Assignment #4: 排序、栈、队列和树

Updated 0005 GMT+8 March 11, 2024

2024 spring, Complied by 同学的姓名、院系

说明：

1）The complete process to learn DSA from scratch can be broken into 4 parts:

Learn about Time complexities, learn the basics of individual Data Structures, learn the basics of Algorithms, and practice Problems.

2）请把每个题目解题思路（可选），源码Python, 或者C++（已经在Codeforces/Openjudge上AC），截图（包含Accepted），填写到下面作业模版中（推荐使用 typora https://typoraio.cn ，或者用word）。AC 或者没有AC，都请标上每个题目大致花费时间。

3）提交时候先提交pdf文件，再把md或者doc文件上传到右侧“作业评论”。Canvas需要有同学清晰头像、提交文件有pdf、"作业评论"区有上传的md或者doc附件。

4）如果不能在截止前提交作业，请写明原因。

编程环境

（请改为同学的操作系统、编程环境等）

操作系统：macOS Ventura 13.4.1 (c)

Python编程环境：Spyder IDE 5.2.2, PyCharm 2023.1.4 (Professional Edition)

C/C++编程环境：Mac terminal vi (version 9.0.1424), g++/gcc (Apple clang version 14.0.3, clang-1403.0.22.14.1)

1. 题目

05902: 双端队列

http://cs101.openjudge.cn/practice/05902/

思路：直接impot deque。

一开始考虑复杂了，以为会有退多了的情况。

代码

```
# from collections import deque

n = int(input())
for i in range(n):
    m = int(input())
    d = deque()
```

```
    for j in range(m):
        (t, c) = [int(x) for x in input().split()]
        if t == 1:
            d.append(c)
        elif t == 2:
            if c == 0:
                d.popleft()
            elif c == 1:
                d.pop()
    if len(d) == 0:
        print("NULL")
    else:
        print(' '.join(map(str, d)))
```

代码运行截图 （至少包含有"Accepted"）

状态: Accepted

源代码

```
from collections import deque

n = int(input())
for i in range(n):
    m = int(input())
    d = deque()
    for j in range(m):
        (t, c) = [int(x) for x in input().split()]
        if t == 1:
            d.append(c)
        elif t == 2:
            if c == 0:
                d.popleft()
            elif c == 1:
                d.pop()
    if len(d) == 0:
        print("NULL")
    else:
        print(' '.join(map(str, d)))
```


