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

Updated 2214 GMT+8 March 24, 2024

2024 spring, Complied by ==同学的姓名、院系==

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：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：递归，遇到比根节点大的元素就递归创建子树

代码

```
def binary_search_tree(l:list):
    if len(l) <= 1:
        return l
    for index in range(1, len(l)):
        if l[index] > l[0]:
            return binary_search_tree(l[1:index]) + binary_search_tree(l[index:]) + [l[0]]
    else:
        return binary_search_tree(l[1:]) + [l[0]]

n = int(input())
print(*binary_search_tree([int(x) for x in input().split()]))
```

代码运行截图 ==（至少包含有"Accepted"）==

05455: 二叉搜索树的层次遍历

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

思路：

实现BST的插入方法后逐元素插入，按深度遍历。

代码

```
class BinarySearchTree:
    depths = {}
    def __init__(self, value, depth=0):
        self.value = value
        self.left_node = None
        self.right_node = None
        self.depth = depth
        BinarySearchTree.depths.setdefault(depth, [])
        BinarySearchTree.depths[depth].append(self.value)

    def insert(self, value):
        if self.value == value:
            return
        if self.value > value:
            if self.left_node is None:
                self.left_node = BinarySearchTree(value, self.depth + 1)
            else:
                self.left_node.insert(value)
        if self.value < value:
            if self.right_node is None:
```

```
                self.right_node = BinarySearchTree(value, self.depth + 1)
            else:
                self.right_node.insert(value)

inputs = [int(x) for x in input().split()]
root = BinarySearchTree(inputs[0])
for i in inputs[1:]:
    root.insert(i)

print(*[' '.join([str(j) for j in sorted(i)]) for i in BinarySearchTree.depths.values()])
```

代码运行截图 ==（至少包含有"Accepted"）==

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：学习了一下最小堆的逻辑，手动实现。

代码

```
heap = []

def heap_swap(i, j):
    heap[i], heap[j] = min(heap[i], heap[j]), max(heap[i], heap[j])

def father(i):
    return (i - 1) // 2

def sons(i):
    return (2 * i + 1, 2 * i + 2)

def insert(value):
    def search_upwards(index):
        if index == 0 or heap[father(index)] <= heap[index]:
            return
        heap_swap(father(index), index)
        search_upwards(father(index))

    heap.append(value)
```

```
    search_upwards(len(heap) - 1)

def pop_smallest():
    def search_downwards(index):
        real_sons = [son for son in sons(index) if son < len(heap)]
        if real_sons == []:
            return
        min_son = real_sons[0]
        for son in real_sons:
            if heap[son] < heap[min_son]:
                min_son = son
        if heap[min_son] >= heap[index]:
            return
        heap_swap(index, min_son)
        search_downwards(min_son)

    heap[0], heap[-1] = heap[-1], heap[0]
    res = heap.pop()
    search_downwards(0)
    return res

n = int(input())
for _ in range(n):
    op = [int(i) for i in input().split()]
    if op[0] == 1:
        insert(op[1])
    elif op[0] == 2:
        print(pop_smallest())
```

