

# Assignment #4: 排序、栈、队列和树

---

Updated 0005 GMT+8 March 11, 2024

2024 spring, Compiled by 城环 吴至超

## 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

Python编程环境：pycharm2023.2.3

## 1. 题目

---

### 05902: 双端队列

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

思路：简单，正常语法

代码

```
# t=int(input())
for i in range(t):
    queue=[]
    n=int(input())
    for u in range(n):#读取操作数据
        x,y=input().split()
        if x=="1":
            queue.append(y)
        if x=="2" and y=="0":
```

```

        queue.pop(0)
    if x=="2" and y=="1":
        queue.pop()
if queue:
    print(" ".join(map(str,queue)))
else:
    print("NULL")

```

代码运行截图 == (至少包含有"Accepted") ==

OpenJudge
题目ID, 标题, 描述
23n2300013289
信箱
账号

CS101 / 题库
题目 排名 状态 提问

### 05902:双端队列

查看 提交 统计 提问

总时间限制: 1000ms 内存限制: 65535kB

**描述**

定义一个双端队列，进队操作与普通队列一样，从队尾进入。出队操作既可以从队头，也可以从队尾，编程实现这个数据结构。

**输入**

第一行输入一个整数t，代表测试数据的组数。

每组数据的第一行输入一个整数n，表示操作的次数。

接着输入n行，每行对应一个操作，首先输入一个整数type。

当type=1，进队操作，接着输入一个整数x，表示进入队列的元素。

当type=2，出队操作，接着输入一个整数c，c=0代表从队头出队，c=1代表从队尾出队。

n <= 1000

**输出**

对于每组测试数据，输出执行完所有的操作后队列中剩余的元素,元素之间用空格隔开，按队头到队尾的顺序输出，占一行。如果队列中已经没有任何的元素，输出NULL。

**样例输入**

```

2
5
1 2
1 3
1 4
2 0
2 1
6
1 1
1 2
1 3

```

全局题号 **5902**  
添加于 **2023-12-14**  
提交次数 **163**  
尝试人数 **90**  
通过人数 **90**  
Other language versions  
English  
你的提交记录  

#	结果	时间
1	Accepted	2024-03-13

## 02694: 波兰表达式

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

思路：用栈的思路比较简单，递归的思路比较巧妙。

eval () 可以运行字符串中的运算式子。

代码

```

# #波兰表达式
s=list(map(str,input().split()))
def bolan():
    a=s.pop(0)
    if a in "+-*/":
        return str(eval(bolan()+a+bolan()))
    else:
        return a
print(f"{float(bolan()):.6f}")

```

代码运行截图 == (至少包含有"Accepted") ==

OpenJudge 题目ID, 标题, 描述 23n2300013289 信箱 账号

CS101 / 题库

题目 排名 状态 提问

#44111545提交状态 查看 提交 统计 提问

状态: Accepted

源代码

```

#波兰表达式
s=list(map(str,input().split()))
def bolan():
    a=s.pop(0)
    if a in "+-*/":
        return str(eval(bolan()+a+bolan()))
    else:
        return a
print(f"{float(bolan()):.6f}")

```

基本信息

#:	44111545
题目:	02694
提交人:	23n2300013289
内存:	3592kB
时间:	20ms
语言:	Python3
提交时间:	2024-03-07 22:33:51

## 24591: 中序表达式转后序表达式

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

思路: shunting yard算法

代码

```

#
n=int(input())

#建立一个符号优先级
char={"+":1,"-":1,"*":2,"/":2,"(":0}
for i in range(n):
    sample=input()
    #处理一下数据
    samlist=list(sample)
    sam_list=[]
    tempo = ""
    for p in samlist:
        if p in "+-*/()":
            if tempo:
                sam_list.append(tempo)
                tempo = ""

```

```

        tempo=""
        sam_list.append(p)
    else:
        if p.isdigit() or p==".":
            tempo+=p
    if tempo!="":
        sam_list.append(tempo)
caculate=[]#运算集合
pre_print=[]#输出集合
for b in sam_list:
    if b=="(":
        caculate.append(b)
        #接下来遇到运算符：如果优先级比caculate的栈顶大或者caculate的栈顶是左括号或者是
        #caculate是空集，则加入caculate，否则一直弹出caculate栈顶直到满足条件为止
    elif b in "+-*/" :
        #判断空集先放前面
        while (caculate and char[b] <= char[caculate[-1]]):
            pre_print.append(caculate.pop())
        caculate.append(b)
    elif b==")":
        while caculate[-1]!="(":
            pre_print.append(caculate.pop())
        caculate.pop()
    else:
        pre_print.append(b)
if caculate:
    pre_print.extend(caculate[::-1])
print(" ".join(map(str,pre_print)))

