Assignment #6: 回溯、树、双向链表和哈希表

Updated 1526 GMT+8 Mar 22, 2025

2025 spring, Complied by 叶靖、信管

> **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：

回溯

代码：

class Solution: def permute(self, nums: List[int]) → List[List[int]]: def backtrack(path, used): if len(path) == len(nums): result.append(path[:]) return for i in range(len(nums)): if not used[i]: used[i] = True path.append(nums[i]) backtrack(path, used) path.pop() used[i] = False

```
result = []
backtrack([], [False] * len(nums))
return result
```

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：

回溯 遍历所有网格，找word的起始字符，从上下左右尝试匹配下个字符，返回结果

代码：

class Solution: def exist(self, board: List[List[str]], word: str) → bool: def dfs(i, j, k): if not 0 ⇐ i < len(board) or not 0 ⇐ j < len(board[0]) or board[i][j] != word[k]: return False if k == len(word) - 1: return True tmp, board[i][j] = board[i][j], '/' res = dfs(i + 1, j, k + 1) or dfs(i - 1, j, k + 1) or dfs(i, j + 1, k + 1) or dfs(i, j - 1, k + 1) board[i][j] = tmp return res

```
for i in range(len(board)):
    for j in range(len(board[0])):
        if dfs(i, j, 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/

思路：

递归（左子树 → 根节点 → 右子树） 先递归左子树，访问当前节点，后递归右子树

代码：

class Solution: def inorderTraversal(self, root: Optional[TreeNode]) → List[int]: result = [] def traverse(node): if not node: return traverse(node.left) result.append(node.val) traverse(node.right) traverse(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/

思路：

deque：先进先出，确保节点按层次遍历 循环处理节点，然后按层储存结果

代码：

class Solution: def levelOrder(self, root: Optional[TreeNode]) → List[List[int]]: if not root: return [] result = [] queue = deque([root]) while queue: level_size = len(queue) level_values = [] for _ in range(level_size): node = queue.popleft() level_values.append(node.val)
if node.left: queue.append(node.left) if node.right: queue.append(node.right)
result.append(level_values) return result

代码运行截图 （至少包含有"Accepted"）

 06-04

LC131.分割回文串

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

思路：

回溯、递归、预处理回文判断（dp） 从头开始找回文子串，找到即储存，然后继续进行递归处理

代码：

class Solution: def partition(self, s: str) → List[List[str]]: n = len(s) result = [] path = []

```
        dp = [[False] * n for _ in range(n)]
        for right in range(n):
            for left in range(right + 1):
                if s[left] == s[right] and (right - left <= 2 or dp[left + 1][right - 1]):
                    dp[left][right] = True

        def backtrack(start):
            if start == n:
                result.append(path[:])
                return
            for end in range(start, n):
                if dp [start][end]:
                    path.append(s[start:end + 1])
                    backtrack(end + 1)
                    path.pop()
        backtrack(0)
        return result
```

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：

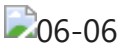
哈希表、双向链表 链头是最近使用的元素，链尾是最久未使用的元素，每访问一次就更新一次，超出capacity就删除链尾元素 哈希表用于储存key-value

代码：

class DListNode: def **init**(self, key=0, value=0): self.key = key self.value = value self.prev = None self.next = None

class LRUCache: def **init**(self, capacity: int): self.capacity = capacity self.cache = self.head = DListNode() self.tail = DListNode() self.head.next = self.tail self.tail.prev = self.head def get(self, key: int) → int: if key in self.cache: node = self.cache[key] node.prev.next = node.next node.next.prev = node.prev node.next = self.head.next node.prev = self.head self.head.next.prev = node self.head.next = node return node.value return -1 def put(self, key: int, value: int) → None: if key in self.cache: node = self.cache[key] node.value = value node.prev.next = node.next node.next.prev = node.prev node.next = self.head.next node.prev = self.head self.head.next.prev = node self.head.next = node else: if len(self.cache) >= self.capacity: tail_node = self.tail.prev tail_node.prev.next = self.tail self.tail.prev = tail_node.prev del self.cache[tail_node.key] new_node = DListNode(key, value) self.cache[key] = new_node new_node.next = self.head.next new_node.prev = self.head self.head.next.prev = new_node self.head.next = new_node

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。

这类型的题目理解并不难，能够很快地理解到操作过程，但是要将思路转化成代码，让其顺利输出有一定难度。 尤其是输出方面，要多层循环，然后将结果储存起来最后输出。但是由于需要不断地进行循环，过程会很绕，所以要怎样制造出口让答案按格式输出是难点。