Assignment #7: April 月考

Updated 1557 GMT+8 Apr 3, 2024

2024 spring, Complied by 尹柚鑫 光华管理学院 2100015878

说明：

1）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。

2）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。

3）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：Win11

Python编程环境：jupter notebook

1. 题目

27706: 逐词倒放

http://cs101.openjudge.cn/practice/27706/

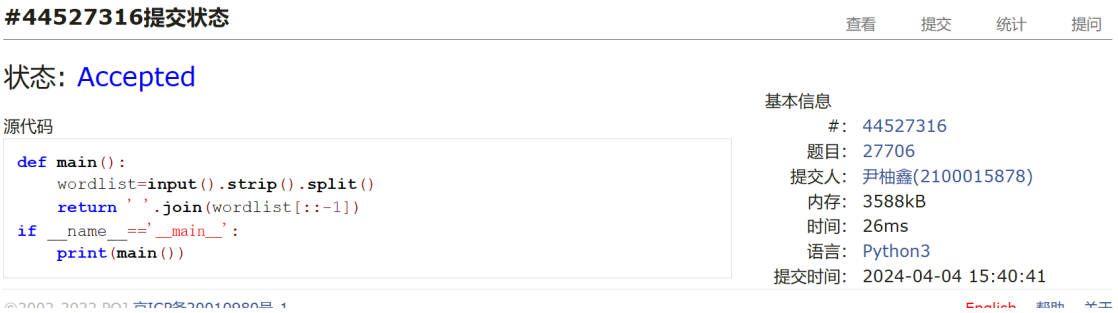
耗时：5mins

思路：输入按照空格分隔，倒序用空格链接后输出

代码

```
#
def main():
    wordlist=input().strip().split()
    return ' '.join(wordlist[::-1])
if __name__=='__main__':
    print(main())
```

代码运行截图 ==（至少包含有"Accepted"）==



27951: 机器翻译

http://cs101.openjudge.cn/practice/27951/

耗时：5mins

思路：比较简单的使用队列的题目

代码

```
#
def main():
    a,b=map(int,input().strip().split())
    queue=[]
    c=[*map(int,input().strip().split())]
    result=0
    for i in c:
        if i in queue:
            continue
        else:
            if len(queue)<a:
                queue.append(i)
                result+=1
            else:
                queue.pop(0)
                queue.append(i)
                result+=1
    return result

if __name__=='__main__':
    print(main())
```

代码运行截图 ==（至少包含有"Accepted"）==

#44527434提交状态 查看 提交 统计 提问

状态: Accepted

源代码

```
def main():
    a,b=map(int,input().strip().split())
    queue=[]
    c=[*map(int,input().strip().split())]
    result=0
    for i in c:
        if i in queue:
            continue
        else:
            if len(queue)<a:
```

基本信息

#: 44527434
题目: 27951
提交人: 尹柚鑫(2100015878)
内存: 3644kB
时间: 26ms
语言: Python3
提交时间: 2024-04-04 15:51:42

27932: Less or Equal

http://cs101.openjudge.cn/practice/27932/

耗时：10mins

思路：排序，之后找第k小的数，同时注意第k小的数是否唯一

代码

```
#
def main():
    a,b=map(int,input().strip().split())
    lis=[1]+[*map(int,input().strip().split())]
    if a<b:
        return -1

    lis.sort()
    if b==a:
        return lis[b]
    else:
        if lis[b]==lis[b+1]:
            return -1
        else:
            return lis[b]

if __name__=='__main__':
    print(main())
```

代码运行截图 ==（AC代码截图，至少包含有"Accepted"）==

#44527569提交状态 查看 提交 统计 提问

状态: Accepted

源代码

```
def main():
    a,b=map(int,input().strip().split())
    lis=[1]+[*map(int,input().strip().split())]
    if a<b:
        return -1

    lis.sort()
    if b==a:
```

基本信息

#: 44527569
题目: 27932
提交人: 尹柚鑫(2100015878)
内存: 10364kB
时间: 47ms
语言: Python3
提交时间: 2024-04-04 16:05:06

