Assignment #4: 排序、栈、队列和树

Updated 0005 GMT+8 March 11, 2024

2024 spring, Complied by 数学科学学院 王镜廷 2300010724

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：Windows11 专业版

Python编程环境：VSCode 1.86.2, with extension Python and python version 3.12.2

1. 题目

05902: 双端队列

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

用时：约20分钟

思路：
用两个对顶的栈来实现，当有一个栈为空时进行重构，从而时间复杂度均摊到每次操作上是O（1）的

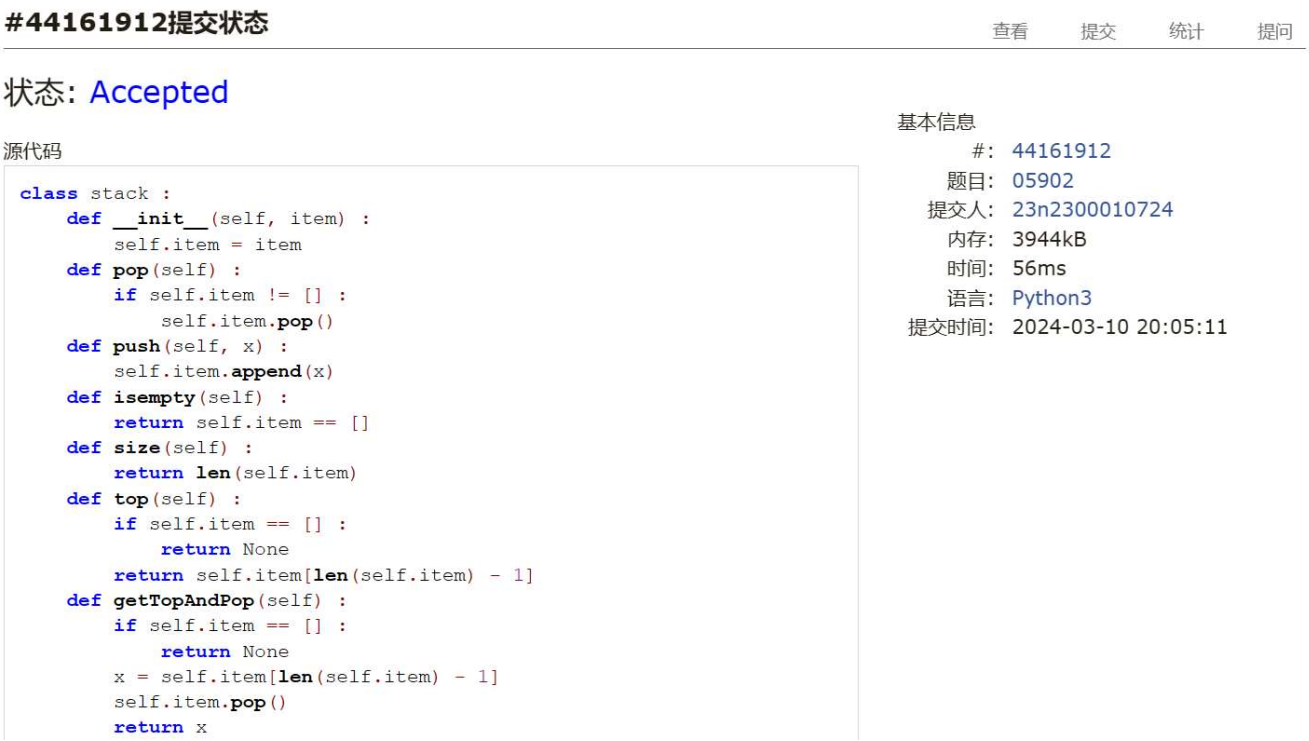
代码

```
class stack :
    def __init__(self, item) :
        self.item = item
    def pop(self) :
        if self.item != [] :
            self.item.pop()
    def push(self, x) :
        self.item.append(x)
    def isempty(self) :
        return self.item == []
    def size(self) :
        return len(self.item)
    def top(self) :
        if self.item == [] :
            return None
        return self.item[len(self.item) - 1]
    def getTopAndPop(self) :
        if self.item == [] :
            return None
        x = self.item[len(self.item) - 1]
        self.item.pop()
        return x
    def __str__(self) :
        return str(self.item)

class deque :
    left = stack([])
    right = stack([])
    def __init__(self, left, right) :
        left.reverse()
        self.left = stack(left)
        self.right = stack(right)
    def balanced(self) :
        if self.left.size() != 0 and self.right.size() != 0 :
            return True
        if self.left.size() <= 1 and self.right.size() <= 1 :
            return True
        return False
    def rebalance(self) :
        if self.balanced() :
            return
        if self.left.size() == 0 :
            n = self.right.size()
            m = n // 2
            s = self.right.item
            s1 = s[0 : m]
            s1.reverse()
            self.left = stack(s1)
```

```
            self.right = stack(s[m : n])
        if self.right.size() == 0 :
            n = self.left.size()
            m = n // 2
            s = self.left.item
            s.reverse()
            s1 = s[0 : m]
            s1.reverse()
            self.left = stack(s1)
            self.right = stack(s[m : n])
            return
    def size(self) :
        return self.left.size() + self.right.size()
    def front(self) : #in a balanced deque
        if self.size() == 0 :
            return None
        if self.left.size() == 0 :
            return self.right.item[0]
        return self.left.top()
    def end(self) : #in a balanced deque
        if self.size() == 0 :
            return None
        if self.right.size() == 0 :
            return self.left.item[0]
        return self.right.top()
    def pushfront(self, x) :
        self.left.push(x)
        self.rebalance()
    def pushback(self, x) :
        self.right.push(x)
        self.rebalance()
    def popfront(self) : #in a balanced deque
        if self.left.size() == 0 :
            self.right.pop()
        else :
            self.left.pop()
        self.rebalance()
    def popback(self) :
        if self.right.size() == 0 :
            self.left.pop()
        else :
            self.right.pop()
        self.rebalance()

T = int(input())
for case in range(T) :
    n = int(input())
    h = deque([], [])
    flag = True
```

```
for i in range(n) :
    op, x = map(int, input().split())
    if op == 1 :
        h.pushback(x)
    else :
        if x == 0 :
            h.popfront()
        else :
            h.popback()
        #if h.size() == 0 :
        #    flag = False
if h.size() == 0 :
    print("NULL")
else :
    s1 = h.left.item
    s1.reverse()
    s2 = h.right.item
    print(" ".join(str(item) for item in s1) + " " + " ".join(str(item) for item in s2))
```