02694: 波兰表达式

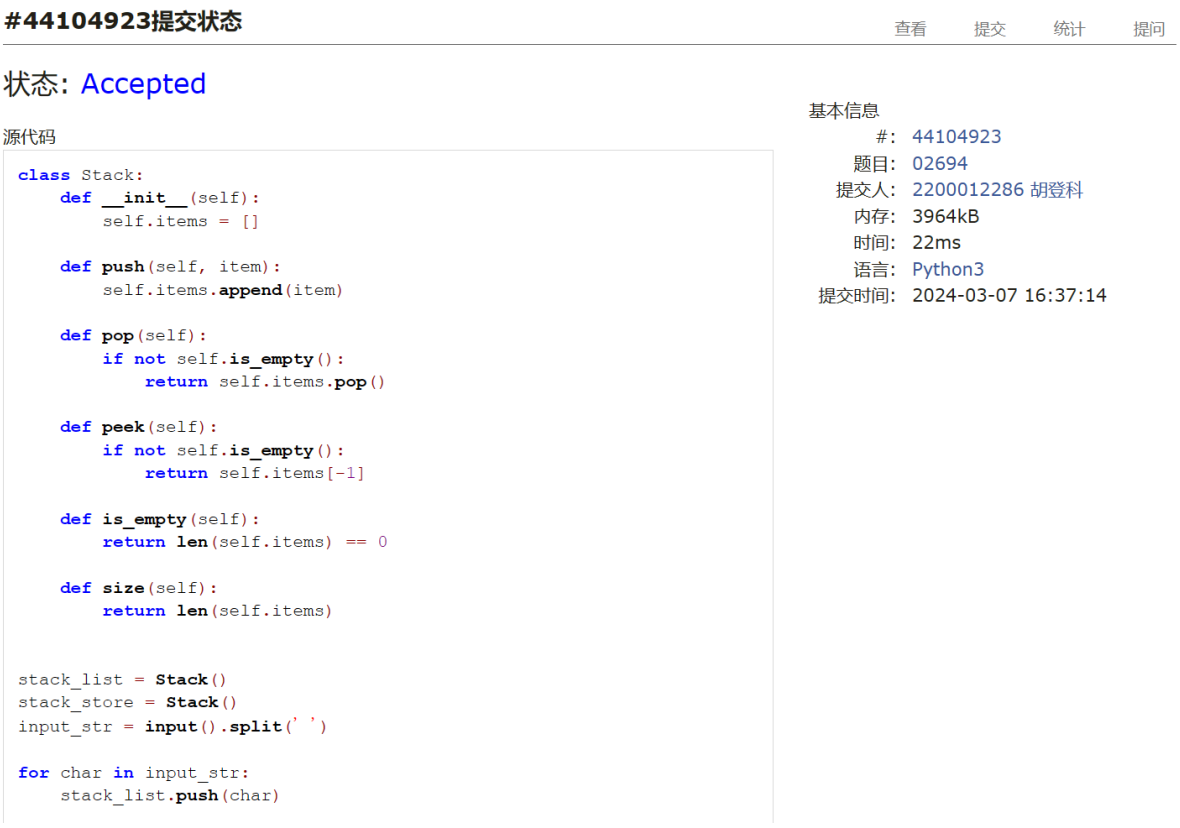
http://cs101.openjudge.cn/practice/02694/

思路：利用栈的基本问题，联系一下类的基本写法，准备笔试。

代码

```
# class Stack:
    def __init__(self):
        self.items = []

    def push(self, item):
        self.items.append(item)

    def pop(self):
        if not self.is_empty():
            return self.items.pop()

    def peek(self):
        if not self.is_empty():
            return self.items[-1]

    def is_empty(self):
        return len(self.items) == 0

    def size(self):
        return len(self.items)


stack_list = Stack()
stack_store = Stack()
input_str = input().split(' ')

for char in input_str:
    stack_list.push(char)

while stack_list.size() != 0:
    a = stack_list.pop()
    if a == '+':
        b = float(stack_store.pop())
        c = float(stack_store.pop())
        d = float(c + b)
        stack_store.push(d)
    elif a == '-':
        b = float(stack_store.pop())
        c = float(stack_store.pop())
        d = float(b - c)
        stack_store.push(d)
    elif a == '*':
        b = float(stack_store.pop())
        c = float(stack_store.pop())
        d = float(c * b)
        stack_store.push(d)
    elif a == '/':
        b = float(stack_store.pop())
        c = float(stack_store.pop())
        d = float(b / c)
        stack_store.push(d)
    else:
        stack_store.push(a)
e = "{:.6f}".format(stack_store.pop())
print(e)
```

代码运行截图 （至少包含有"Accepted"）



#44104923提交状态

查看 提交 统计 提问

状态: Accepted

源代码

```
class Stack:
    def __init__(self):
        self.items = []

    def push(self, item):
        self.items.append(item)

    def pop(self):
        if not self.is_empty():
            return self.items.pop()

    def peek(self):
        if not self.is_empty():
            return self.items[-1]

    def is_empty(self):
        return len(self.items) == 0

    def size(self):
        return len(self.items)


stack_list = Stack()
stack_store = Stack()
input_str = input().split(' ')

for char in input_str:
    stack_list.push(char)
```

基本信息

#: 44104923
题目: 02694
提交人: 2200012286 胡登科
内存: 3964kB
时间: 22ms
语言: Python3
提交时间: 2024-03-07 16:37:14

24591: 中序表达式转后序表达式

http://cs101.openjudge.cn/practice/24591/

思路:

