Assignment #6: "树"算：Huffman,BinHeap,BST,AVL,DisjointSet

Updated 2214 GMT+8 March 24, 2024

2024 spring, Complied by 胡登科、生科

说明：

1）这次作业内容不简单，耗时长的话直接参考题解。

2）请把每个题目解题思路（可选），源码Py

thon, 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：macOS Ventura 13.4.1 (c)

Python编程环境：Spyder IDE 5.2.2, PyCharm 2023.1.4 (Professional Edition)

C/C++编程环境：Mac terminal vi (version 9.0.1424), g++/gcc (Apple clang version 14.0.3, clang-1403.0.22.14.1)

1. 题目

22275: 二叉搜索树的遍历

http://cs101.openjudge.cn/practice/22275/

思路：本题思路比较简单，通过merge sort的思想进行递归建树，不需要建立节点树或者链表树，在每一次递归中，只需要找到root然后输出root就行。本题为比较考察递归的算法，熟练掌握递归即可理解。其实如果要求输出中序表达式，就需要建树了。

代码

```
# # 二叉搜索树的遍历
# merge sort 解决问题
def post_order(pre_str):
```

```
    if not pre_str:
        return []
    root = pre_str[0]
    pre_str_left = [x for x in pre_str if x < pre_str[0]]
    pre_str_right = [x for x in pre_str if x > pre_str[0]]
    return post_order(pre_str_left) + post_order(pre_str_right) + [root]


n = int(input())
pre_order = list(map(int, input().split()))
print(' '.join(map(str, post_order(pre_order))))
```

代码运行截图 （至少包含有"Accepted"）

状态：Accepted

源代码

```
# 二叉搜索树的遍历
# merge sort 解决问题
def post_order(pre_str):
    if not pre_str:
        return []
    root = pre_str[0]
    pre_str_left = [x for x in pre_str if x < pre_str[0]]
    pre_str_right = [x for x in pre_str if x > pre_str[0]]
    return post_order(pre_str_left) + post_order(pre_str_right) + [root]


n = int(input())
pre_order = list(map(int, input().split()))
print(' '.join(map(str, post_order(pre_order))))
```

05455: 二叉搜索树的层次遍历

http://cs101.openjudge.cn/practice/05455/

思路：

这段代码实现了二叉搜索树的构建和层次遍历输出。

首先定义了一个二叉树节点类`TreeNode`，包含一个值`val`和两个子节点`left`和`right`。

然后定义了一个插入操作函数`insert`，用于向二叉搜索树中插入一个新的节点。如果根节点为空，则直接创建一个新的节点作为根节点；如果插入的值小于根节点的值，则将新节点插入到左子树中；如果插入的值大于根节点的值，则将新节点插入到右子树中。最后返回根节点。

接着定义了一个层次遍历函数`level_order_traversal`，用于对二叉搜索树进行层次遍历。如果根节点为空，则返回空列表；否则使用一个队列存储待遍历的节点，从根节点开始。在每一层遍历过程中，将当前层的节点值添加到结果列表中，并将其左右子节点加入队列。最后返回结果列表。

在主函数中，首先读取输入的数字序列，然后逐个插入到二叉搜索树中。接着调用层次遍历函数，将结果存储在`result`变量中。最后将每一层的节点值用空格连接起来，并输出结果。

没怎么想明白最后该如何层次遍历，想过几种办法，储存lst，loci，但觉得写递归又难办，看了AI写的，觉得还行。

代码

```
#
class TreeNode:
    def __init__(self, val=0):
        self.val = val
        self.left = None
        self.right = None


def insert(root, val):
    if not root:
        return TreeNode(val)
    if val < root.val:
        root.left = insert(root.left, val)
    elif val > root.val:
        root.right = insert(root.right, val)
    return root


def level_order_traversal(root):
    if not root:
        return []
    queue = [root]
    res = []
    while queue:
        level = []
        for i in range(len(queue)):
            node = queue.pop(0)
            level.append(node.val)
            if node.left:
                queue.append(node.left)
            if node.right:
                queue.append(node.right)
        res.append(level)
    return res


if __name__ == "__main__":
    nums = list(map(int, input().split()))
    root = None
    for num in nums:
        root = insert(root, num)
    result = level_order_traversal(root)
    output = ""
    for level in result:
        output += " ".join(map(str, level)) + " "
    print(output.strip())
```

