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==

罗瑞哲 生命科学学院

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：Windows 10

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

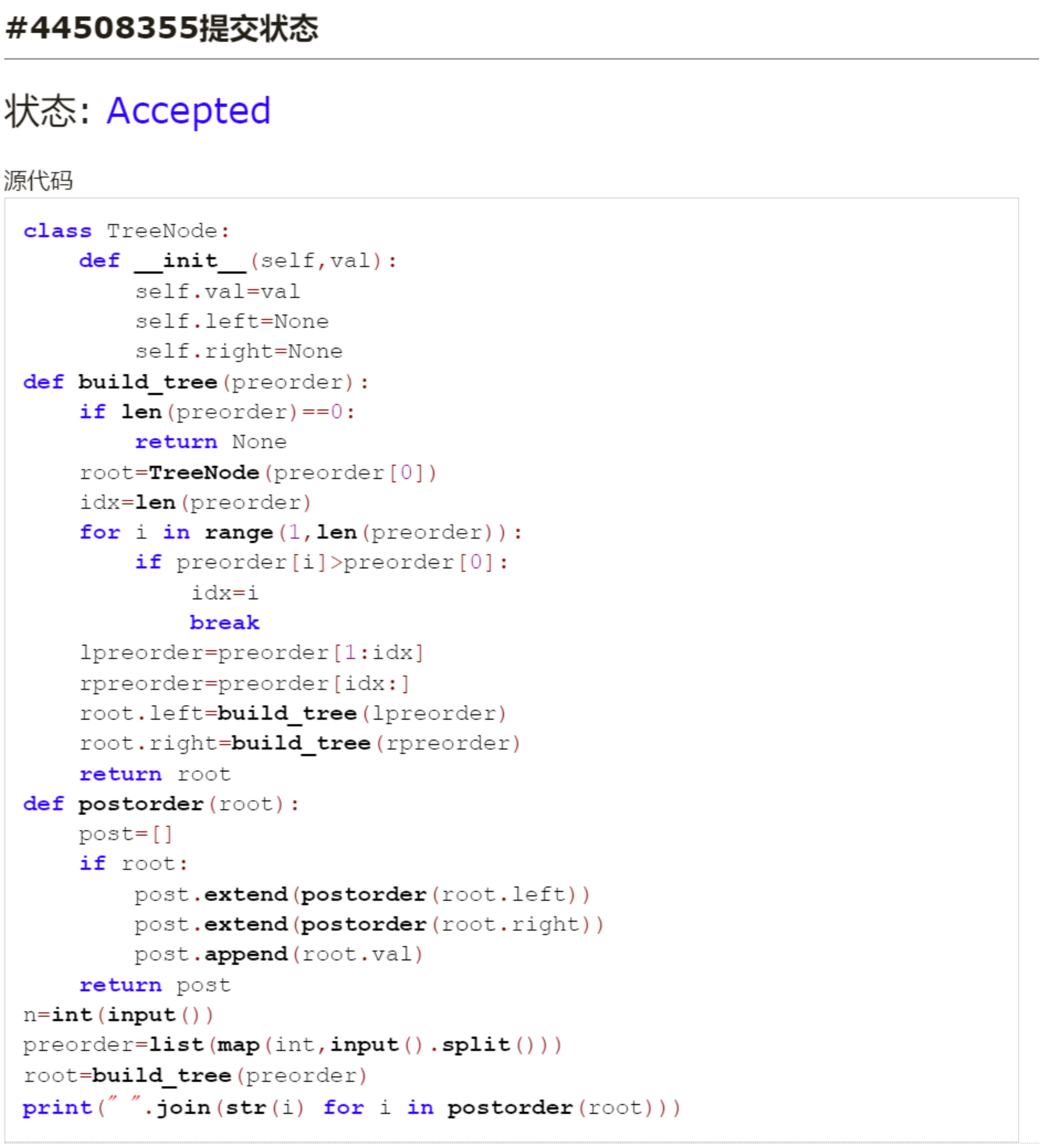
http://cs101.openjudge.cn/practice/22275/