27948: FBI树

http://cs101.openjudge.cn/practice/27948/

耗时：30mins

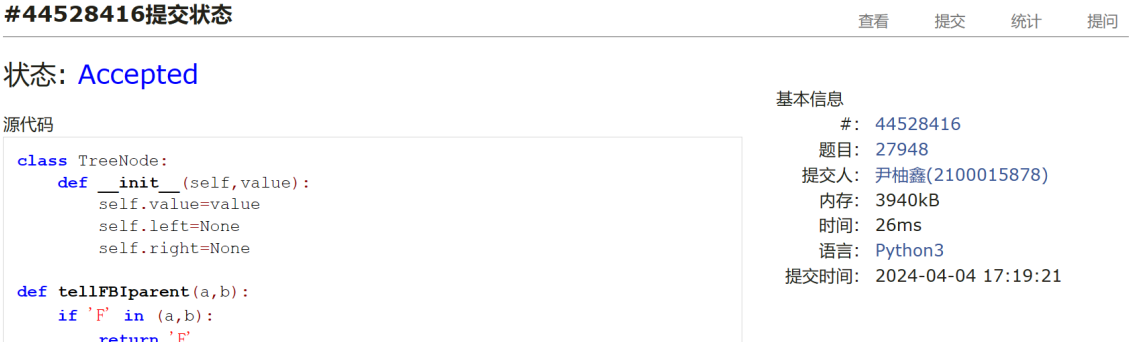
思路：从底层开始，建立列表形式的树，也就是生成层次遍历树，之后把层次遍历树变成node型的树，再做后序遍历

代码

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

def tellFBIparent(a,b):
    if 'F' in (a,b):
        return 'F'
    elif a!=b:
        return 'F'
    else:
        return a

def build_tree_list(lis):
    treelist=[]
    newlis = ['B' if i == 0 else 'I' for i in lis]
    treelist=treelist+newlis
    while len(newlis)>=1:
        tmplis=[]
        while len(newlis)>1:
            x=newlis.pop(0)
            y=newlis.pop(0)
            z=tellFBIparent(x,y)
            tmplis.append(z)
        newlis=tmplis
        treelist=newlis+treelist
    return treelist

def build_tree_list2node(lis):
    if not lis:
        return None

    root = TreeNode(lis[0])
    queue = [root]
    i = 1

    while i < len(lis):
        node = queue.pop(0)
```

```
        left_value = lis[i]
        if left_value is not None:
            node.left = TreeNode(left_value)
            queue.append(node.left)
        i += 1

        if i < len(lis):
            right_value = lis[i]
            if right_value is not None:
                node.right = TreeNode(right_value)
                queue.append(node.right)
        i += 1

    return root

def postorder(root):
    output=[]
    if root:
        output.extend(postorder(root.left))
        output.extend(postorder(root.right))
        output.append(str(root.value))
    return ''.join(output)

def main():
    _=input()
    s=[int(i) for i in list(input().strip())]
    lis=build_tree_list(s)

    root=build_tree_list2node(lis)
    string=postorder(root)

    print(string)

if __name__=='__main__':
    main()
```

代码运行截图 ==（AC代码截图，至少包含有"Accepted"）==



#44528416提交状态

查看 提交 统计 提问

状态: Accepted

源代码

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

def tellFBIparent(a,b):
    if 'F' in (a,b):
        return 'F'
```

基本信息

#: 44528416
题目: 27948
提交人: 尹柚鑫(2100015878)
内存: 3940kB
时间: 26ms
语言: Python3
提交时间: 2024-04-04 17:19:21

27925: 小组队列

http://cs101.openjudge.cn/practice/27925/

耗时：60mins

思路：

用最小堆实现优先队列，使用词典计算小组当前的优先级

最小堆可以实现优先队列，这个题目的小组优先级是会变化的，我们建立两个词典，第一个词典用来记录每一个小组在队列中的个数，另一个词典用来记录在队列中的小组目前的优先级。

优先级是从0开始递增的prio_index

push：如果这个元素所属小组已经在队列中（第一个词典小组对应个数不为0），就读取第二个词典中该小组的优先级，然后添加到最小堆中；如果这个元素所属小组不在队列中，就让这个元素的优先级为prio_index，并且记录在第二个词典中，prio_index+=1

