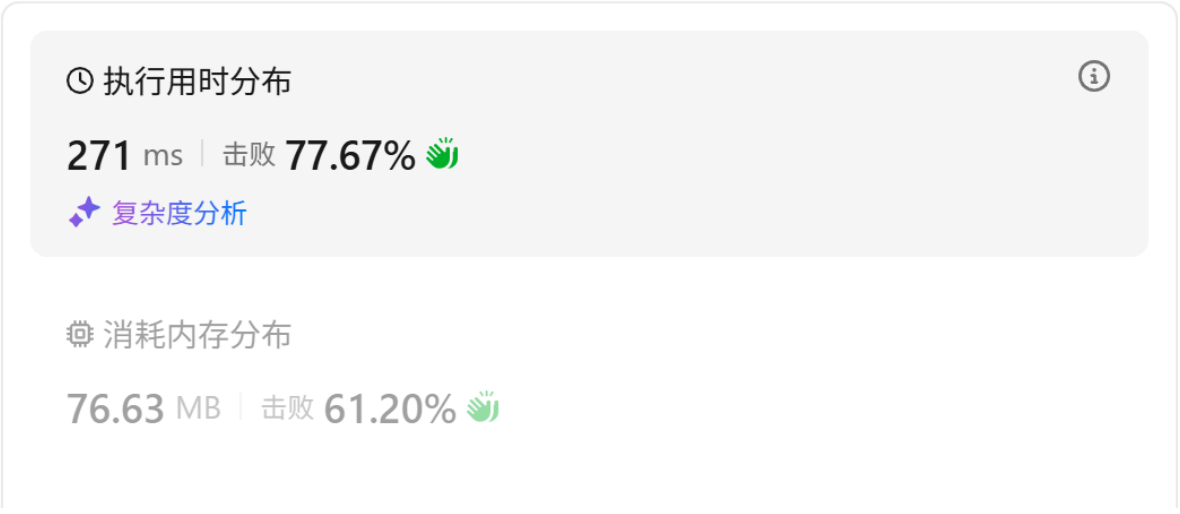
hash table, doubly-linked list, https://leetcode.cn/problems/lru-cache/

代码：

```
class ListNode:
    def __init__(self, key=0, value=0):
        self.next = None
        self.prev = None
        self.key = key
        self.value = value


class LRUCache(object):

    def __init__(self, capacity):
        """
        :type capacity: int
        """
        self.capacity = capacity
        self.dic = {}
        self.head = ListNode()
        self.tail = ListNode()
        self.head.next = self.tail
        self.tail.prev = self.head

    def rem(self, node):
        node.prev.next = node.next
        node.next.prev = node.prev

    def ins(self, node):
        node.prev = self.head
        node.next = self.head.next
        self.head.next = node
        node.next.prev = node

    def get(self, key):
```

```
        """
        :type key: int
        :rtype: int
        """
        if key in self.dic:
            node = self.dic[key]
            self.rem(node)
            self.ins(node)
            return node.value
        return -1

    def put(self, key, value):
        """
        :type key: int
        :type value: int
        :rtype: None
        """
        node = ListNode(key, value)
        if key in self.dic:
            self.rem(self.dic[key])
        self.dic[key] = node
        self.ins(node)
        if len(self.dic) > self.capacity:
            del self.dic[self.tail.prev.key]
            self.rem(self.tail.prev)
```

代码运行截图（至少包含有"Accepted"）



2. 学习总结和收获

从第六题的数据结构实现学习到了创造虚拟头结点和虚拟尾节点，从而方便更新数据使用时间，head和tail固定，方便直接插入，不用绕来绕去。