思路：

代码

```python
#
class TreeNode:
    def __init__(self,val):
        self.val=val
        self.left=None
        self.right=None
def build_tree(preorder):
    if len(preorder)==0:
        return None
    root=TreeNode(preorder[0])
    idx=len(preorder)
    for i in range(1,len(preorder)):
        if preorder[i]>preorder[0]:
            idx=i
            break
    lpreorder=preorder[1:idx]
    rpreorder=preorder[idx:]
    root.left=build_tree(lpreorder)
    root.right=build_tree(rpreorder)
    return root
def postorder(root):
    post=[]
    if root:
        post.extend(postorder(root.left))
        post.extend(postorder(root.right))
        post.append(root.val)
    return post
n=int(input())
preorder=list(map(int,input().split()))
root=build_tree(preorder)
print(" ".join(str(i) for i in postorder(root)))
```

代码运行截图 ==（至少包含有"Accepted"）==



#44508355提交状态

状态: Accepted

源代码

```python
class TreeNode:
    def __init__(self,val):
        self.val=val
        self.left=None
        self.right=None
def build_tree(preorder):
    if len(preorder)==0:
        return None
    root=TreeNode(preorder[0])
    idx=len(preorder)
    for i in range(1,len(preorder)):
        if preorder[i]>preorder[0]:
            idx=i
            break
    lpreorder=preorder[1:idx]
    rpreorder=preorder[idx:]
    root.left=build_tree(lpreorder)
    root.right=build_tree(rpreorder)
    return root
def postorder(root):
    post=[]
    if root:
        post.extend(postorder(root.left))
        post.extend(postorder(root.right))
        post.append(root.val)
    return post
n=int(input())
preorder=list(map(int,input().split()))
root=build_tree(preorder)
print(" ".join(str(i) for i in postorder(root)))
```

05455: 二叉搜索树的层次遍历

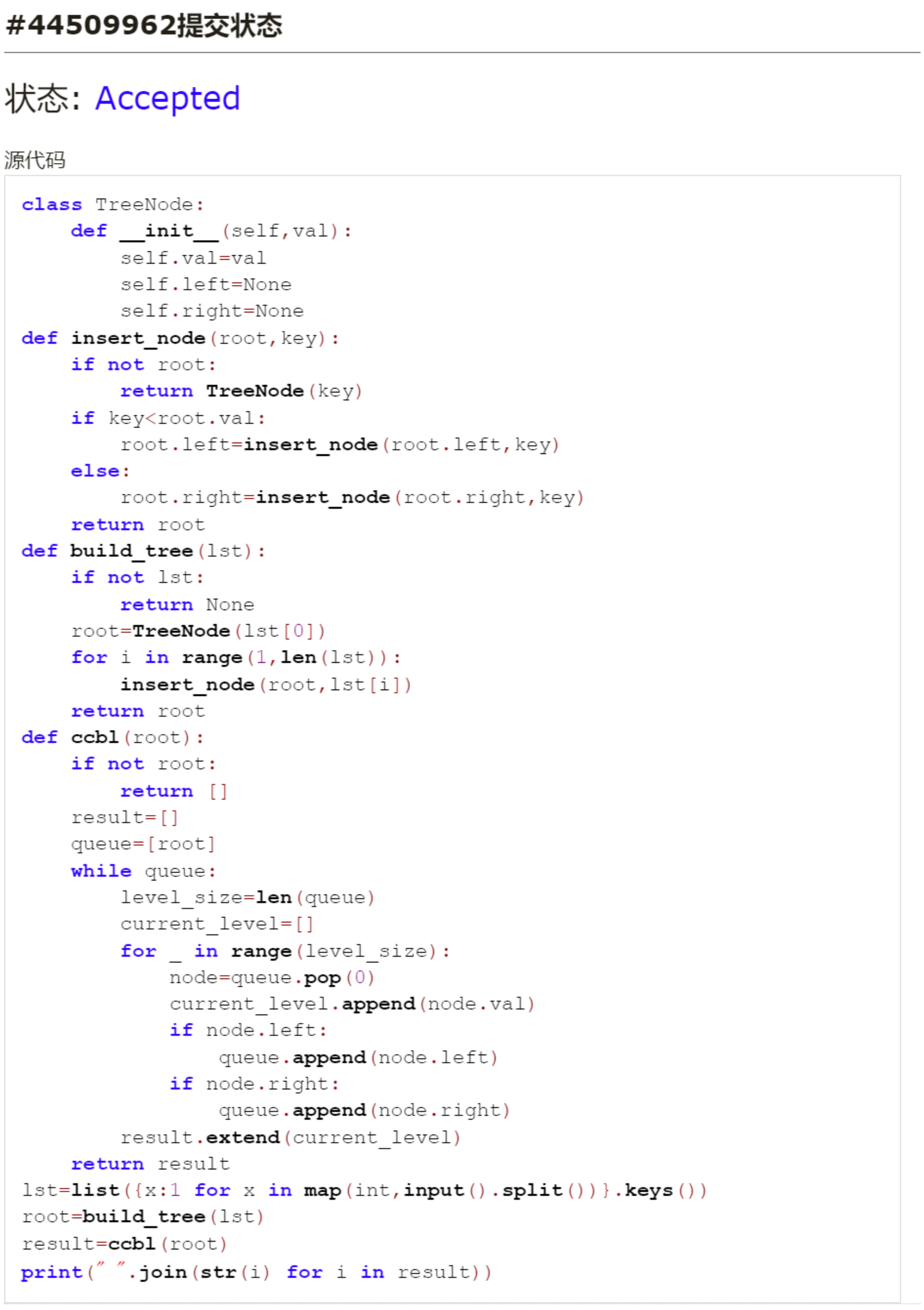
http://cs101.openjudge.cn/practice/05455/