pop：最小堆弹出最小的，第一个词典小组对应的个数-1

代码

```
#
from collections import defaultdict
import heapq
class PriorityQueue:
    def __init__(self):
        self._queue = []  # 人的队列
        self._index = 0
        self._num_group_count=defaultdict(int)
        self._pri_group_index={}
        self._groupindex = 0


    def calpriority(self, index_group):
        if self._num_group_count[index_group]!=0:
            prio_groupindex = self._pri_group_index[index_group]
            return prio_groupindex
        else:
            #self._num_group_count[index_group]+=1
            prio_groupindex = self._groupindex
            self._pri_group_index[index_group]=prio_groupindex
            self._groupindex += 1
            return prio_groupindex

    def push(self, item, index_group):
        priority = self.calpriority(index_group)

        heapq.heappush(self._queue, (priority, self._index, item))

        self._num_group_count[index_group]+=1
        #self._priority_groupindex.append(priority)

        self._index += 1
```

```
    def pop(self):
        item=heapq.heappop(self._queue)[-1]

        index_group=groupdict[item]

        self._num_group_count[index_group]-=1

        return item


groupnums=int(input())
groupdict={}
for i in range(groupnums):
    teplist=[*map(int,input().strip().split())]
    for j in teplist:
        groupdict[j]=i
pq = PriorityQueue()
while True:
    a,*b=input().strip().split()
    if a=='ENQUEUE':
        b=int(b[0])
        c=groupdict[b]
        pq.push(b,c)
    elif a=='DEQUEUE':
        print(pq.pop())
    elif a=='STOP':
        break
```

代码运行截图 ==（AC代码截图，至少包含有"Accepted"）==

#44537837提交状态

查看 提交 统计 提问

状态: Accepted

源代码

```
from collections import defaultdict
import heapq
class PriorityQueue:
    def __init__(self):
        self._queue = []  # 人的队列
        self._index = 0
        self._num_group_count=defaultdict(int)
        self._pri_group_index={}
        self._groupindex = 0


    def calpriority(self, index_group):
```

基本信息

#: 44537837
题目: 27925
提交人: 尹柚鑫(2100015878)
内存: 4968kB
时间: 130ms
语言: Python3
提交时间: 2024-04-05 17:42:30

27928: 遍历树

http://cs101.openjudge.cn/practice/27928/

耗时：20mins

思路：和之前一道找树的高度的题目很像，需要建立找爸爸和找儿子的数据结构，这里用的是词典。

第一个词典find_father记录所有子节点的父亲，根节点不在这个词典中
第二个词典find_son记录所有爸爸的儿子，叶节点不在这个词典中，儿子从小到大排序排序

通过第一个词典找到根节点，然后对根节点做递归

如果节点没有儿子，就输出这个节点

如果节点有儿子，用一个指示数ind=0表示父节点还未输出，ind=1表示父节点已经输出，遍历儿子，如果父节点小于儿子且ind=0，就输出父节点，递归儿子；如果父节点大于儿子，就直接递归儿子。递归完所有儿子之后，如果ind=0，表明父节点还未输出，就输出父节点

代码

```
#
n=int(input().strip())
find_father={}
find_son={}

for _ in range(n):
    a,*b=map(int,input().strip().split())
    if b:
        find_son[a]=b
    for i in b:
        find_father[i]=a
for i in find_son:
    if i not in find_father:
        root_father=i
        break
for i in find_son:
    find_son[i].sort()

result=[]
def bianlitree(node):
    if node not in find_son:
        result.append(node)
        return
    ind=0
    for son in find_son[node]:
        if node<son and ind==0:
            result.append(node)
            ind=1
        bianlitree(son)
    if ind==0:
        result.append(node)
bianlitree(root_father)
for i in result:
    print(i)
```

代码运行截图 ==（AC代码截图，至少包含有"Accepted"）==

状态：Accepted

源代码

```
n=int(input().strip())
find_father={}
find_son={}

for _ in range(n):
    a,*b=map(int,input().strip().split())
    if b:
        find_son[a]=b
    for i in b:
        find_father[i]=a
```

基本信息

#：44538544
题目：27928
提交人：尹柚鑫(2100015878)
内存：3944kB
时间：27ms
语言：Python3
提交时间：2024-04-05 18:55:50

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。==

这次作业前三题都比较简单

第四题FBI树，题不麻烦，建树的代码比较长，同时按照我的做法，有一步需要把层次遍历列表型的树转化为node型的树，这个函数第一次写，略生疏

第五题小组队列，优先队列加上一点小变化，优先队列可以用heapq实现，主要在于写函数得到每个输入的优先级是多少。写起来很生疏，多练

第六题，不一定需要建树，其实最基础的树只是一种指示了节点的儿子是谁的数据结构，也可以用两个列表（词典）表示出来，第一个是找爸爸列表（词典），第二个是找儿子列表（词典），作业中之前出现的找树的高度的题目，也用到了这种想法，这一题同样可以这样做，难度不大

1.

