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

Updated 2214 GMT+8 March 24, 2024

2024 spring, Complied by 刘思瑞 元培学院

**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**

操作系统：Windows 11 22H2 22621.2283

Python编程环境：Visual Studio (1.82.2); python 3.11.3

C/C++编程环境：无

## 1. 题目

### 22275: 二叉搜索树的遍历

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

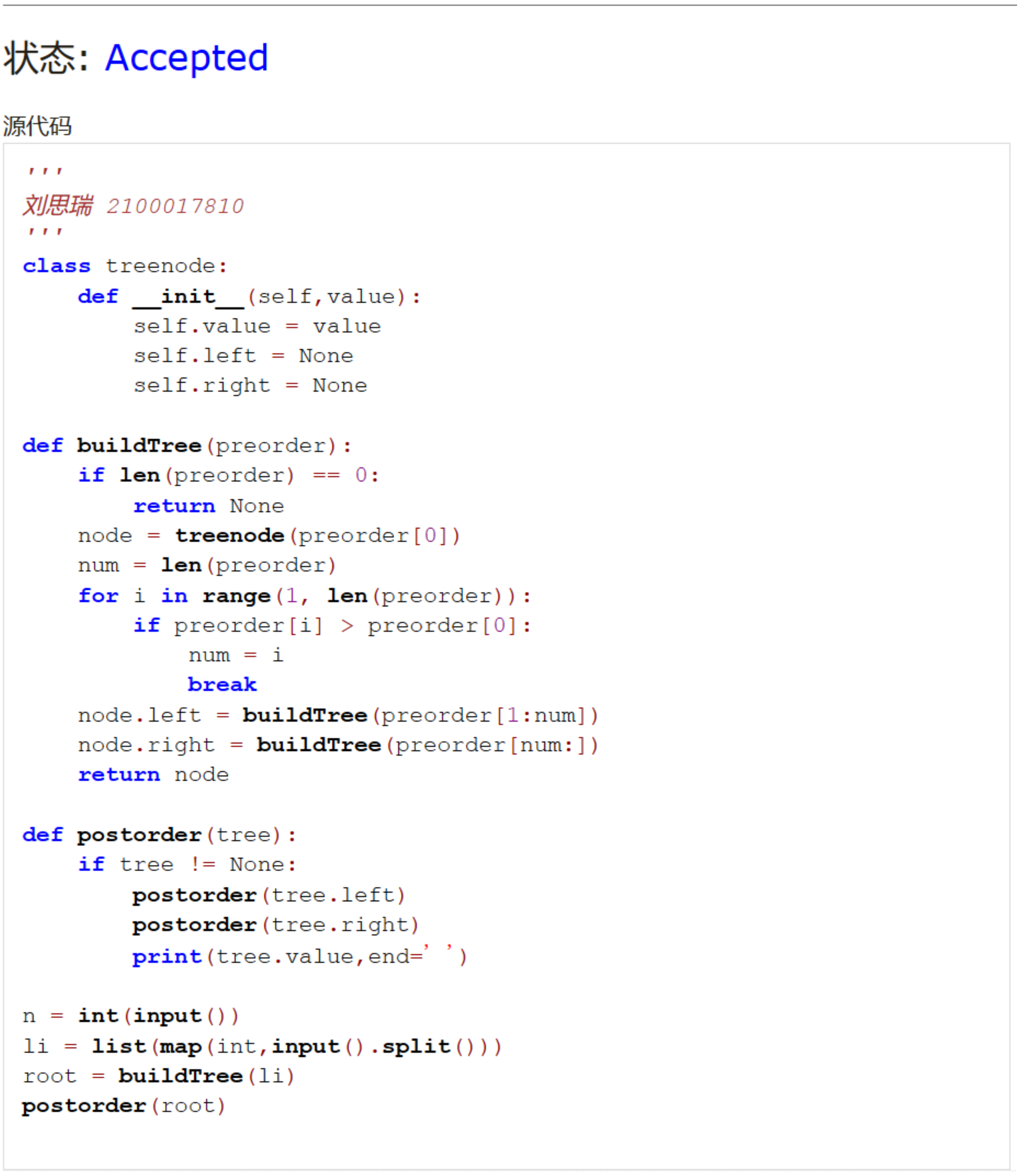
思路：

建树然后递归

代码

```python
'''
刘思瑞 2100017810
'''
class treenode:
    def __init__(self,value):
        self.value = value
        self.left = None
        self.right = None

def buildTree(preorder):
    if len(preorder) == 0:
        return None
    node = treenode(preorder[0])
    num = len(preorder)
    for i in range(1, len(preorder)):
        if preorder[i] > preorder[0]:
            num = i
            break
    node.left = buildTree(preorder[1:num])
    node.right = buildTree(preorder[num:])
    return node

def postorder(tree):
    if tree != None:
        postorder(tree.left)
        postorder(tree.right)
        print(tree.value,end=' ')

n = int(input())
li = list(map(int,input().split()))
root = buildTree(li)
postorder(root)
```