代码运行截图 （至少包含有"Accepted"）



#44161912提交状态

查看 提交 统计 提问

状态: Accepted

源代码

```
class stack :
    def __init__(self, item) :
        self.item = item
    def pop(self) :
        if self.item != [] :
            self.item.pop()
    def push(self, x) :
        self.item.append(x)
    def isempty(self) :
        return self.item == []
    def size(self) :
        return len(self.item)
    def top(self) :
        if self.item == [] :
            return None
        return self.item[len(self.item) - 1]
    def getTopAndPop(self) :
        if self.item == [] :
            return None
        x = self.item[len(self.item) - 1]
        self.item.pop()
        return x
```

基本信息

#: 44161912
题目: 05902
提交人: 23n2300010724
内存: 3944kB
时间: 56ms
语言: Python3
提交时间: 2024-03-10 20:05:11

02694: 波兰表达式

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

用时：约10分钟

思路：
写函数递归实现计算即可（也可使用下一题中定义的SyntaxTree类）

代码

```
#
tot = 0
s = []
def solve() :
    global tot
    global s
    c = s[tot]
    tot += 1
    if c == '+' :
        return solve() + solve()
    if c == '-' :
        return solve() - solve()
    if c == '*' :
        return solve() * solve()
    if c == "/" :
        return solve() / solve()
    else :
        return float(c)


s = input().split()
print(f"{solve():.6f}")
```