代码运行截图 ==（AC代码截图，至少包含有"Accepted"）==

22161: 哈夫曼编码树

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

思路：维护优先队列，建立Huffman树。

代码

```
import heapq

class Char:
    char_dict = {}
```

```
    def __init__(self, weight, char):
        self.weight = weight
        self.char = char
        self.chars = [char]
        self.parent = self
        self.code = ''
        Char.char_dict[char] = self

    def __lt__(self, other):
        if self.weight == other.weight:
            return min(self.chars) < min(other.chars)
        else:
            return self.weight < other.weight

class Node:
    def __init__(self, lchild, rchild):
        self.weight = lchild.weight + rchild.weight
        self.children = (lchild, rchild)
        self.parent = self
        self.code = ''
        self.chars = lchild.chars + rchild.chars
        lchild.parent = self
        rchild.parent = self
        self.children[0].code = '0' + self.children[0].code
        self.children[1].code = '1' + self.children[1].code
        if isinstance(self.children[0], Node):
            self.children[0].code_update('0')
        if isinstance(self.children[1], Node):
            self.children[1].code_update('1')

    def code_update(self, code):
        self.children[0].code = code + self.children[0].code
        self.children[1].code = code + self.children[1].code
        if isinstance(self.children[0], Node):
            self.children[0].code_update(code)
        if isinstance(self.children[1], Node):
            self.children[1].code_update(code)

    def __lt__(self, other):
        if self.weight == other.weight:
            return min(self.chars) < min(other.chars)
        else:
            return self.weight < other.weight

heap = []

n = int(input())
for _ in range(n):
    char, weight = input().split()
    heapq.heappush(heap, (int(weight), Char(int(weight), char)))

while len(heap) > 1:
```

```
    lchild, rchild = heapq.heappop(heap)[1], heapq.heappop(heap)[1]
    if lchild > rchild:
        lchild, rchild = rchild, lchild
    father = Node(lchild, rchild)
    heapq.heappush(heap, (father.weight, father))
root = heap[0][1]

while 1:
    try:
        inp = input()
        if inp.isdigit():
            pointer = root
            res = []
            for digit in inp:
                pointer = pointer.children[int(digit)]
                if isinstance(pointer, Char):
                    res.append(pointer.char)
                    pointer = root
            print(''.join(res))
        else:
            res = []
            for char in inp:
                res.append(Char.char_dict[char].code)
            print(''.join(res))
    except EOFError:
        break
```

代码运行截图 ==（AC代码截图，至少包含有"Accepted"）==

#44496295提交状态

状态: Accepted

源代码

```
#22161:哈夫曼编码树
import heapq

class Char:
    char_dict = {}
    def __init__(self, weight, char):
        self.weight = weight
        self.char = char
        self.chars = [char]
        self.parent = self
        self.code = ''
        Char.char_dict[char] = self

    def __lt__(self, other):
        if self.weight == other.weight:
            return min(self.chars) < min(other.chars)
        else:
            return self.weight < other.weight

class Node:
    def __init__(self, lchild, rchild):
        self.weight = lchild.weight + rchild.weight
        self.children = (lchild, rchild)
        self.parent = self
        self.code = ''
        self.chars = lchild.chars + rchild.chars
        lchild.parent = self
        rchild.parent = self
        self.children[0].code = '0' + self.children[0].code
        self.children[1].code = '1' + self.children[1].code
        if isinstance(self.children[0], Node):
            self.children[0].code_update('0')
        if isinstance(self.children[1], Node):
            self.children[1].code_update('1')

    def code_update(self, code):
        self.children[0].code = code + self.children[0].code
        self.children[1].code = code + self.children[1].code
        if isinstance(self.children[0], Node):
            self.children[0].code_update(code)
        if isinstance(self.children[1], Node):
            self.children[1].code_update(code)
```

晴问9.5: 平衡二叉树的建立

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

思路:

AVL树的实现。

代码

```
class Node:
    def __init__(self, key):
        self.key = key
        self.left = None
        self.right = None
        self.height = 1

class AVL_Tree:
    def insert(self, root, key):
        if not root:
            return Node(key)
        elif key < root.key:
            root.left = self.insert(root.left, key)
        else:
            root.right = self.insert(root.right, key)

        root.height = 1 + max(self.getHeight(root.left), self.getHeight(root.right))

        balance = self.getBalance(root)

        if balance > 1:
            if key < root.left.key:
                return self.rightRotate(root)
            else:
                root.left = self.leftRotate(root.left)
                return self.rightRotate(root)

        if balance < -1:
            if key > root.right.key:
                return self.leftRotate(root)
            else:
                root.right = self.rightRotate(root.right)
                return self.leftRotate(root)

        return root

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