代码运行截图 （至少包含有"Accepted"）

状态：Accepted

源代码

```
class TreeNode:
    def __init__(self, val=0):
        self.val = val
        self.left = None
        self.right = None


def insert(root, val):
    if not root:
        return TreeNode(val)
    if val < root.val:
        root.left = insert(root.left, val)
    elif val > root.val:
        root.right = insert(root.right, val)
    return root


def level_order_traversal(root):
    if not root:
        return []
    queue = [root]
    res = []
    while queue:
        level = []
        for i in range(len(queue)):
            node = queue.pop(0)
            level.append(node.val)
            if node.left:
                queue.append(node.left)
            if node.right:
                queue.append(node.right)
        res.append(level)
    return res


if __name__ == "__main__":
    nums = list(map(int, input().split()))
    root = None
    for num in nums:
        root = insert(root, num)
    result = level_order_traversal(root)
    output = ""
    for level in result:
        output += " ".join(map(str, level)) + " "
    print(output.strip())
```

04078: 实现堆结构

http://cs101.openjudge.cn/practice/04078/

练习自己写个BinHeap。当然机考时候，如果遇到这样题目，直接import heapq。手搓栈、队列、堆、AVL等，考试前需要搓个遍。

思路：

按照课件思路打了一遍，思路明白，但是手不明白

代码

```
#
# 实现堆的结构
class BinHeap:
    def __init__(self):
        self.heapList = [0]
        self.currentSize = 0

    def percUp(self, i):
        while i // 2 > 0:
            if self.heapList[i] < self.heapList[i // 2]:
                tmp = self.heapList[i // 2]
                self.heapList[i // 2] = self.heapList[i]
                self.heapList[i] = tmp
            i = i // 2

    def insert(self, k):
        self.heapList.append(k)
        self.currentSize = self.currentSize + 1
        self.percUp(self.currentSize)

    def percDown(self, i):
        while (i * 2) <= self.currentSize:
            mc = self.minChild(i)
            if self.heapList[i] > self.heapList[mc]:
                tmp = self.heapList[i]
                self.heapList[i] = self.heapList[mc]
                self.heapList[mc] = tmp
            i = mc

    def minChild(self, i):
        if i * 2 + 1 > self.currentSize:
            return i * 2
        else:
            if self.heapList[i * 2] < self.heapList[i * 2 + 1]:
                return i * 2
            else:
                return i * 2 + 1

    def delMin(self):
        retval = self.heapList[1]
        self.heapList[1] = self.heapList[self.currentSize]
        self.currentSize = self.currentSize - 1
```

```
        self.heapList.pop()
        self.percDown(1)
        return retval

    def buildHeap(self, alist):
        i = len(alist) // 2
        self.currentSize = len(alist)
        self.heapList = [0] + alist[:]
        while (i > 0):
            # print(f'i = {i}, {self.heapList}')
            self.percDown(i)
            i = i - 1
        # print(f'i = {i}, {self.heapList}')


n = int(input().strip())
bh = BinHeap()
for _ in range(n):
    inp = input().strip()
    if inp[0] == '1':
        bh.insert(int(inp.split()[1]))
    else:
        print(bh.delMin())
```

代码运行截图 （AC代码截图，至少包含有"Accepted"）

状态: Accepted

原代码

```
# 实现堆的结构
class BinHeap:
    def __init__(self):
        self.heapList = [0]
        self.currentSize = 0

    def percUp(self, i):
        while i // 2 > 0:
            if self.heapList[i] < self.heapList[i // 2]:
                tmp = self.heapList[i // 2]
                self.heapList[i // 2] = self.heapList[i]
                self.heapList[i] = tmp
            i = i // 2

    def insert(self, k):
        self.heapList.append(k)
        self.currentSize = self.currentSize + 1
        self.percUp(self.currentSize)

    def percDown(self, i):
        while (i * 2) <= self.currentSize:
            mc = self.minChild(i)
            if self.heapList[i] > self.heapList[mc]:
                tmp = self.heapList[i]
                self.heapList[i] = self.heapList[mc]
                self.heapList[mc] = tmp
            i = mc

    def minChild(self, i):
        if i * 2 + 1 > self.currentSize:
            return i * 2
        else:
            if self.heapList[i * 2] < self.heapList[i * 2 + 1]:
                return i * 2
            else:
```