代码运行截图



状态: Accepted

源代码

```python
'''
刘思瑞 2100017810
'''
class treenode:
    def __init__(self,value):
        self.value = value
        self.left = None
        self.right = None

def buildTree(preorder):
    if len(preorder) == 0:
        return None
    node = treenode(preorder[0])
    num = len(preorder)
    for i in range(1, len(preorder)):
        if preorder[i] > preorder[0]:
            num = i
            break
    node.left = buildTree(preorder[1:num])
    node.right = buildTree(preorder[num:])
    return node

def postorder(tree):
    if tree != None:
        postorder(tree.left)
        postorder(tree.right)
        print(tree.value,end=' ')

n = int(input())
li = list(map(int,input().split()))
root = buildTree(li)
postorder(root)
```

## 05455: 二叉搜索树的层次遍历

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

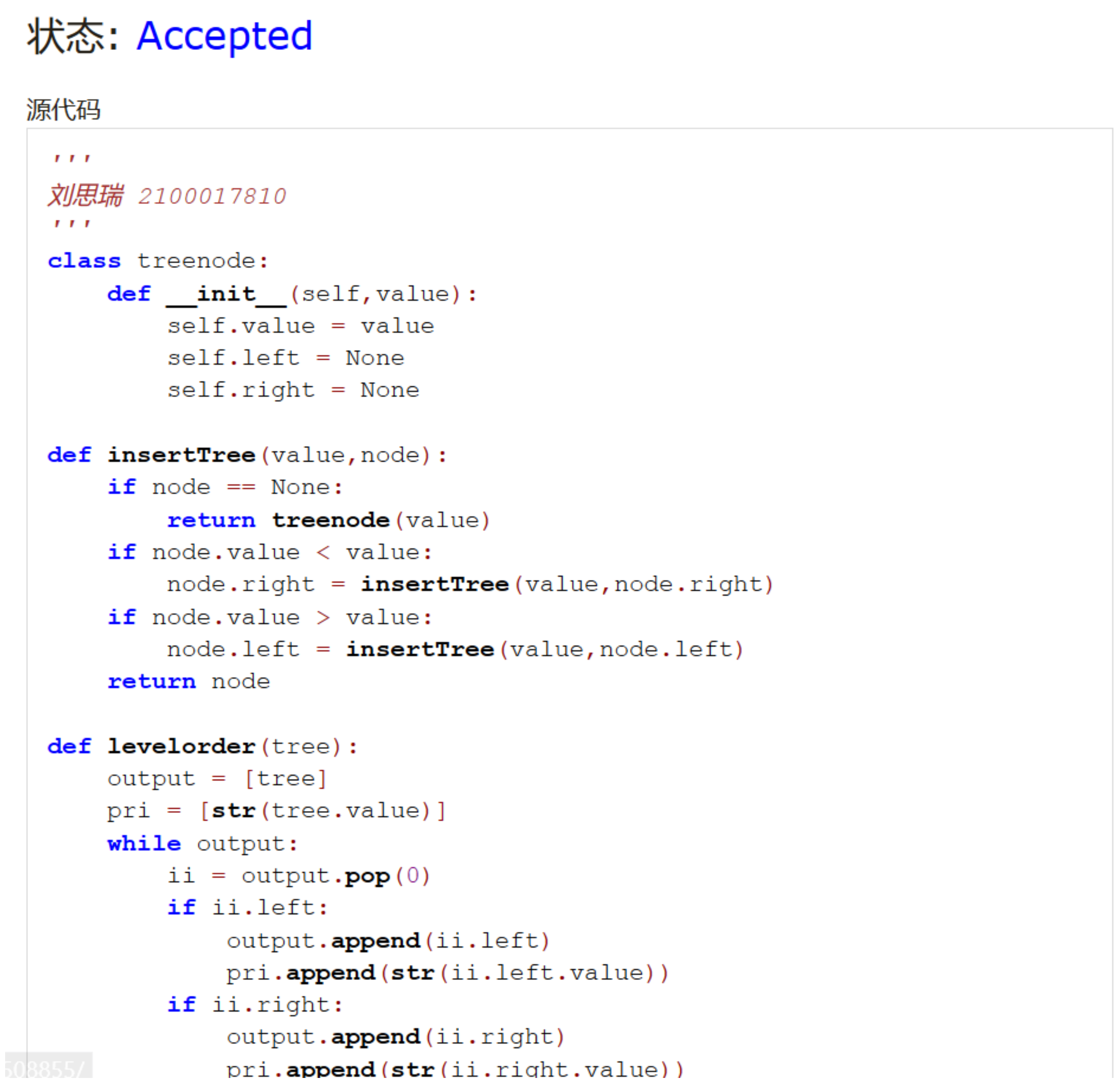
思路:

