Assignment #4: 排序、栈、队列和树

2024 spring, Complied by 胡豪俊 工学院

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

操作系统：Windows11

Python编程环境：Visual Studio Code

1. 题目

05902: 双端队列

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

代码

```
n=int(input())
ans=[]
for i in range(n):
    m=int(input())
    L=[]
    for j in range(m):
        w=[int(i) for i in input().split()]
        x=w[0]
        y=w[1]
        if x==1:
            L.append(str(y))
        if x==2:
            if len(L)==0:
                ans.append("NULL")
                break
            elif y==0:
                L.pop(0)
```

```
            elif y==1:
                L.pop()
    t=" ".join(L)
    if t=="":
        ans.append("NULL")
    else:
        ans.append(t)
for t in ans:
    print(t)
```

##一开始想用deque实现来着，除去一个addFront就可以了，但是后来想想还是再直接写一次，不同方法都要练习

#44295425提交状态

查看 提交 统计 提问

状态: Accepted

源代码

```
n=int(input())
ans=[]
for i in range(n):
    m=int(input())
    L=[]
    for j in range(m):
        w=[int(i) for i in input().split()]
        x=w[0]
        y=w[1]
        if x==1:
            L.append(str(y))
        if x==2:
            if len(L)==0:
                ans.append("NULL")
                break
            elif y==0:
                L.pop(0)
            elif y==1:
                L.pop()
    t=" ".join(L)
    if t=="":
        ans.append("NULL")
    else:
        ans.append(t)
for t in ans:
    print(t)
```

基本信息

#: 44295425
题目: 05902
提交人: 22n2200011800
内存: 3660kB
时间: 50ms
语言: Python3
提交时间: 2024-03-19 10:48:24

English 帮助 关于

02694: 波兰表达式

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

代码

```
class Stack:
    def __init__(self):
        self.items=[]
    def is_empty(self):
        return self.items==[]
    def peek(self):
        return self.items[len(self.items)-1]
    def pop(self):
        return self.items.pop()
    def push(self,item):
        self.items.append(item)
    def size(self):
        return len(self.items)
def prefixEval(postfixExpression):
```

```
    operandStack=Stack()
    tokenList=postfixExpression.split()
    tokenList.reverse()
    for token in tokenList:
        if token not in "+-*/":
            operandStack.push(float(token))
        else:
            operand2=operandStack.pop()
            operand1=operandStack.pop()
            result=doMath(token,operand1,operand2)
            operandStack.push(result)
    return operandStack.pop()

def doMath(operator,operand1,operand2):
    if operator=="*":
        return operand1*operand2
    elif operator=="/":
        return operand2/operand1
    elif operator=="+":
        return operand1+operand2
    else:
        return operand2-operand1
expression=input()
print('%.6f'% prefixEval(expression))
##这题当年在计算概论的时候写过，看了之前ac的代码，感觉当年的做法完全是只为了解决问题，毫无美感可言
##这回用了栈来写，思路也挺清晰，核心是算符附近的两个元素进行计算，这题和后序表达式求值是一样的，反过来遍历即可，不过要注意除法和减法是不交换的，修改一下计算函数即可
```

#44295565提交状态

查看 提交 统计 提问

状态: Accepted

基本信息

#: 44295565
题目: 02694
提交人: 22n2200011800
内存: 3572kB
时间: 24ms
语言: Python3
提交时间: 2024-03-19 10:59:45

源代码

```
class Stack:
    def __init__(self):
        self.items=[]
    def is_empty(self):
        return self.items==[]
    def peek(self):
        return self.items[len(self.items)-1]
    def pop(self):
        return self.items.pop()
    def push(self,item):
        self.items.append(item)
    def size(self):
        return len(self.items)
def prefixEval(postfixExpression):
    operandStack=Stack()
    tokenList=postfixExpression.split()
    tokenList.reverse()
    for token in tokenList:
        if token not in "+-*/":
            operandStack.push(float(token))
        else:
            operand2=operandStack.pop()
            operand1=operandStack.pop()
            result=doMath(token,operand1,operand2)
            operandStack.push(result)
    return operandStack.pop()

def doMath(operator,operand1,operand2):
    if operator=="*":
        return operand1*operand2
    elif operator=="/":
        return operand2/operand1
    elif operator=="+":
        return operand1+operand2
    else:
        return operand2-operand1
expression=input()
print('%.6f' % prefixEval(expression))
```


