assignmentA

Assignment #A: 递归回溯、🌲 (3/4)

Updated 2203 GMT+8 Nov 3, 2025

2025 fall, Complied by 杨知进 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本人头像，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

T51.N皇后

backtracking, https://leetcode.cn/problems/n-queens/

思路：

就是八皇后的代码将8改成n。进一步在题解区学习了更规范高效的回溯写法：通过维护更新的used数组的True or False来作为判断对象。

代码：

```
class Solution:
    def solveNQueens(self, n: int) -> List[List[str]]:
        def dfs(nums, row):
            if row == n:
                temp = []
```

```
                for i in board:
                    temp_row = ['.' for _ in range(n)]
                    temp_row[i] = 'Q'
                    temp.append(''.join(temp_row))
                ans.append(temp)
                return

            for col in range(n):
                if usedCol[col] or usedDiag45[row - col + n - 1] or
usedDiag135[row + col]:
                    continue
                nums[row] = col
                usedCol[col] = True
                usedDiag45[row - col + n - 1] = True
                usedDiag135[row + col] = True
                dfs(nums, row + 1)
                usedCol[col] = False
                usedDiag45[row - col + n - 1] = False
                usedDiag135[row + col] = False

        ans = []
        usedCol = [False] * n
        usedDiag45 = [False] * (2 * n - 1)
        usedDiag135 = [False] * (2 * n - 1)
        board = [-1] * n

        dfs(board, 0)
        return ans
```