使用下一题所写的类代码如下

```
def isSign(c) :
    if c == "+" or c == "-" or c == "*" or c == "/" :
        return True
    else :
        return False
def isSignOrBracket(c) :
    if isSign(c) or c == "(" or c == ")" :
        return True
    else :
        return False
def SignLevel(c) :
    if isSign(c) :
        if c == ")" :
            return 10
        if c == "+" or c == "-" :
            return 20
        if c == "*" or c == "/" :
            return 30
        if c == "(" :
            return 40
    return -1

class stack :
    def __init__(self, item) :
        self.item = item
    def top(self) :
        if self.item == [] :
            return None
        else :
            return self.item[len(self.item) - 1]
    def pop(self) :
        if self.item == [] :
            return
        else :
            self.item.pop()
    def getTopAndPop(self) :
        if self.item == [] :
            return None
        else :
            x = self.top()
            self.pop()
            return x
    def isempty(self) :
        if self.item == [] :
            return True
        else :
            return False
    def push(self, elem) :
```

```
        self.item.append(elem)
    def size(self) :
        return len(self.item)

class SyntaxTree :
    def __init__(self, left, right, this) :
        self.this = this
        self.left = left
        self.right = right
    def toPrefixExpr(self) :
        c = self.this
        if not isSign(c):
            return c
        else :
            return self.this + " " + self.left.toPrefixExpr() + " " + self.right.toPrefixExpr()
    def toPostfixExpr(self) :
        c = self.this
        if not isSign(c) :
            return c
        else :
            return self.left.toPostfixExpr() + " " + self.right.toPostfixExpr() + " " + self.this
    def eval(self) :
        c = self.this
        if c == "+" :
            return self.left.eval() + self.right.eval()
        elif c == "-" :
            return self.left.eval() - self.right.eval()
        elif c == "*" :
            return self.left.eval() * self.right.eval()
        elif c == "/" :
            return self.left.eval() / self.right.eval()
        else :
            return float(c)

def SplitIntoNum(s) :
    res = []
    tmp = []
    flag = True
    for c in s :
        if c == " " :
            if not flag :
                res.append("".join(str(i) for i in tmp))
                tmp = []
                flag = True
        elif isSignOrBracket(c) :
            if not flag :
                res.append("".join(str(i) for i in tmp))
                tmp = []
                flag = True
```

```
            res.append(c)
        else :
            tmp.append(c)
            flag = False
    if not flag :
        res.append("".join(str(i) for i in tmp))
    return res

def PostfixToSyntaxTree(s) : # No brackets in Postfix Exprs
    q = stack([])
    for c in s :
        if isSign(c) :
            if q.size() <= 1 :
                print("failure while converting postfix expression to syntax tree")
                return None
            else :
                y = q.getTopAndPop()
                x = q.getTopAndPop()
                q.push(SyntaxTree(x, y, c))
        else :
            q.push(SyntaxTree(None, None, c))
    return q.top()

def PrefixToSyntaxTree(s) : # No brackets in Prefix Exprs
    q = stack([])
    s.reverse()
    for c in s :
        if isSign(c) :
            if q.size() <= 1 :
                print("failure while converting prefix expression to syntax tree")
                return None
            else :
                x = q.getTopAndPop()
                y = q.getTopAndPop()
                q.push(SyntaxTree(x, y, c))
        else :
            q.push(SyntaxTree(None, None, c))
    return q.top()

def InfixToSyntaxTree(s) : # with brackets in Infix Exprs
    qTree = stack([])
    qSign = stack([])
    for c in s :
        if not isSignOrBracket(c) :
            qTree.push(SyntaxTree(None, None, c))
        elif c == "(" :
            qSign.push(c)
        elif c != ")" :
            while (not qSign.isempty()) and SignLevel(qSign.top()) >= SignLevel(c) :
```

```
                c1 = qSign.getTopAndPop()
                if qTree.size() <= 1 :
                    print("failure while converting infix expression to syntax tree")
                    return None
                y = qTree.getTopAndPop()
                x = qTree.getTopAndPop()
                qTree.push(SyntaxTree(x, y, c1))
                # print(qTree.top().toPostfixExpr())
            qSign.push(c)
        else :
            while (not qSign.isempty()) and qSign.top() != "(" :
                c1 = qSign.getTopAndPop()
                if qTree.size() <= 1 :
                    print("failure while converting infix expression to syntax tree")
                    return None
                y = qTree.getTopAndPop()
                x = qTree.getTopAndPop()
                qTree.push(SyntaxTree(x, y, c1))
                # print(qTree.top().toPostfixExpr())
            if qSign.isempty() :
                print("failure while converting infix expression to syntax tree")
                return None
            qSign.pop()
        # print(qSign.item)
    while not qSign.isempty():
        c1 = qSign.getTopAndPop()
        if qTree.size() <= 1 :
            print("failure while converting infix expression to syntax tree")
            return None
        y = qTree.getTopAndPop()
        x = qTree.getTopAndPop()
        qTree.push(SyntaxTree(x, y, c1))
        # print(qTree.top().toPostfixExpr())
    return qTree.top()

s = input()
s = SplitIntoNum(s)
print(f"{PrefixToSyntaxTree(s).eval():.6f}")
```