English 帮助 关于

24591: 中序表达式转后序表达式

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

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)

def infixToPostfix(infixexpression):
    precedence={}
    precedence["*"]=3
    precedence["/"]=3
    precedence["+"]=2
    precedence["-"]=2
    precedence["("]=1
    operatorStack=Stack()
```

```
    postfixList=[]
    tokenList=str(infixexpression)
    number=""
    for i in range(len(tokenList)):
        token=tokenList[i]
        if token.isnumeric() or token==".":
            number+=token
            if i==len(tokenList)-1:
                num=float(number)
                postfixList.append(str(int(num)) if num.is_integer() else
str(num))
                number=""
        else:
            if number:
                num=float(number)
                postfixList.append(str(int(num)) if num.is_integer() else
str(num))
                number=""
            if token=="(":
                operatorStack.push(token)
            elif token==")":
                topToken=operatorStack.pop()
                while topToken!="(":
                    postfixList.append(topToken)
                    topToken=operatorStack.pop()
            else:
                while not operatorStack.is_empty() and
precedence[operatorStack.peek()]>=precedence[token]:
                    postfixList.append(operatorStack.pop())
                operatorStack.push(token)
    while not operatorStack.is_empty():
        postfixList.append(operatorStack.pop())
    return " ".join(postfixList)


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

##这题...其实思路是很直接的，肯定需要遍历，数字的相对位置在最终输出的表达式中是不会发生变化的，要考虑的就是括号和运算符号，我觉得最需要注意的有两点，一是括号可以看作对某一块计算过程得到结合，每读到一个右括号说明这部分算完了，我们可以处理，第二是运算符号在后序表达式中连续出现的情况，这就需要考虑优先级的问题，优先级高的在输出在前，注意这点整体思路就没问题了

##然而...想要做对一道题目不仅仅需要整体的思路理顺，细节也很重要。对于符号表示，或者就个位数的计算，完全按上述的想法加上对字符串的判断就可以了，但问题坏就坏在这里是真正的数字，我一开始考虑到了小数，但是没有想到十位数百位数，自作聪明去掉了小数点前后的空格然后输出，错了。后面用了参考题解的方式搞到了数字和符号的列表，但是使用类的时候Stack()的这个括号漏了，导致在oj上compile error了半天，最后终于解决了

##收获很大的一道题目，栈的运用的思路一下子被打开了

#44295584提交状态

查看 提交 统计 提问

状态: Accepted

基本信息

#: 44295584
题目: 24591
提交人: 22n2200011800
内存: 3728kB
时间: 29ms
语言: Python3
提交时间: 2024-03-19 11:00:42

源代码

```
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)

def infixToPostfix(infixexpression):
    precedence={}
    precedence["*"]=3
    precedence["/"]=3
    precedence["+"]=2
    precedence["-"]=2
    precedence["("]=1
    operatorStack=Stack()
    postfixList=[]
    tokenList=str(infixexpression)
    number=""
    for i in range(len(tokenList)):
        token=tokenList[i]
        if token.isnumeric() or token==".":
            number+=token
            if i==len(tokenList)-1:
                num=float(number)
                postfixList.append(str(int(num)) if num.is_integer() else
                number=""
        else:
            if number:
                num=float(number)
                postfixList.append(str(int(num)) if num.is_integer() else
                number=""
            if token=="(":
                operatorStack.push(token)
            elif token==")":
                topToken=operatorStack.pop()
                while topToken!="(":
                    postfixList.append(topToken)
                    topToken=operatorStack.pop()
            else:
                while not operatorStack.is_empty() and precedence[operato
                    postfixList.append(operatorStack.pop())
                operatorStack.push(token)
    while not operatorStack.is_empty():
        postfixList.append(operatorStack.pop())
    return " ".join(postfixList)


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

22068: 合法出栈序列

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

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)

def test_output_sequence(string,output):
    manipulate_Stack=Stack()
    origin=list(string)
    if len(string)!=len(output):
        return "NO"
    for x in output:
        while (manipulate_Stack.is_empty() or manipulate_Stack.peek()!=x) and
origin:
            manipulate_Stack.push(origin.pop(0))

        if manipulate_Stack.is_empty() or manipulate_Stack.peek()!=x:
            return "NO"

        manipulate_Stack.pop()

    return "YES"
string=input().strip()
while True:
    try:
        output=input().strip()
        print(test_output_sequence(string,output))

    except EOFError:
        break
##核心是用栈去模拟出栈行为，把没遇到的都压入栈中，压入的出现了对应的符号则弹出，继续压直到压完，
此时只会弹出，那么看看栈的peek上是否对应弹出的东西即可，如果不是就错了，全部匹配上就符合
##坑的地方在于output可以和原来不一样长，看了题解才发现...
```