代码运行截图 （至少包含有"Accepted"）

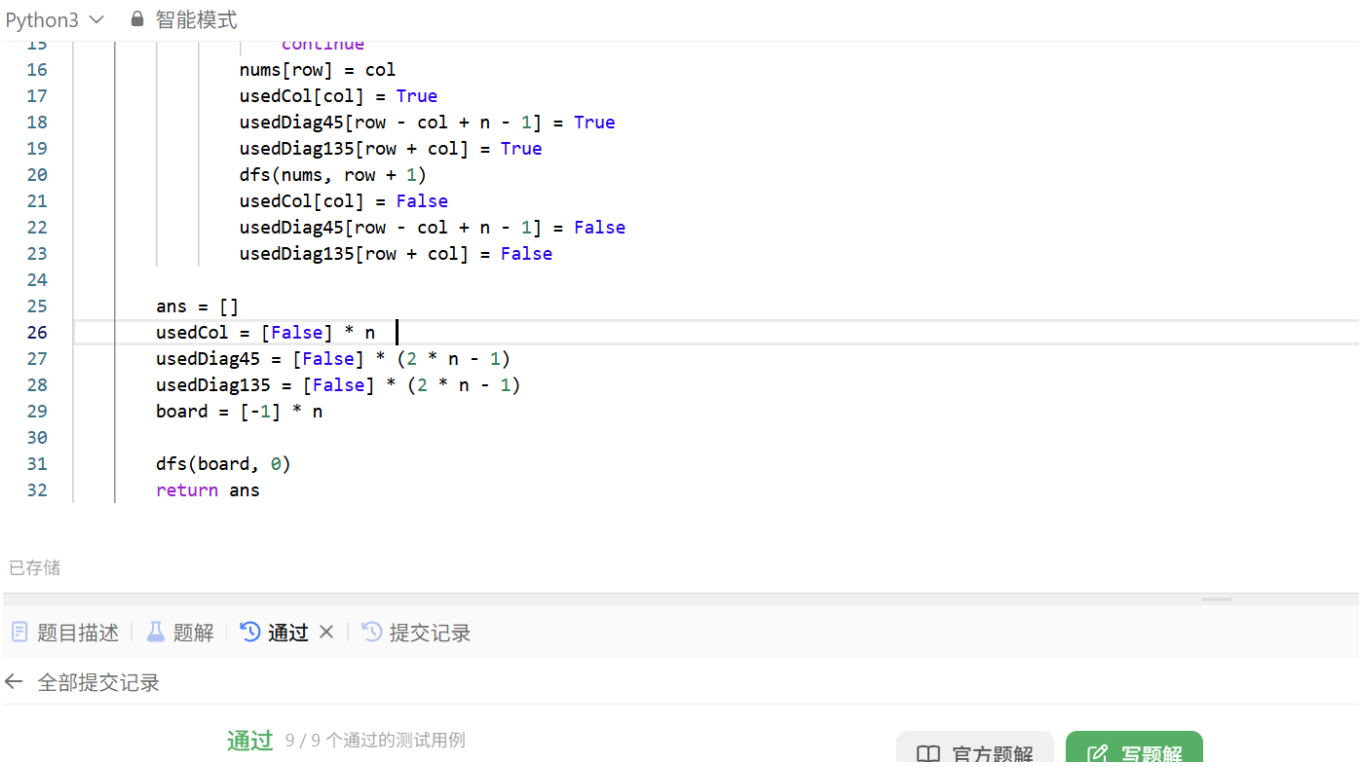

image.png

用时：30min左右

M22275: 二叉搜索树的遍历

http://cs101.openjudge.cn/practice/22275/

思路：

代码：

```
def postorder_from_preorder(preorder):
    if not preorder:
        return []
    root = preorder[0]
    left_subtree = []
    right_subtree = []
    for i in range(1, len(preorder)):
        if preorder[i] < root:
            left_subtree.append(preorder[i])
        else:
            right_subtree.append(preorder[i])

    return postorder_from_preorder(left_subtree) +
postorder_from_preorder(right_subtree) + [root]

n = int(input().strip())
```

```
preorder = list(map(int, input().strip().split()))
postorder = postorder_from_preorder(preorder)
print(' '.join(map(str, postorder)))
```

代码运行截图 （至少包含有"Accepted"）

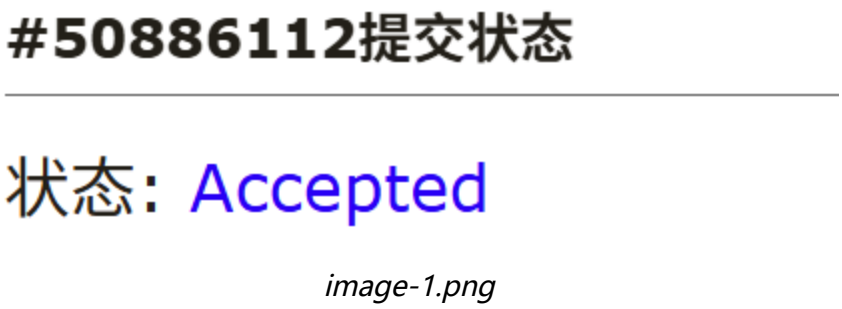

image-1.png

用时：30min左右

M25145: 猜二叉树（按层次遍历）

http://cs101.openjudge.cn/practice/25145/

思路：

关键是重建二叉树的过程。根据后序遍历和中序遍历的性质，找到root，学会递归建树。

代码：

```
from collections import deque

def build_tree(inorder, postorder):
    if not inorder or not postorder:
        return None
    root_val = postorder[-1]
    root = {'val': root_val, 'left': None, 'right': None}
    root_index = inorder.index(root_val)
    root['left'] = build_tree(inorder[:root_index], postorder[:root_index])
    root['right'] = build_tree(inorder[root_index + 1:],
postorder[root_index:-1])
    return root

def levelorder(root):
    if not root:
        return ""

    result = []
    queue = deque([root])
```

```
    while queue:
        node = queue.popleft()
        result.append(node['val'])
        if node['left']:
            queue.append(node['left'])
        if node['right']:
            queue.append(node['right'])

    return ''.join(result)

n = int(input().strip())
results = []
for _ in range(n):
    inorder = input().strip()
    postorder = input().strip()
    tree = build_tree(inorder, postorder)
    level_order = levelorder(tree)
    results.append(level_order)
for result in results:
    print(result)
```