```

代码运行截图 == (AC代码截图，至少包含有"Accepted") ==

OpenJudge
题目ID, 标题, 描述
23n2300013289
信箱
账号

CS101 / 题库
题目
排名
状态
提问

### 24591:中序表达式转后序表达式

查看
提交
统计
提问

总时间限制：1000ms 内存限制：65536kB

**描述**

中序表达式是运算符放在两个数中间的表达式。乘、除运算优先级高于加减。可以用"()"来提升优先级 --- 就是小学生写的四则算术运算表达式。中序表达式可用如下方式递归定义：

- 1) 一个数是一个中序表达式。该表达式的值就是数的值。
- 2) 若a是中序表达式，则"(a)"也是中序表达式(引号不算)，值为a的值。
- 3) 若a,b是中序表达式，c是运算符，则"acb"是中序表达式。"acb"的值是对a和b做c运算的结果，且a是左操作数，b是右操作数。

输入一个中序表达式，要求转换成一个后序表达式输出。

**输入**

第一行是整数n(n<100)。接下来n行，每行一个中序表达式，数和运算符之间没有空格，长度不超过700。

**输出**

对每个中序表达式，输出转成后序表达式后的结果。后序表达式的数之间、数和运算符之间用一个空格分开。

全局题号 **24591**  
添加于 **2024-02-02**  
提交次数 **156**  
尝试人数 **62**  
通过人数 **59**  
你的提交记录

#	结果	时间
3	Accepted	2024-03-13
2	Wrong Answer	2024-03-13
1	Wrong Answer	2024-03-13

## 22068: 合法出栈序列

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

思路：栈的思想

代码

```
#
yuan=input()
while True:
    try:
        stack=[]
        a=input()#测试样例
        c=list(yuan)
        flag=0
        if len(a)!=len(yuan):
            print("NO")

        else:
            for i in a:#对待输入数据的一个个字母进行判断
                while ( not stack or stack[-1]!=i) and c:
                    stack.append(c.pop(0))

                if stack[-1]==i:
                    stack.pop(-1)
                else:
                    flag=1
                    break
            if flag==0:
                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") ==

OpenJudge

题目ID, 标题, 描述

23n2300013289

信箱

账号

CS101 / 2024spring每日选做

[题目](#)[排名](#)[状态](#)[提问](#)

#44068686提交状态

[查看](#)[提交](#)[统计](#)[提问](#)

状态: Accepted

源代码

```
yuan=input()
while True:
    try:
        stack=[]
        a=input() #测试样例
        c=list(yuan)
        flag=0
        if len(a)!=len(yuan):
            print("NO")

        else:
            for i in a: #对待输入数据的一个个字母进行判断
                while ( not stack or stack[-1]!=i) and c:
                    stack.append(c.pop(0))

                if stack[-1]==i:
                    stack.pop(-1)
                else:
                    flag=1
                    break
            if flag==0:
                print("YES")
            else:
                print("NO")
        except EOFError:
            break
```

基本信息

#: 44068686  
题目: 22068  
提交人: 23n2300013289  
内存: 3628kB  
时间: 24ms  
语言: Python3  
提交时间: 2024-03-04 19:35:13

©2002-2022 POJ 京ICP备20010980号-1

[English](#)[帮助](#)[关于](#)

## 06646: 二叉树的深度

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

思路: 构建class 树。

代码

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

n=int(input())
ab=[treeNode() for _ in range(n)]#列表中每一位都是treeNode化的实例
for m in range(n):
    ab[m].left,ab[m].right=map(int,input().split())

def measure(treeNode,ab):
    if treeNode.left!=-1 and treeNode.right!=-1:
        return 1+max(measure(ab[treeNode.left-1],ab),measure(ab[treeNode.right-1],ab))
    elif treeNode.left==-1 and treeNode.right==-1:#基本结束条件
        return 1
    elif treeNode.left!=-1 and treeNode.right==-1:
        return 1+measure(ab[treeNode.left-1],ab)
    else:
        return 1+measure(ab[treeNode.right-1],ab)
```

```
print(measure(ab[0],ab))
```