二叉树插入数据

代码

```python
'''
刘思瑞 2100017810
'''
class treenode:
    def __init__(self,value):
        self.value = value
        self.left = None
        self.right = None

def insertTree(value,node):
    if node == None:
        return treenode(value)
    if node.value < value:
        node.right = insertTree(value,node.right)
    if node.value > value:
        node.left = insertTree(value,node.left)
    return node

def levelorder(tree):
    output = [tree]
    pri = [str(tree.value)]
    while output:
        ii = output.pop(0)
        if ii.left:
            output.append(ii.left)
            pri.append(str(ii.left.value))
        if ii.right:
            output.append(ii.right)
            pri.append(str(ii.right.value))
    print(' '.join(pri))

li = list(map(int,input().split()))
root = None
for i in li:
    root = insertTree(i,root)
levelorder(root)
```

代码运行截图


状态: Accepted

源代码

```python
'''
刘思瑞 2100017810
'''
class treenode:
    def __init__(self,value):
        self.value = value
        self.left = None
        self.right = None

def insertTree(value,node):
    if node == None:
        return treenode(value)
    if node.value < value:
        node.right = insertTree(value,node.right)
    if node.value > value:
        node.left = insertTree(value,node.left)
    return node

def levelorder(tree):
    output = [tree]
    pri = [str(tree.value)]
    while output:
        ii = output.pop(0)
        if ii.left:
            output.append(ii.left)
            pri.append(str(ii.left.value))
        if ii.right:
            output.append(ii.right)
            pri.append(str(ii.right.value))
```

## 04078: 实现堆结构

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

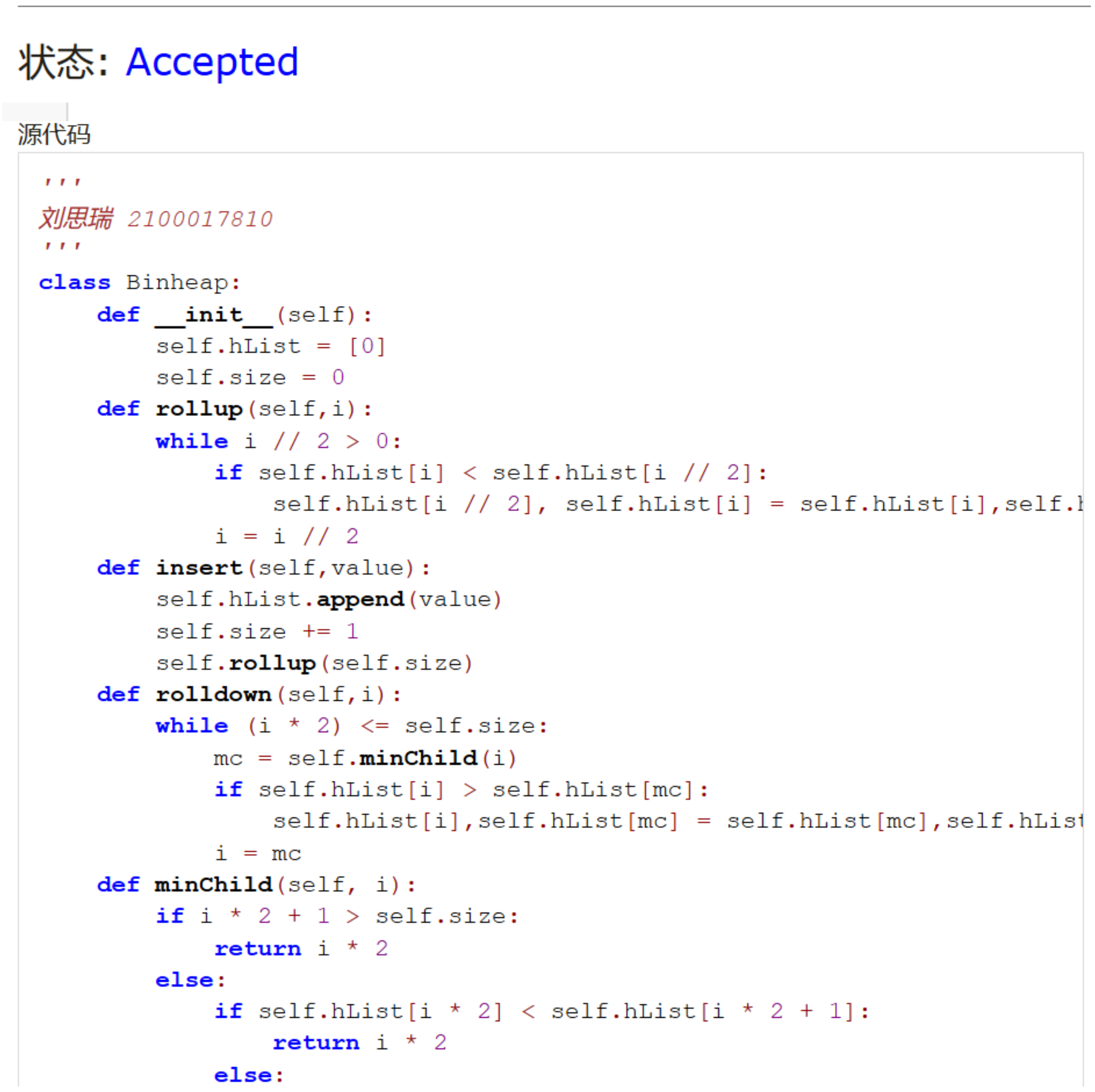
练习自己写个BinHeap。当然机考时候，如果遇到这样题目，直接import heapq。手搓栈、队列、堆、AVL等，考试前需要搓个遍。