初看以为简单双栈问题，细读发现比较困难，原来是调度场。

代码

```
#
#
def infix_to_postfix(expression):
    precedence = {'+': 1, '-': 1, '*': 2, '/': 2}
    stack = []
    postfix = []
    number = ''

    for char in expression:
        if char.isnumeric() or char == '.':
            number += char
        else:
            if number:
                num = float(number)
```

```
                postfix.append(int(num) if num.is_integer() else num)
                number = ''
            if char in '+-*/':
                while stack and stack[-1] in '+-*/' and precedence[char] <=
precedence[stack[-1]]:
                    postfix.append(stack.pop())
                stack.append(char)
            elif char == '(':
                stack.append(char)
            elif char == ')':
                while stack and stack[-1] != '(':
                    postfix.append(stack.pop())
                stack.pop()

    if number:
        num = float(number)
        postfix.append(int(num) if num.is_integer() else num)

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

    return ' '.join(str(x) for x in postfix)


n = int(input())
for _ in range(n):
    expression = input()
    print(infix_to_postfix(expression))
```

代码运行截图 （AC代码截图，至少包含有"Accepted"）

状态: Accepted

源代码

```
# 简单的双栈问题，但是有括号，www难
# 搞错了是调度场算法 www 学习学习
def infix_to_postfix(expression):
    precedence = {'+': 1, '-': 1, '*': 2, '/': 2}
    stack = []
    postfix = []
    number = ''

    for char in expression:
        if char.isnumeric() or char == '.':
            number += char
        else:
            if number:
                num = float(number)
                postfix.append(int(num) if num.is_integer() else num)
                number = ''
            if char in '+-*/':
                while stack and stack[-1] in '+-*/' and precedence[char]
                    postfix.append(stack.pop())
                stack.append(char)
            elif char == '(':
                stack.append(char)
            elif char == ')':
                while stack and stack[-1] != '(':
                    postfix.append(stack.pop())
                stack.pop()

    if number:
        num = float(number)
        postfix.append(int(num) if num.is_integer() else num)

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

    return ' '.join(str(x) for x in postfix)


n = int(input())
for _ in range(n):
    expression = input()
    print(infix_to_postfix(expression))
```

22068: 合法出栈序列

http://cs101.openjudge.cn/practice/22068/

思路：合法出栈序列的思路很巧妙，避免用全排列的复杂度。本题最后While True 的用法之前也没见过。

while True:
    try:

except EOFError:

break

这个也像极了我学python的态度。

代码

```
# 合法出栈队列 再次练习
def is_valid_stack_pop_sequence(origin, output):
    if len(origin) != len(output):
        return False

    stack = []
    bank = list(origin)

    for char in output:
        # 入栈条件
        while (not stack or stack[-1] != char) and bank:
            stack.append(bank.pop(0))  # 左边比右边先入栈
        # 非法序列
        if not stack or stack[-1] != char:
            return False

        stack.pop()
    return True


origin = input().strip()

while True:
    try:
        output = input().strip()
        if is_valid_stack_pop_sequence(origin, output):
            print('YES')
        else:
            print('NO')
    except EOFError:
        break
```

代码运行截图 （AC代码截图，至少包含有"Accepted"）

状态: Accepted

源代码

```
# 合法出栈队列 再次练习
def is_valid_stack_pop_sequence(origin, output):
    if len(origin) != len(output):
        return False

    stack = []
    bank = list(origin)

    for char in output:
        # 入栈条件
        while (not stack or stack[-1] != char) and bank:
            stack.append(bank.pop(0))  # 左边比右边先入栈
        # 非法序列
        if not stack or stack[-1] != char:
            return False

        stack.pop()
    return True


origin = input().strip()

while True:
    try:
        output = input().strip()
        if is_valid_stack_pop_sequence(origin, output):
            print('YES')
        else:
            print('NO')
    except EOFError:
        break
```