代码运行截图 == (AC代码截图, 至少包含有"Accepted") ==



OpenJudge 题目ID, 标题, 描述 23n2300013289 信箱 账号

CS101 / 数算pre每日选做

题目 排名 状态 提问

#44250919提交状态 查看 提交 统计 提问

状态: Accepted

源代码

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

n=int(input())
ab=[treeNode() for _ in range(n)] #列表中每一位都是treeNode化的实例
for m in range(n):
    ab[m].left,ab[m].right=map(int,input().split())

def measure(treeNode,ab):
    if treeNode.left!=-1 and treeNode.right!=-1:
        return 1+max(measure(ab[treeNode.left-1],ab),measure(ab[treeNode.right-1],ab))
    elif treeNode.left!=-1 and treeNode.right==1:
        return 1
    elif treeNode.left==1 and treeNode.right!=-1:
        return 1+measure(ab[treeNode.left-1],ab)
    else:
        return 1+measure(ab[treeNode.right-1],ab)
print(measure(ab[0],ab))
```

基本信息

#:	44250919
题目:	06646
提交人:	23n2300013289
内存:	3628kB
时间:	23ms
语言:	Python3
提交时间:	2024-03-16 17:33:03

©2002-2022 POJ 京ICP备20010980号-1 English 帮助 关于

## 02299: Ultra-QuickSort

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

思路: 归并排序求逆序数

代码

```
# flag=0
while flag==0:
    n=int(input())
    if n==0:
        flag=1
        break
    arr=[]
    for i in range(n):
        arr.append(int(input()))
    cnt=0

    def merge_div(arr):

        if len(arr) == 1:
            return arr
```

```

mid = len(arr) // 2
left = arr[:mid]
right = arr[mid:]
# 开始递归
left = merge_div(left) # 返回排好序的左边
right = merge_div(right) # 返回排好序的右边
return merged(left, right)

def merged(left, right):
    global cnt
    cc = []
    in_left = in_right = 0
    while in_left < len(left) and in_right < len(right):
        if left[in_left] <= right[in_right]:
            cc.append(left[in_left])
            in_left += 1
        else:
            cc.append(right[in_right])
            in_right += 1
            cnt += len(left) - in_left

    cc.extend(left[in_left:])
    cc.extend(right[in_right:])
    return cc
ac=merge_div(arr)
print(cnt)

```

代码运行截图 == (AC代码截图, 至少包含有"Accepted") ==




OpenJudge

题目ID, 标题, 描述

23n2300013289

信箱

账号

CS101 / 题库

题目 排名 状态 提问

#44276366提交状态

查看 提交 统计 提问

状态: Accepted

源代码

```
flag=0
while flag==0:
    n=int(input())
    if n==0:
        flag=1
        break
    arr=[]
    for i in range(n):
        arr.append(int(input()))
    cnt=0

    def merge_div(arr):
        if len(arr) == 1:
            return arr
        mid = len(arr) // 2
        left = arr[:mid]
        right = arr[mid:]
        # 开始递归
        left = merge_div(left) # 返回排好序的左边
        right = merge_div(right) # 返回排好序的右边
        return merged(left, right)

    def merged(left, right):
        global cnt
        cc = []
        in_left = in_right = 0
        while in_left < len(left) and in_right < len(right):
            if left[in_left] <= right[in_right]:
                cc.append(left[in_left])
                in_left += 1
            else:
                cc.append(right[in_right])
                in_right += 1
        cnt += len(left)-in_left
```

基本信息

#: 44276366

题目: 02299

提交人: 23n2300013289

内存: 32348kB

时间: 4345ms

语言: Python3

提交时间: 2024-03-17 19:41:59

## 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。==

- 1.感觉本周的难度还是蛮大的）自己觉得有些东西比如shunting yard算法啥的短时间内自己肯定想不出来，于是有些题目都是看答案理解大致思路以后再自己从头写一遍，基本上会有很多bug，然后慢慢debug慢慢纠正理解，应该也是一种学习吧。
- 2.并且在这个过程中继续加深了对递归的理解，比如波兰表达式和归并排序。
- 3.原来切片也会引起时间复杂度的增加。
- 4.把pre的积木和浇水给写了，都属于思路比较清晰不太涉及算法的题目，其中积木了解了全排列的包。
- 5.总的来说还是学了很多，希望下次遇到类似的思路可以回想起来。