Assignment #4: 排序、栈、队列和树

Updated 0005 GMT+8 March 11, 2024

2024 spring, Complied by ==韩萱 工学院==

我的课程主页https://github.com/hanxuan0422/2024spring-cs201

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：Windows 11 家庭中文版 22H2

Python编程环境：Visual Studio Code

C/C++编程环境：Visual Studio Code

1. 题目

05902: 双端队列

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

思路：数据结构用class实现一下就行

代码

```
class deque():
    def __init__(self):
        self.items = []
    def isEmpty(self):
        return self.items == []
    def addFront(self, item):
        self.items.append(item)
```

```
    def addRear(self, item):
        self.items.insert(0,item)
    def removeFront(self):
        return self.items.pop()
    def removeRear(self):
        return self.items.pop(0)
    def size(self):
        return len(self.items)


t = int(input())
for i in range(t):
    n = int(input())
    d = deque()
    for j in range(n):
        a, b = map(int, input().split())
        if a == 1:
            d.addRear(b)
        else:
            if b == 0:
                d.removeFront()
            else:
                d.removeRear()
    if d.isEmpty():
        print("NULL")
    else:
        #把deque的items逆序输出
        print(" ".join(map(str, d.items[::-1])))
```

代码运行截图 ==（至少包含有"Accepted"）==

02694: 波兰表达式

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

思路：用栈，判断好什么时候pop就行

代码

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

    def is_empty(self):
        return self.items == []

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

    def pop(self):
        return self.items.pop()
```

```
    def peek(self):
        return self.items[len(self.items) - 1]

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

l = list(input().split())

for i in l:
    if i in '+-*/':
        continue
    else:
        l[l.index(i)] = float(i)

def convert(l):
    s = Stack()
    for i in l:
        if type(i) == str:
            s.push(i)
        else:
            if type(s.items[-1]) == float and type(s.items[-2]) == str:
                a = float(s.pop())
                b = s.pop()
                if b == '+':
                    s.push(a + i)
                elif b == '-':
                    s.push(a - i)
                elif b == '*':
                    s.push(a * i)
                elif b == '/':
                    s.push(a / i)
            else:
                s.push(float(i))
    return s.items

while len(l) > 1:
    l = convert(l)

print("{:.6f}".format(l[0]))
```

代码运行截图 ==（至少包含有"Accepted"）==

24591: 中序表达式转后序表达式

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

思路：照着讲义写的

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"）==

22068: 合法出栈序列

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

思路：用栈检查

代码

```
def is_valid_pop_sequence(origin, output):
    if len(origin) != len(output):
        return False  # 长度不同，直接返回False
```

```
    stack = []
    bank = list(origin)

    for char in output:
        # 如果当前字符不在栈顶，且bank中还有字符，则继续入栈
        while (not stack or stack[-1] != char) and bank:
            stack.append(bank.pop(0))

        # 如果栈为空，或栈顶字符不匹配，则不是合法的出栈序列
        if not stack or stack[-1] != char:
            return False

        stack.pop()  # 匹配成功，弹出栈顶元素

    return True  # 所有字符都匹配成功

# 读取原始字符串
origin = input().strip()

# 循环读取每一行输出序列并判断
while True:
    try:
        output = input().strip()
        if is_valid_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"）==

06646: 二叉树的深度

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

思路：和讲义不同的是，编号从0到n-1->从1到n，求高度->深度，改改就行

代码

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

def tree_height(node):
    if node is None:
        return -1  # 根据定义，空树高度为-1
    return max(tree_height(node.left), tree_height(node.right)) + 1

def count_leaves(node):
    if node is None:
```

```
            return 0
        if node.left is None and node.right is None:
            return 1
        return count_leaves(node.left) + count_leaves(node.right)

    n = int(input())  # 读取节点数量
    nodes = [TreeNode() for _ in range(n+1)]
    has_parent = [False] * (n+1)  # 用来标记节点是否有父节点

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

    # 寻找根节点，也就是没有父节点的节点
    root_index = has_parent[1:].index(False)+1
    root = nodes[root_index]

    # 计算高度
    height = tree_height(root)

    print(height+1)
```

代码运行截图 ==（AC代码截图，至少包含有"Accepted"）==

02299: Ultra-QuickSort

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

思路： 本题大意：对于给定的无序数组，求出经过最少多少次相邻元素的交换之后，可以使数组从小到大有序

对于逆序对的解释：两个数（a, b）的排列，若满足a > b，则称之为一个逆序对

归并排序的思想：先把每个数看成一段，然后两两合并成一个较大的有序数组，再把较大的两两合并，直到最后成为一个有序数组

归并排序复杂度： O(nlogn)

接下来就是我们解题的步骤了：（P.S：听说这道题还是到数状树组的板子题，可惜本蒟蒻想了半天想不出来数状树组解法，于是就来了一发归并排序）

根据排序算法，我们知道如果相邻的两个元素满足前一个大于后一个便会交换一次，由于题目要求排序后是单调递增，所以我们可以将这道题看做求原数组逆序对的数量

举一个归并排序的例子：

假设初始数组为4 2 1 3

先把每一个数单独分成一组，即(4) (2) (1) (3)

接着两两合并，即(2 4) (1 3)

最后合成一个有序的数组，即(1 2 3 4)

不难发现，在排序过程中，若某个数向前移动了N位，则必定存在N个逆序数,如上面例子中，数字1由原先的第三位移到了第一位，前移了两位，则存在(2 1)和(4 1)两个逆序

而根据题意，我们只需要在归并排序的过程中把这个数记下来即可

代码

```
global ans
ans = 0
def merge_sort(a):
    global ans
    if len(a) > 1:
        mid = len(a) // 2
        left = a[:mid]
        right = a[mid:]
        merge_sort(left)
        merge_sort(right)
        i = j = k = 0
        while i < len(left) and j < len(right):
            if left[i] < right[j]:
                a[k] = left[i]
                i += 1
            else:
                a[k] = right[j]
                j += 1
                ans += len(left) - i
            k += 1
        while i < len(left):
            a[k] = left[i]
            i += 1
            k += 1
        while j < len(right):
            a[k] = right[j]
            j += 1
            k += 1
    return a

while True:
    n = int(input())
    if n == 0:
        break
    a = []
    for i in range(n):
        a.append(int(input()))
    ans = 0
```

```
    a = merge_sort(a)
    print(ans)
```

代码运行截图 ==（AC代码截图，至少包含有"Accepted"）==

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。==

```
#把deque的items逆序输出
#在Python中，列表的切片操作可以接受三个参数：开始索引、结束索引和步长。当步长为-1时，切片操作将从后向前选择元素，因此[::-1]就表示将整个列表反向。
print(" ".join(map(str, d.items[::-1])))

#输出float小数
print("{:.6f}".format(ans))
print('%f' %ans) #默认输出六位
```

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

def tree_height(node):
    if node is None:
        return -1  # 根据定义，空树高度为-1
    return max(tree_height(node.left), tree_height(node.right)) + 1

def count_leaves(node):
    if node is None:
        return 0
    if node.left is None and node.right is None:
        return 1
    return count_leaves(node.left) + count_leaves(node.right)

n = int(input())  # 读取节点数量
nodes = [TreeNode() for _ in range(n+1)]
has_parent = [False] * (n+1)  # 用来标记节点是否有父节点

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

```
# 寻找根节点，也就是没有父节点的节点
root_index = has_parent[1:].index(False)+1
root = nodes[root_index]

# 计算高度
height = tree_height(root)

print(height+1)
```