22161: 哈夫曼编码树

http://cs101.openjudge.cn/practice/22161/

思路：按照课件上摸索了一遍，思路明白。

代码

```
#
import heapq
```

```python
class Node:
    def __init__(self, weight, char=None):
        self.weight = weight
        self.char = char
        self.left = None
        self.right = None

    def __lt__(self, other):
        if self.weight == other.weight:
            return self.char < other.char
        return self.weight < other.weight

def build_huffman_tree(characters):
    heap = []
    for char, weight in characters.items():
        heapq.heappush(heap, Node(weight, char))

    while len(heap) > 1:
        left = heapq.heappop(heap)
        right = heapq.heappop(heap)
        #merged = Node(left.weight + right.weight) #note: 合并后，char 字段默认值是空
        merged = Node(left.weight + right.weight, min(left.char, right.char))
        merged.left = left
        merged.right = right
        heapq.heappush(heap, merged)

    return heap[0]

def encode_huffman_tree(root):
    codes = {}

    def traverse(node, code):
        #if node.char:
        if node.left is None and node.right is None:
            codes[node.char] = code
        else:
            traverse(node.left, code + '0')
            traverse(node.right, code + '1')

    traverse(root, '')
    return codes

def huffman_encoding(codes, string):
    encoded = ''
    for char in string:
        encoded += codes[char]
    return encoded

def huffman_decoding(root, encoded_string):
    decoded = ''
    node = root
    for bit in encoded_string:
        if bit == '0':
            node = node.left
        else:
            node = node.right
```

```
        #if node.char:
        if node.left is None and node.right is None:
            decoded += node.char
            node = root
    return decoded

# 读取输入
n = int(input())
characters = {}
for _ in range(n):
    char, weight = input().split()
    characters[char] = int(weight)

#string = input().strip()
#encoded_string = input().strip()

# 构建哈夫曼编码树
huffman_tree = build_huffman_tree(characters)

# 编码和解码
codes = encode_huffman_tree(huffman_tree)

strings = []
while True:
    try:
        line = input()
        strings.append(line)

    except EOFError:
        break

results = []
#print(strings)
for string in strings:
    if string[0] in ('0','1'):
        results.append(huffman_decoding(huffman_tree, string))
    else:
        results.append(huffman_encoding(codes, string))

for result in results:
    print(result)
```

代码运行截图 （AC代码截图，至少包含有"Accepted"）

状态：Accepted

源代码

```
import heapq

class Node:
    def __init__(self, weight, char=None):
        self.weight = weight
        self.char = char
        self.left = None
        self.right = None

    def __lt__(self, other):
        if self.weight == other.weight:
            return self.char < other.char
        return self.weight < other.weight

def build_huffman_tree(characters):
    heap = []
    for char, weight in characters.items():
        heapq.heappush(heap, Node(weight, char))

    while len(heap) > 1:
        left = heapq.heappop(heap)
        right = heapq.heappop(heap)
        #merged = Node(left.weight + right.weight) #note: 合并后, char 字
        merged = Node(left.weight + right.weight, min(left.char, right.c
        merged.left = left
        merged.right = right
        heapq.heappush(heap, merged)

    return heap[0]

def encode_huffman_tree(root):
    codes = {}

    def traverse(node, code):
        #if node.char:
```