代码运行截图 （至少包含有"Accepted"）

#44042256提交状态 查看 提交 统计 提问

状态: Accepted

源代码

```
tot = 0
s = []
def solve() :
    global tot
    global s
    c = s[tot]
    tot += 1
    if c == '+' :
        return solve() + solve()
    if c == '-' :
        return solve() - solve()
    if c == '*' :
        return solve() * solve()
    if c == "/" :
        return solve() / solve()
    else :
        return float(c)


s = input().split()
print(f"{solve():.6f}")
```

基本信息

#: 44042256
题目: 02694
提交人: 23n2300010724
内存: 3612kB
时间: 21ms
语言: Python3
提交时间: 2024-03-02 19:53:29

#44179434提交状态 查看 提交 统计 提问

状态: Accepted

源代码

```
def isSign(c) :
    if c == "+" or c == "-" or c == "*" or c == "/" :
        return True
    else :
        return False
def isSignOrBracket(c) :
    if isSign(c) or c == "(" or c == ")" :
        return True
    else :
        return False
def SignLevel(c) :
    if isSign(c) :
        if c == ")" :
            return 10
        if c == "+" or c == "-" :
            return 20
        if c == "*" or c == "/" :
            return 30
```

基本信息

#: 44179434
题目: 02694
提交人: 23n2300010724
内存: 3928kB
时间: 23ms
语言: Python3
提交时间: 2024-03-12 09:58:45

24591: 中序表达式转后序表达式

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

用时：约1小时

思路：
实现一个类SyntaxTree，先把中序表达式转换为SyntaxTree，再将SyntaxTree转化为后序表达式

代码

```
def isSign(c) :
    if c == "+" or c == "-" or c == "*" or c == "/" :
        return True
    else :
        return False
def isSignOrBracket(c) :
    if isSign(c) or c == "(" or c == ")" :
        return True
    else :
        return False
def SignLevel(c) :
    if isSign(c) :
        if c == ")" :
            return 10
        if c == "+" or c == "-" :
            return 20
        if c == "*" or c == "/" :
            return 30
        if c == "(" :
            return 40
    return -1

class stack :
    def __init__(self, item) :
        self.item = item
    def top(self) :
        if self.item == [] :
            return None
        else :
            return self.item[len(self.item) - 1]
    def pop(self) :
        if self.item == [] :
            return
        else :
            self.item.pop()
    def getTopAndPop(self) :
        if self.item == [] :
            return None
        else :
            x = self.top()
            self.pop()
            return x
    def isempty(self) :
        if self.item == [] :
            return True
        else :
            return False
    def push(self, elem) :
```

```
        self.item.append(elem)
    def size(self) :
        return len(self.item)

class SyntaxTree :
    def __init__(self, left, right, this) :
        self.this = this
        self.left = left
        self.right = right
    def toPrefixExpr(self) :
        c = self.this
        if not isSign(c):
            return c
        else :
            return self.this + " " + self.left.toPrefixExpr() + " " + self.right.toPrefixExpr()
    def toPostfixExpr(self) :
        c = self.this
        if not isSign(c) :
            return c
        else :
            return self.left.toPostfixExpr() + " " + self.right.toPostfixExpr() + " " + self.this
    def eval(self) :
        c = self.this
        if c == "+" :
            return self.left.eval() + self.right.eval()
        elif c == "-" :
            return self.left.eval() - self.right.eval()
        elif c == "*" :
            return self.left.eval() * self.right.eval()
        elif c == "/" :
            return self.left.eval() / self.right.eval()
        else :
            return float(c)

def SplitIntoNum(s) :
    res = []
    tmp = []
    flag = True
    for c in s :
        if c == " " :
            if not flag :
                res.append("".join(str(i) for i in tmp))
                tmp = []
                flag = True
        elif isSignOrBracket(c) :
            if not flag :
                res.append("".join(str(i) for i in tmp))
                tmp = []
                flag = True
```

```
            res.append(c)
        else :
            tmp.append(c)
            flag = False
    if not flag :
        res.append("".join(str(i) for i in tmp))
    return res

def PostfixToSyntaxTree(s) : # No brackets in Postfix Exprs
    q = stack([])
    for c in s :
        if isSign(c) :
            if q.size() <= 1 :
                print("failure while converting postfix expression to syntax tree")
                return None
            else :
                y = q.getTopAndPop()
                x = q.getTopAndPop()
                q.push(SyntaxTree(x, y, c))
        else :
            q.push(SyntaxTree(None, None, c))
    return q.top()

def PrefixToSyntaxTree(s) : # No brackets in Prefix Exprs
    q = stack([])
    s.reverse()
    for c in s :
        if isSign(c) :
            if q.size() <= 1 :
                print("failure while converting prefix expression to syntax tree")
                return None
            else :
                x = q.getTopAndPop()
                y = q.getTopAndPop()
                q.push(SyntaxTree(x, y, c))
        else :
            q.push(SyntaxTree(None, None, c))
    return q.top()

def InfixToSyntaxTree(s) : # with brackets in Infix Exprs
    qTree = stack([])
    qSign = stack([])
    for c in s :
        if not isSignOrBracket(c) :
            qTree.push(SyntaxTree(None, None, c))
        elif c == "(" :
            qSign.push(c)
        elif c != ")" :
            while (not qSign.isempty()) and SignLevel(qSign.top()) >= SignLevel(c) :
```

```
                c1 = qSign.getTopAndPop()
                if qTree.size() <= 1 :
                    print("failure while converting infix expression to syntax tree")
                    return None
                y = qTree.getTopAndPop()
                x = qTree.getTopAndPop()
                qTree.push(SyntaxTree(x, y, c1))
                # print(qTree.top().toPostfixExpr())
            qSign.push(c)
        else :
            while (not qSign.isempty()) and qSign.top() != "(" :
                c1 = qSign.getTopAndPop()
                if qTree.size() <= 1 :
                    print("failure while converting infix expression to syntax tree")
                    return None
                y = qTree.getTopAndPop()
                x = qTree.getTopAndPop()
                qTree.push(SyntaxTree(x, y, c1))
                # print(qTree.top().toPostfixExpr())
            if qSign.isempty() :
                print("failure while converting infix expression to syntax tree")
                return None
            qSign.pop()
        # print(qSign.item)
    while not qSign.isempty():
        c1 = qSign.getTopAndPop()
        if qTree.size() <= 1 :
            print("failure while converting infix expression to syntax tree")
            return None
        y = qTree.getTopAndPop()
        x = qTree.getTopAndPop()
        qTree.push(SyntaxTree(x, y, c1))
        # print(qTree.top().toPostfixExpr())
    return qTree.top()

T = int(input())
for Case in range(T) :
    s = input()
    s = SplitIntoNum(s)
    print(InfixToSyntaxTree(s).toPostfixExpr())
```