#44298174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 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)

def test_output_sequence(string,output):
    manipulate_Stack=Stack()
    origin=list(string)
    if len(string)!=len(output):
        return "NO"
    for x in output:
        while (manipulate_Stack.is_empty() or manipulate_Stack.peek()!=x
            manipulate_Stack.push(origin.pop(0))

        if manipulate_Stack.is_empty() or manipulate_Stack.peek()!=x:
            return "NO"

        manipulate_Stack.pop()

    return "YES"
string=input().strip()
while True:
    try:
        output=input().strip()
        print(test_output_sequence(string,output))

    except EOFError:
        break
```

基本信息

#: 44298174
题目: 22068
提交人: 22n2200011800
内存: 3588kB
时间: 29ms
语言: Python3
提交时间: 2024-03-19 15:34:52

English 帮助 关于

06646: 二叉树的深度

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

代码

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

def build_tree(node_list):
    nodes = {i: TreeNode(i) for i in range(1, len(node_list) + 1)}
    for i, (left, right) in enumerate(node_list, 1):
        if left != -1:
            nodes[i].left = nodes[left]
        if right != -1:
            nodes[i].right = nodes[right]
    return nodes[1]

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

```
n = int(input())
node_list = []
for _ in range(n):
    left, right = map(int, input().split())
    node_list.append((left, right))
root = build_tree(node_list)
depth = max_depth(root)

print(depth)
```

##参考了题解，感觉对树还不是很熟悉，当年学计算概论的时候用到过类似的思想，现在学了类能更好地结合起来，对于树的问题我要找更多题目练练手

#44303958提交状态

查看 提交 统计 提问

状态: Accepted

源代码

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

def build_tree(node_list):
    nodes = {i: TreeNode(i) for i in range(1, len(node_list) + 1)}
    for i, (left, right) in enumerate(node_list, 1):
        if left != -1:
            nodes[i].left = nodes[left]
        if right != -1:
            nodes[i].right = nodes[right]
    return nodes[1]

def max_depth(root):
    if root is None:
        return 0
    else:
        left_depth = max_depth(root.left)
        right_depth = max_depth(root.right)
        return max(left_depth, right_depth) + 1

n = int(input())
node_list = []
for _ in range(n):
    left, right = map(int, input().split())
    node_list.append((left, right))
root = build_tree(node_list)
depth = max_depth(root)

print(depth)
```

基本信息

#: 44303958
题目: 06646
提交人: 22n2200011800
内存: 3652kB
时间: 29ms
语言: Python3
提交时间: 2024-03-19 20:25:55

English 帮助 关于

02299: Ultra-QuickSort

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

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)

    return merged, inv_left + inv_right + inv_merge
```

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

    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

    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)
```

##这题我写了半天都wa，最后看题解感觉和我自己的思路一致就是不知道错哪了，最后发现有个i打成j了，比较之下还是题解思路更清晰，我就用了题解

##其实思路很简单，这个固定的算法只能交换临近的数，那么其实就是求逆序数了，最开始我的思路是找出最小值，放到最前面，然后反复操作，类似冒泡的那种想法，然后tle了，后面看到群里老师发的同学的思路想到了merge，求逆序数，把列表分成两段，其逆序数等于前面的逆序数加上后面的逆序数再加上前面相对后面的逆序数，而最后一部分正是在merge中的互插能很好地体现并且求解出来，而前面两部分则可以递归求解，很是方便，而且时间复杂度由于算法更优被大大降低了，最后也是这样过了

#44303293提交状态

查看 提交 统计 提问

状态: Accepted

源代码

```python
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)

    return merged, inv_left + inv_right + inv_merge

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

    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

    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)
```

基本信息

#:	44303293
题目:	02299
提交人:	22n2200011800
内存:	29832kB
时间:	3930ms
语言:	Python3
提交时间:	2024-03-19 19:52:52

English 帮助 关于

2. 学习总结和收获

变成完全的ddl战士了...这周仔细地把老师之前发的部分资料看了一遍，努力做到理解并且能独立求解里面的典型问题，每日选做挑了几题，熟悉了一下栈队列等数据结构，因为我感觉对这些的把握还不是很好，题目难起来可以很难，我要加深对这些东西的理解，继续努力吧！