06646: 二叉树的深度

http://cs101.openjudge.cn/practice/06646/

思路：递归树节点，找到多的链接。感觉这种树结构是真的漂亮。

代码

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


# 递归找最长
```

```
def tree_depth(node):
    if node is None:
        return 0
    left_depth = tree_depth(node.left)
    right_depth = tree_depth(node.right)
    return max(left_depth, right_depth) + 1


n = int(input())  # 读取节点数量
nodes = [TreeNode() for _ in range(n)]

for i in range(n):
    left_index, right_index = map(int, input().split())
    if left_index != -1:
        nodes[i].left = nodes[left_index - 1]
    if right_index != -1:
        nodes[i].right = nodes[right_index - 1]

root = nodes[0]
depth = tree_depth(root)
print(depth)
```

代码运行截图 （AC代码截图，至少包含有"Accepted"）

状态：Accepted

源代码

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


# 递归找最长
def tree_depth(node):
    if node is None:
        return 0
    left_depth = tree_depth(node.left)
    right_depth = tree_depth(node.right)
    return max(left_depth, right_depth) + 1


n = int(input())  # 读取节点数量
nodes = [TreeNode() for _ in range(n)]

for i in range(n):
    left_index, right_index = map(int, input().split())
    if left_index != -1:
        nodes[i].left = nodes[left_index - 1]
    if right_index != -1:
        nodes[i].right = nodes[right_index - 1]

root = nodes[0]
depth = tree_depth(root)
print(depth)
```

02299: Ultra-QuickSort

http://cs101.openjudge.cn/practice/02299/

思路：

归并算法，分治思想。但是变一下形就又不会了。要算交换了多少次。

代码

```
#
def merge_sort(lst):
    if len(lst) <= 1:
        return lst, 0

    middle = len(lst) // 2
    left, inv_left = merge_sort(lst[:middle])
    right, inv_right = merge_sort(lst[middle:])

    merged, inv_merge = merge(left, right)
```

```
    # The total number of inversions is the sum of inversions in the recursion
and the merge process.
    return merged, inv_left + inv_right + inv_merge


def merge(left, right):
    merged = []
    inv_count = 0
    i = j = 0

    # Merge smaller elements first.
    while i < len(left) and j < len(right):
        if left[i] <= right[j]:
            merged.append(left[i])
            i += 1
        else:
            merged.append(right[j])
            j += 1
            inv_count += len(left) - i  # left[i~mid)都比right[j]要大，他们都会与
right[j]构成逆序对，将他们加入答案

    # If there are remaining elements in the left or right half, append them to
the result.
    merged += left[i:]
    merged += right[j:]

    return merged, inv_count


while True:
    n = int(input())
    if n == 0:
        break

    lst = []
    for _ in range(n):
        lst.append(int(input()))

    _, inversions = merge_sort(lst)
    print(inversions)
```

代码运行截图 （AC代码截图，至少包含有"Accepted"）

#44279498提交状态

状态: Accepted

源代码

```
def merge_sort(lst):
    if len(lst) <= 1:
        return lst, 0

    middle = len(lst) // 2
    left, inv_left = merge_sort(lst[:middle])
    right, inv_right = merge_sort(lst[middle:])

    merged, inv_merge = merge(left, right)

    # The total number of inversions is the sum of inversions in the re
    return merged, inv_left + inv_right + inv_merge


def merge(left, right):
    merged = []
    inv_count = 0
    i = j = 0

    # Merge smaller elements first.
    while i < len(left) and j < len(right):
        if left[i] <= right[j]:
            merged.append(left[i])
            i += 1
        else:
            merged.append(right[j])
            j += 1
            inv_count += len(left) - i  # left[i~mid)都比right[j]要大，他

    # If there are remaining elements in the left or right half, append
    merged += left[i:]
    merged += right[j:]

    return merged, inv_count
```

2. 学习总结和收获

如果作业题目简单，有否额外练习题目，比如：OJ“2024spring每日选做”、CF、LeetCode、洛谷等网站题目。

感觉这周题目比上周简单，可能是之前没接触过太多的计概算法。而对于数据结构来说理解起来更成系统一些。而且不少题目的逻辑也仔细思考过，例如merge的分治思想，以及数的深度的递归过程都是上课讲过的部分。这周的作业让我复习了不少，是一种思路指导代码的过程。但是调度场算法还是理解的不太直观。希望每周学习都有收获，不追求分数，只追求知识。