思路：

代码

```python
#
class TreeNode:
    def __init__(self,val):
        self.val=val
        self.left=None
        self.right=None
def insert_node(root,key):
    if not root:
        return TreeNode(key)
    if key<root.val:
        root.left=insert_node(root.left,key)
    else:
        root.right=insert_node(root.right,key)
    return root
def build_tree(lst):
    if not lst:
        return None
    root=TreeNode(lst[0])
    for i in range(1,len(lst)):
        insert_node(root,lst[i])
    return root
def ccbl(root):
    if not root:
        return []
    result=[]
    queue=[root]
    while queue:
        level_size=len(queue)
        current_level=[]
        for _ in range(level_size):
            node=queue.pop(0)
            current_level.append(node.val)
            if node.left:
                queue.append(node.left)
            if node.right:
                queue.append(node.right)
        result.extend(current_level)
    return result
lst=list({x:1 for x in map(int,input().split())}.keys())
root=build_tree(lst)
result=ccbl(root)
print(" ".join(str(i) for i in result))
```

代码运行截图 ==（至少包含有"Accepted"）==



#44509962提交状态

状态: Accepted

源代码

```python
class TreeNode:
    def __init__(self,val):
        self.val=val
        self.left=None
        self.right=None
def insert_node(root,key):
    if not root:
        return TreeNode(key)
    if key<root.val:
        root.left=insert_node(root.left,key)
    else:
        root.right=insert_node(root.right,key)
    return root
def build_tree(lst):
    if not lst:
        return None
    root=TreeNode(lst[0])
    for i in range(1,len(lst)):
        insert_node(root,lst[i])
    return root
def ccbl(root):
    if not root:
        return []
    result=[]
    queue=[root]
    while queue:
        level_size=len(queue)
        current_level=[]
        for _ in range(level_size):
            node=queue.pop(0)
            current_level.append(node.val)
            if node.left:
                queue.append(node.left)
            if node.right:
                queue.append(node.right)
        result.extend(current_level)
    return result
lst=list({x:1 for x in map(int,input().split())}.keys())
root=build_tree(lst)
result=ccbl(root)
print(" ".join(str(i) for i in result))
```