代码运行截图 （至少包含有"Accepted"）

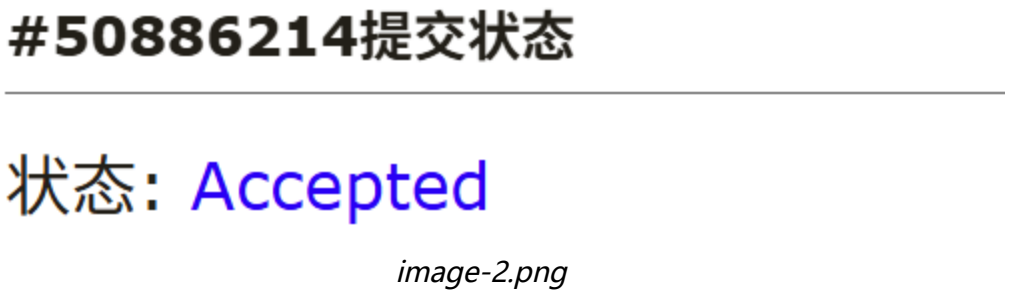

image-2.png

用时：50min左右

T20576: printExp（逆波兰表达式建树）

http://cs101.openjudge.cn/practice/20576/

思路：

代码

```
class TreeNode:
    def __init__(self, value):
        self.value = value
        self.left = None
        self.right = None
```

```python
def infix_to_postfix(expression):
    precedence = {'not': 3, 'and': 2, 'or': 1}

    output = []
    stack = []
    tokens = expression.split()

    for token in tokens:
        if token in ['True', 'False']:
            output.append(token)
        elif token == '(':
            stack.append(token)
        elif token == ')':
            while stack and stack[-1] != '(':
                output.append(stack.pop())
            stack.pop()  # 弹出 '('
        elif token in ['and', 'or', 'not']:
            while (stack and stack[-1] != '(' and
                   precedence.get(stack[-1], 0) >= precedence[token]):
                output.append(stack.pop())
            stack.append(token)

    while stack:
        output.append(stack.pop())

    return output

def build_expression_tree(postfix):
    stack = []

    for token in postfix:
        if token in ['True', 'False']:
            stack.append(TreeNode(token))
        elif token == 'not':
            node = TreeNode(token)
            node.left = stack.pop()
            stack.append(node)
        else:  # 'and', 'or'
            node = TreeNode(token)
            node.right = stack.pop()
            node.left = stack.pop()
            stack.append(node)

    return stack[0]

def tree_to_infix(root, parent_precedence=0):
```

```
    if root.value in ['True', 'False']:
        return root.value

    precedence = {'not': 3, 'and': 2, 'or': 1}

    if root.value == 'not':
        operand = tree_to_infix(root.left, precedence['not'])

        if (root.left.value in ['and', 'or'] and
                precedence[root.left.value] < precedence['not']):
            return f"not ({operand})"
        else:
            return f"not {operand}"
    else:
        left_str = tree_to_infix(root.left, precedence[root.value])
        right_str = tree_to_infix(root.right, precedence[root.value])

        if (root.left.value in ['and', 'or'] and
                precedence[root.left.value] < precedence[root.value]):
            left_str = f"({left_str})"

        if (root.right.value in ['and', 'or'] and
                precedence[root.right.value] <= precedence[root.value]):
            right_str = f"({right_str})"

        return f"{left_str} {root.value} {right_str}"

expression = input()
postfix = infix_to_postfix(expression)
tree = build_expression_tree(postfix)
print(tree_to_infix(tree))
```

代码运行截图（至少包含有"Accepted"）

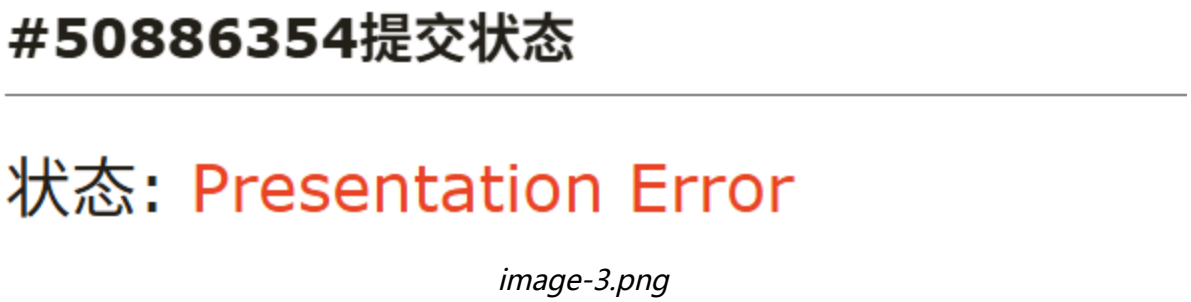

image-3.png

T04080:Huffman编码树

greedy, http://cs101.openjudge.cn/practice/04080/

思路：

代码

```
```

代码运行截图（至少包含有"Accepted"）

M04078: 实现堆结构

http://cs101.openjudge.cn/practice/04078/

要求手搓堆实现。

思路：

代码：

```
```

代码运行截图（至少包含有"Accepted"）

2. 学习总结和个人收获

曾经，天真的我以为，Tree相关的东西已十分有限，如今才堪堪知道什么叫树据结构😭😭