代码运行截图 （AC代码截图，至少包含有"Accepted"）

#44138257提交状态

查看 提交 统计 提问

状态: Accepted

基本信息

#: 44138257
题目: 24591
提交人: 23n2300010724
内存: 5684kB
时间: 47ms
语言: Python3
提交时间: 2024-03-09 17:50:44

源代码

```
def isSign(c) :
    if c == "+" or c == "-" or c == "*" or c == "/" :
        return True
    else :
        return False
def isSignOrBracket(c) :
    if isSign(c) or c == "(" or c == ")" :
        return True
    else :
        return False
def SignLevel(c) :
    if isSign(c) :
        if c == ")" :
            return 10
        if c == "+" or c == "-" :
            return 20
        if c == "*" or c == "/" :
            return 30
        if c == "(" :
            return 40
    return -1
```

22068: 合法出栈序列

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

用时：约15分钟

思路：
直接模拟，注意因为字符互异，每个时刻如果栈顶的元素不是输出序列的下一个元素则必定压栈，否则必定弹栈

代码

```
class stack :
    def __init__(self, item) :
        self.item = item
    def top(self) :
        if self.item == [] :
            return None
        else :
            return self.item[len(self.item) - 1]
    def pop(self) :
        if self.item == [] :
            return
        else :
            self.item.pop()
    def isempty(self) :
        if self.item == [] :
            return True
        else :
            return False
    def push(self, elem) :
        self.item.append(elem)

def check(s, s1) :
    if len(s) != len(s1) :
        return False
    i = 0
    n = len(s)
    q = stack([])
    j = 0
    while i <= n and j < n:
        if q.top() == s1[j] :
            q.pop()
            j += 1
            continue
        elif i < n:
            q.push(s[i])
            i += 1
        else :
            break
        #print(f"{i} {j} {q.item}")
    if j == n :
        return True
    else :
        return False

s = input()
while(True) :
    try :
        s1 = input()
```

```
        if check(s, s1) :
            print("YES")
        else :
            print("NO")
    except :
        break
```