04078: 实现堆结构

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

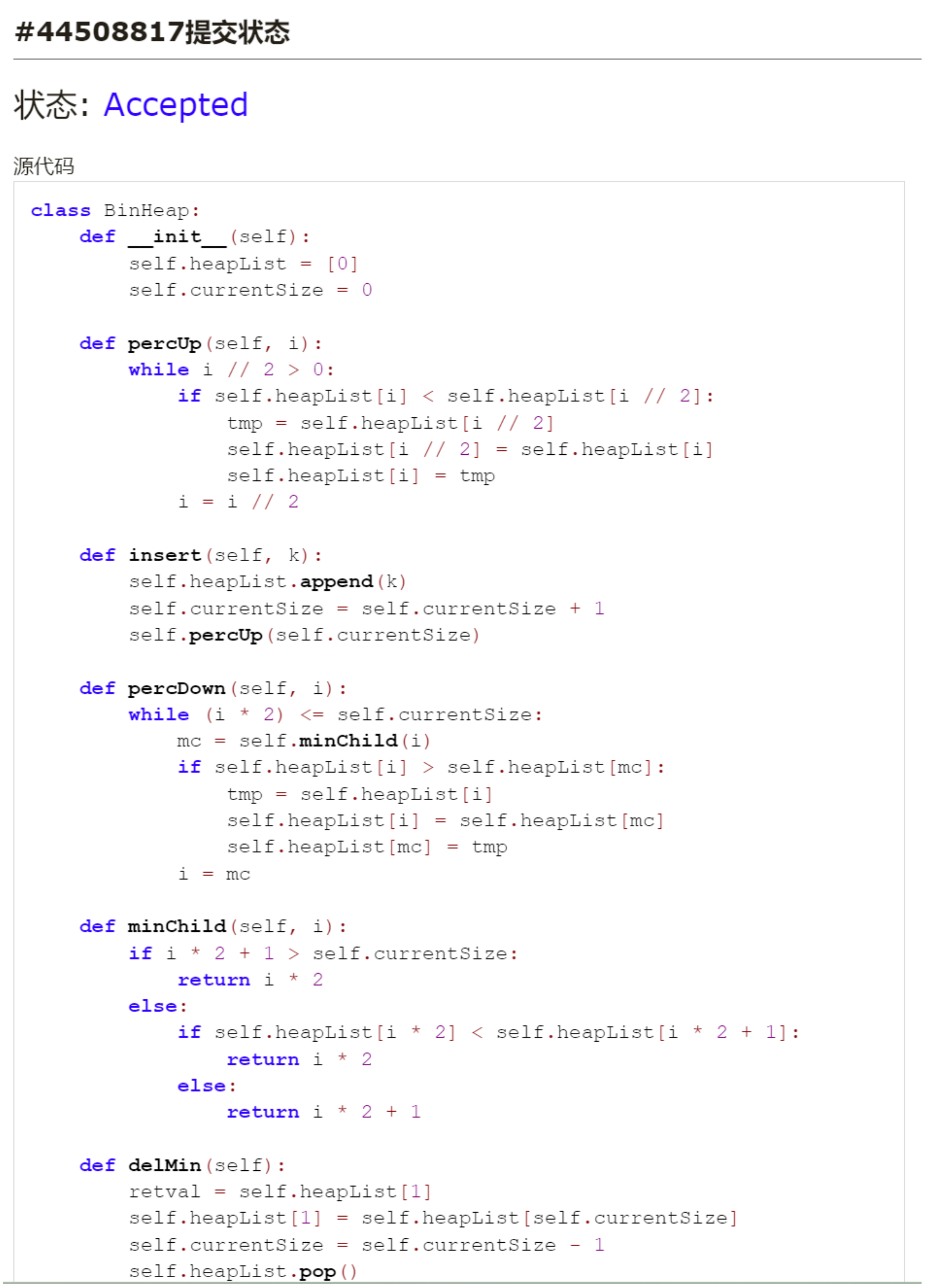
练习自己写个BinHeap。当然机考时候，如果遇到这样题目，直接import heapq。手搓栈、队列、堆、AVL等，考试前需要搓个遍。

思路:

代码

```
#
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):
            self.percDown(i)
            i = i - 1


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"）==



#44508817提交状态

状态: Accepted

源代码

```python
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()
```

22161: 哈夫曼编码树

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

思路:

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, 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.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.left is None and node.right is None:
                decoded += node.char
                node = root
    return decoded

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

huffman_tree = build_huffman_tree(characters)

codes = encode_huffman_tree(huffman_tree)

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

    except EOFError:
        break

results = []

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"）==

#44510016提交状态

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, min(left.char, right.
        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.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
```