```
        y.left = z
        z.right = T2
        z.height = 1 + max(self.getHeight(z.left), self.getHeight(z.right))
        y.height = 1 + max(self.getHeight(y.left), self.getHeight(y.right))
        return y

    def rightRotate(self, y):
        x = y.left
        T3 = x.right
        x.right = y
        y.left = T3
        y.height = 1 + max(self.getHeight(y.left), self.getHeight(y.right))
        x.height = 1 + max(self.getHeight(x.left), self.getHeight(x.right))
        return x

    def getHeight(self, root):
        if not root:
            return 0
        return root.height

    def getBalance(self, root):
        if not root:
            return 0
        return self.getHeight(root.left) - self.getHeight(root.right)
res = []
# Usage of the AVL_Tree
def preOrder(root):
    if not root:
        return
    res.append(root.key)
    preOrder(root.left)
    preOrder(root.right)

myTree = AVL_Tree()
root = None

n = int(input())
nums = [int(x) for x in input().split()]

for num in nums:
    root = myTree.insert(root, num)

preOrder(root)
print(*res)
```

代码运行截图 ==（AC代码截图，至少包含有"Accepted"）==

完美通过

100% 数据通过测试

运行时长: 0 ms

语言: Python

```python
#9.5: 平衡二叉树的建立
class Node:
    def __init__(self, key):
        self.key = key
        self.left = None
        self.right = None
        self.height = 1

class AVL_Tree:
    def insert(self, root, key):
        if not root:
            return Node(key)
        elif key < root.key:
            root.left = self.insert(root.left, key)
        else:
            root.right = self.insert(root.right, key)

        root.height = 1 + max(self.getHeight(root.left), sel

        balance = self.getBalance(root)

        if balance > 1:
            if key < root.left.key:
                return self.rightRotate(root)
```

02524: 宗教信仰

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

思路:

并查集的实现，包括 Path compression 和 rank 的实现。

代码

```
class Node:
    def __init__(self):
        self.parent = self
        self.rank = 0

class DisjointSet:
    def __init__(self, n):
        self.nodes = [Node() for _ in range(n)]
        self.count = n

    def _find(self, n):
        if n.parent != n:
            n.parent = self._find(n.parent)
        return n.parent

    def find(self, n):
        return self._find(self.nodes[n])

    def union(self, u, v):
        u = self.find(u)
        v = self.find(v)
        if u != v:
            if u.rank < v.rank:
                u.parent = v
            else:
                v.parent = u
                if v.rank == u.rank:
                    u.rank += 1
            self.count -= 1

    def __len__(self):
        return self.count
case = 0
while 1:
    case += 1
    n, m = map(int, input().split())
    if (n, m) == (0, 0):
        break
    ds = DisjointSet(n)
    for _ in range(m):
        i, j = map(int, input().split())
        ds.union(i - 1, j - 1)
    print(f"Case {case}: {len(ds)}")
```

代码运行截图 ==（AC代码截图，至少包含有"Accepted"）==

状态: Accepted

源代码

```
#02524:宗教信仰
class Node:
    def __init__(self):
        self.parent = self
        self.rank = 0

class DisjointSet:
    def __init__(self, n):
        self.nodes = [Node() for _ in range(n)]
        self.count = n

    def _find(self, n):
        if n.parent != n:
            n.parent = self._find(n.parent)
        return n.parent

    def find(self, n):
        return self._find(self.nodes[n])

    def union(self, u, v):
        u = self.find(u)
        v = self.find(v)
        if u != v:
            if u.rank < v.rank:
                u.parent = v
            else:
                v.parent = u
                if v.rank == u.rank:
                    u.rank += 1
            self.count -= 1

    def __len__(self):
        return self.count
case = 0
while 1:
    case += 1
    n, m = map(int, input().split())
    if (n, m) == (0, 0):
        break
    ds = DisjointSet(n)
    for _ in range(m):
        i, j = map(int, input().split())
        ds.union(i - 1, j - 1)
    print(f"Case {case}: {len(ds)}")
```

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。==

通过这几道题目，对上周涉及的数据结构有了较深的理解。AVL树果然还是最难的一部分。