代码运行截图 （AC代码截图，至少包含有"Accepted"）

#44130449提交状态

查看 提交 统计 提问

状态: Accepted

源代码

```
class stack :
    def __init__(self, item) :
        self.item = item
    def top(self) :
        if self.item == [] :
            return None
        else :
            return self.item[len(self.item) - 1]
    def pop(self) :
        if self.item == [] :
            return
        else :
            self.item.pop()
    def isempty(self) :
        if self.item == [] :
            return True
        else :
            return False
```

基本信息

#: 44130449
题目: 22068
提交人: 23n2300010724
内存: 3680kB
时间: 26ms
语言: Python3
提交时间: 2024-03-09 13:31:17

06646: 二叉树的深度

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

用时：约10分钟

思路：
直接递归即可

代码

```python
class node :
    def __init__ (self, left, right) :
        self.left = left
        self.right = right
nodePool = [None]
depth = []

def getdepth(u) :
    global nodePool
    if u == -1 :
        return 0
    if depth[u] != 0 :
        return depth[u]
    depth[u] = max(getdepth(nodePool[u].left), getdepth(nodePool[u].right)) + 1
    return depth[u]

n = int(input())
depth = [0] * (n + 1)
for i in range(n) :
    u, v = map(int, input().split())
    nodePool.append(node(u, v))
print(getdepth(1))
```

代码运行截图 （AC代码截图，至少包含有"Accepted"）

#44150982提交状态

查看 提交 统计 提问

状态: Accepted

源代码

```python
class node :
    def __init__ (self, left, right) :
        self.left = left
        self.right = right
nodePool = [None]
depth = []

def getdepth(u) :
    global nodePool
    if u == -1 :
        return 0
    if depth[u] != 0 :
        return depth[u]
    depth[u] = max(getdepth(nodePool[u].left), getdepth(nodePool[u].righ
    return depth[u]

n = int(input())
depth = [0] * (n + 1)
for i in range(n) :
    u, v = map(int, input().split())
    nodePool.append(node(u, v))
print(getdepth(1))
```

基本信息

#:	44150982
题目:	06646
提交人:	23n2300010724
内存:	3612kB
时间:	22ms
语言:	Python3
提交时间:	2024-03-10 13:42:45

English 帮助 关于

02299: Ultra-QuickSort

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

用时：约20分钟

思路：
注意到所求即为序列的逆序对个数，使用归并排序即可在$O(n \log n)$的时间复杂度内完成计算。（也可离散化后使用树状数组实现）

代码

```
s = []
res = 0

def mergesort(l, r) :
    global s
    global res
    if l == r :
        return
    m = (l + r) // 2
    mergesort(l, m)
    mergesort(m + 1, r)
    s1 = s[l : m + 1]
    s2 = s[m + 1 : r + 1]
    n1 = len(s1)
    n2 = len(s2)
    pos1 = 0
    pos2 = 0
    for i in range(l, r + 1) :
        if pos1 == n1 :
            s[i] = s2[pos2]
            pos2 += 1
            continue
        if pos2 == n2 or s1[pos1] <= s2[pos2]:
            s[i] = s1[pos1]
            pos1 += 1
            res += pos2
        else :
            s[i] = s2[pos2]
            pos2 += 1

while(True) :
    n = int(input())
    s = []
    res = 0
    if n == 0 :
        break
    for i in range(n) :
        s.append(int(input()))
    mergesort(0, n - 1)
    print(res)
```

代码运行截图 （AC代码截图，至少包含有"Accepted"）

#44168276提交状态

查看 提交 统计 提问

状态: Accepted

源代码

```
s = []
res = 0

def mergesort(l, r) :
    global s
    global res
    if l == r :
        return
    m = (l + r) // 2
    mergesort(l, m)
    mergesort(m + 1, r)
    s1 = s[l : m + 1]
    s2 = s[m + 1 : r + 1]
    n1 = len(s1)
    n2 = len(s2)
    pos1 = 0
    pos2 = 0
    for i in range(l, r + 1) :
        if pos1 == n1 :
            s[i] = s2[pos2]
            pos2 += 1
            continue
        if pos2 == n2 or s1[pos1] <= s2[pos2]:
            s[i] = s1[pos1]
            pos1 += 1
            res += pos2
```

基本信息

#: 44168276
题目: 02299
提交人: 23n2300010724
内存: 25052kB
时间: 3356ms
语言: Python3
提交时间: 2024-03-11 13:51:11

2. 学习总结和收获

这周在练习中进一步熟悉了类的写法，同时在中序表达式转后序表达式一题中尝试实现了相对较多的功能。并在那一题所实现的类的基础上写了一个计算24点的程序。