在这个例子中，`lis` 变量会被赋予 `None` 值。这是因为 `extend()` 方法在原地修改列表，而不返回新的列表。因此，`lis` 变量实际上指向的是 `None`。

```
pythonCopy codelis = [1].extend([2, 3])
print(lis)  # 输出：None
```

如果你想要 `lis` 成为扩展后的列表，应该分两步完成：

```
pythonCopy codelis = [1]
lis.extend([2, 3])
print(lis)  # 输出：[1, 2, 3]
```

这样，`lis` 将包含扩展后的列表 `[1, 2, 3]`。

这个会返回None

和下面这一段比较像：

lis.sort()同样返回的是None

2. 字符串的连接join只能连接str，不能连接int

3.实现从层次遍历的树生成node树的代码：

def build_tree_list2node(lis):
    if not lis:
        return None

```
root = TreeNode(lis[0])
queue = [root]
i = 1

while i < len(lis):
    node = queue.pop(0)

    left_value = lis[i]
    if left_value is not None:
        node.left = TreeNode(left_value)
        queue.append(node.left)
    i += 1

    if i < len(lis):
        right_value = lis[i]
        if right_value is not None:
            node.right = TreeNode(right_value)
            queue.append(node.right)
    i += 1

return root
```

4.heapq用法：

`heapq` 模块提供了一系列函数来操作堆，主要包括以下几个常用函数：

1. `heapq.heappush(heap, item)`：将 `item` 元素添加到 `heap` 中，保持堆的不变性。
2. `heapq.heappop(heap)`：弹出并返回 `heap` 中的最小元素，保持堆的不变性。
3. `heapq.heapify(x)`：将列表 `x` 转换为堆，将列表中的元素进行堆排序。
4. `heapq.heapreplace(heap, item)`：弹出并返回 `heap` 中的最小元素，然后将 `item` 添加到 `heap` 中。相当于先执行 `heappop()`，然后再执行 `heappush()`，但效率更高。
5. `heapq.nlargest(n, iterable)`：返回 `iterable` 中最大的 `n` 个元素，以列表形式返回。此函数不会改变 `iterable`。
6. `heapq.nsmallest(n, iterable)`：返回 `iterable` 中最小的 `n` 个元素，以列表形式返回。此函数不会改变 `iterable`。

以下是这些函数的简单示例用法：

```
pythonCopy codeimport heapq

# 创建一个空堆
heap = []

# 向堆中添加元素
heapq.heappush(heap, 4)
heapq.heappush(heap, 1)
heapq.heappush(heap, 7)
```

```
print(heap)  # 输出: [1, 4, 7]

# 弹出并返回堆中的最小元素
print(heapq.heappop(heap))  # 输出: 1

# 将列表转换为堆
x = [3, 2, 1, 5, 6]
heapq.heapify(x)
print(x)  # 输出: [1, 2, 3, 5, 6]

# 弹出并返回堆中的最小元素，并将新元素添加到堆中
print(heapq.heapreplace(heap, 2))  # 输出: 2

# 返回列表中最大的3个元素
print(heapq.nlargest(3, x))  # 输出: [6, 5, 3]

# 返回列表中最小的2个元素
print(heapq.nsmallest(2, x))  # 输出: [1, 2]
```

这些函数在操作堆时非常有用，可以方便地进行堆的插入、弹出和查找等操作。

5.下面是一个通用的用堆实现优先队列的代码：

以下是一个示例演示如何使用 `heapq` 模块实现优先队列：

```
pythonCopy codeimport heapq

class PriorityQueue:
    def __init__(self):
        self._queue = []
        self._index = 0

    def push(self, item, priority):
        heapq.heappush(self._queue, (priority, self._index, item))
        self._index += 1

    def pop(self):
        return heapq.heappop(self._queue)[-1]

# 示例用法
pq = PriorityQueue()
pq.push('task1', 3)
pq.push('task2', 1)
pq.push('task3', 2)

print(pq.pop())  # 输出：task2
print(pq.pop())  # 输出：task3
print(pq.pop())  # 输出：task1
```