晴问9.5: 平衡二叉树的建立

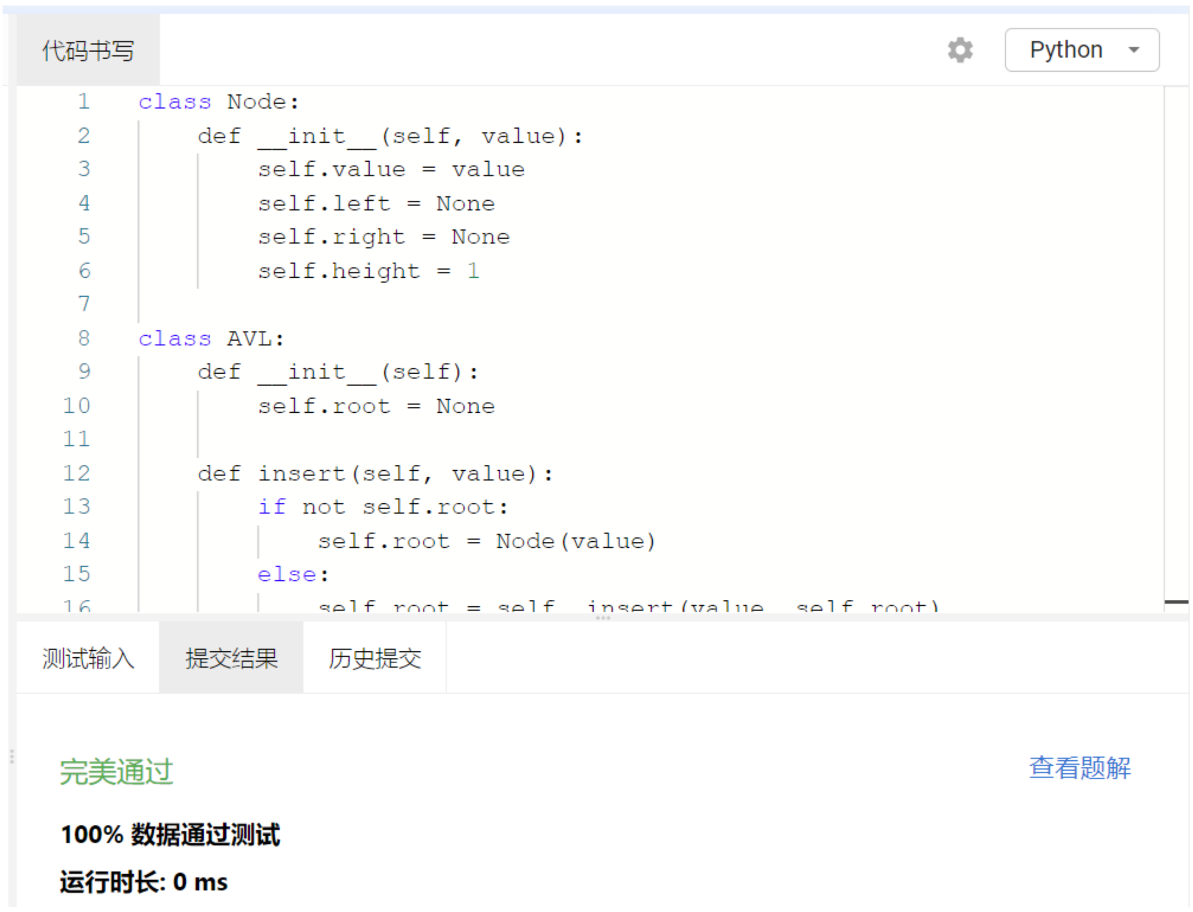
https://sunnywhy.com/sfbj/9/5/359

思路:

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:
                return self._rotate_right(node)
            else:
                node.left = self._rotate_left(node.left)
                return self._rotate_right(node)
        if balance < -1:
            if value > node.right.value:
                return self._rotate_left(node)
            else:
                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, node):
        new_root = node.right
        node.right = new_root.left
        new_root.left = node
        node.height = 1 + max(self._get_height(node.left),
self._get_height(node.right))
        new_root.height = 1 + max(self._get_height(new_root.left),
self._get_height(new_root.right))
        return new_root

    def _rotate_right(self, node):
        new_root = node.left
        node.left = new_root.right
        new_root.right = node
        node.height = 1 + max(self._get_height(node.left),
self._get_height(node.right))
        new_root.height = 1 + max(self._get_height(new_root.left),
self._get_height(new_root.right))
        return new_root

    def pre(self):
        return self._pre(self.root)

    def _pre(self, node):
        if not node:
            return []
        sq = [node.value]
        sq.extend(self._pre(node.left))
        sq.extend(self._pre(node.right))

        return sq

n = int(input())
sq = [int(x) for x in input().split()]
avl = AVL()
for x in sq:
    avl.insert(x)
print(*avl.pre())
```

代码运行截图 ==（AC代码截图，至少包含有"Accepted"）==



```python
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)
```

完美通过

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:

代码

```python
#
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)
```

```python
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}")
```

代码运行截图 ==（AC代码截图，至少包含有"Accepted"）==

#44510117提交状态

状态: Accepted

源代码

```python
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)

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}")
```

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。==

对于一些经典的算法，我需要充分理解并记住它们，还要知道什么时候应用这些算法。感觉这学期的数算明显比我上学期学的计概难度大，我必须要抓紧了。