晴问9.5: 平衡二叉树的建立

https://sunnywhy.com/sfbj/9/5/359

思路：

按照课件上打了一遍，还是晕晕的。

代码

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

```
        self.height = 1

class AVL:
    def __init__(self):
        self.root = None

    def insert(self, value):
        if not self.root:
            self.root = Node(value)
        else:
            self.root = self._insert(value, self.root)

    def _insert(self, value, node):
        if not node:
            return Node(value)
        elif value < node.value:
            node.left = self._insert(value, node.left)
        else:
            node.right = self._insert(value, node.right)

        node.height = 1 + max(self._get_height(node.left),
self._get_height(node.right))

        balance = self._get_balance(node)

        if balance > 1:
            if value < node.left.value: # 树形是 LL
                return self._rotate_right(node)
            else:   # 树形是 LR
                node.left = self._rotate_left(node.left)
                return self._rotate_right(node)

        if balance < -1:
            if value > node.right.value:    # 树形是 RR
                return self._rotate_left(node)
            else:   # 树形是 RL
                node.right = self._rotate_right(node.right)
                return self._rotate_left(node)

        return node

    def _get_height(self, node):
        if not node:
            return 0
        return node.height

    def _get_balance(self, node):
        if not node:
            return 0
        return self._get_height(node.left) - self._get_height(node.right)

    def _rotate_left(self, z):
        y = z.right
        T2 = y.left
        y.left = z
        z.right = T2
```

```
        z.height = 1 + max(self._get_height(z.left), self._get_height(z.right))
        y.height = 1 + max(self._get_height(y.left), self._get_height(y.right))
        return y

    def _rotate_right(self, y):
        x = y.left
        T2 = x.right
        x.right = y
        y.left = T2
        y.height = 1 + max(self._get_height(y.left), self._get_height(y.right))
        x.height = 1 + max(self._get_height(x.left), self._get_height(x.right))
        return x

    def preorder(self):
        return self._preorder(self.root)

    def _preorder(self, node):
        if not node:
            return []
        return [node.value] + self._preorder(node.left) +
self._preorder(node.right)

n = int(input().strip())
sequence = list(map(int, input().strip().split()))

avl = AVL()
for value in sequence:
    avl.insert(value)

print(' '.join(map(str, avl.preorder())))
```

代码运行截图 （AC代码截图，至少包含有"Accepted"）

测试输入 | 提交结果 | 历史提交

完美通过 查看题解

100% 数据通过测试

运行时长: 0 ms

02524: 宗教信仰

http://cs101.openjudge.cn/practice/02524/

思路：

看了答案还是不是很理解。

代码

```
#
# 宗教信仰
def init_set(n):
    return list(range(n))


def get_father(x, father):
    if father[x] != x:
        father[x] = get_father(father[x], father)
    return father[x]


def join(x, y, father):
    fx = get_father(x, father)
    fy = get_father(y, father)
    if fx == fy:
        return
    father[fx] = fy


def is_same(x, y, father):
    return get_father(x, father) == get_father(y, father)


def main():
    case_num = 0
    while True:
        n, m = map(int, input().split())
        if n == 0 and m == 0:
            break
        count = 0
        father = init_set(n)
        for _ in range(m):
            s1, s2 = map(int, input().split())
            join(s1 - 1, s2 - 1, father)
        for i in range(n):
            if father[i] == i:
                count += 1
        case_num += 1
        print(f"Case {case_num}: {count}")


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

代码运行截图 （AC代码截图，至少包含有"Accepted"）

状态：Accepted

源代码

```
# 宗教信仰
def init_set(n):
    return list(range(n))


def get_father(x, father):
    if father[x] != x:
        father[x] = get_father(father[x], father)
    return father[x]


def join(x, y, father):
    fx = get_father(x, father)
    fy = get_father(y, father)
    if fx == fy:
        return
    father[fx] = fy


def is_same(x, y, father):
    return get_father(x, father) == get_father(y, father)


def main():
    case_num = 0
    while True:
        n, m = map(int, input().split())
        if n == 0 and m == 0:
            break
        count = 0
        father = init_set(n)
        for _ in range(m):
            s1, s2 = map(int, input().split())
            join(s1 - 1, s2 - 1, father)
        for i in range(n):
```

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。

好难，就前两个体做的明白，后面的堆和AVL树都感觉不熟练。

期中越来越忙了，日子什么时候是个头呀。