思路：

代码

```python
'''
刘思瑞 2100017810
'''
class Binheap:
    def __init__(self):
        self.hList = [0]
        self.size = 0
    def rollup(self,i):
        while i // 2 > 0:
            if self.hList[i] < self.hList[i // 2]:
                self.hList[i // 2], self.hList[i] = self.hList[i],self.hList[i // 2]
            i = i // 2
    def insert(self,value):
        self.hList.append(value)
        self.size += 1
        self.rollup(self.size)
    def rolldown(self,i):
        while (i * 2) <= self.size:
            mc = self.minChild(i)
            if self.hList[i] > self.hList[mc]:
                self.hList[i],self.hList[mc] = self.hList[mc],self.hList[i]
            i = mc
    def minChild(self, i):
        if i * 2 + 1 > self.size:
            return i * 2
        else:
            if self.hList[i * 2] < self.hList[i * 2 + 1]:
                return i * 2
            else:
                return i * 2 + 1
    def delMin(self):
        retval = self.hList[1]
        self.hList[1] = self.hList[self.size]
        self.size = self.size - 1
        self.hList.pop()
        self.rolldown(1)
        return retval
    def buildHeap(self, alist):
        i = len(alist) // 2
        self.size = len(alist)
        self.hList = [0] + alist[:]
        while (i > 0):
            self.rolldown(i)
            i = i - 1


n = int(input())
m = Binheap()
for i in range(n):
    s = input()
    if s[0] =='1':
        m.insert(int(s[2:]))
    else:
        print(m.delMin())
```

代码运行截图



状态: Accepted

源代码

```
'''
刘思瑞 2100017810
'''
class Binheap:
    def __init__(self):
        self.hList = [0]
        self.size = 0
    def rollup(self,i):
        while i // 2 > 0:
            if self.hList[i] < self.hList[i // 2]:
                self.hList[i // 2], self.hList[i] = self.hList[i],self.h
            i = i // 2
    def insert(self,value):
        self.hList.append(value)
        self.size += 1
        self.rollup(self.size)
    def rolldown(self,i):
        while (i * 2) <= self.size:
            mc = self.minChild(i)
            if self.hList[i] > self.hList[mc]:
                self.hList[i],self.hList[mc] = self.hList[mc],self.hList
            i = mc
    def minChild(self, i):
        if i * 2 + 1 > self.size:
            return i * 2
        else:
            if self.hList[i * 2] < self.hList[i * 2 + 1]:
                return i * 2
            else:
```

## 22161: 哈夫曼编码树

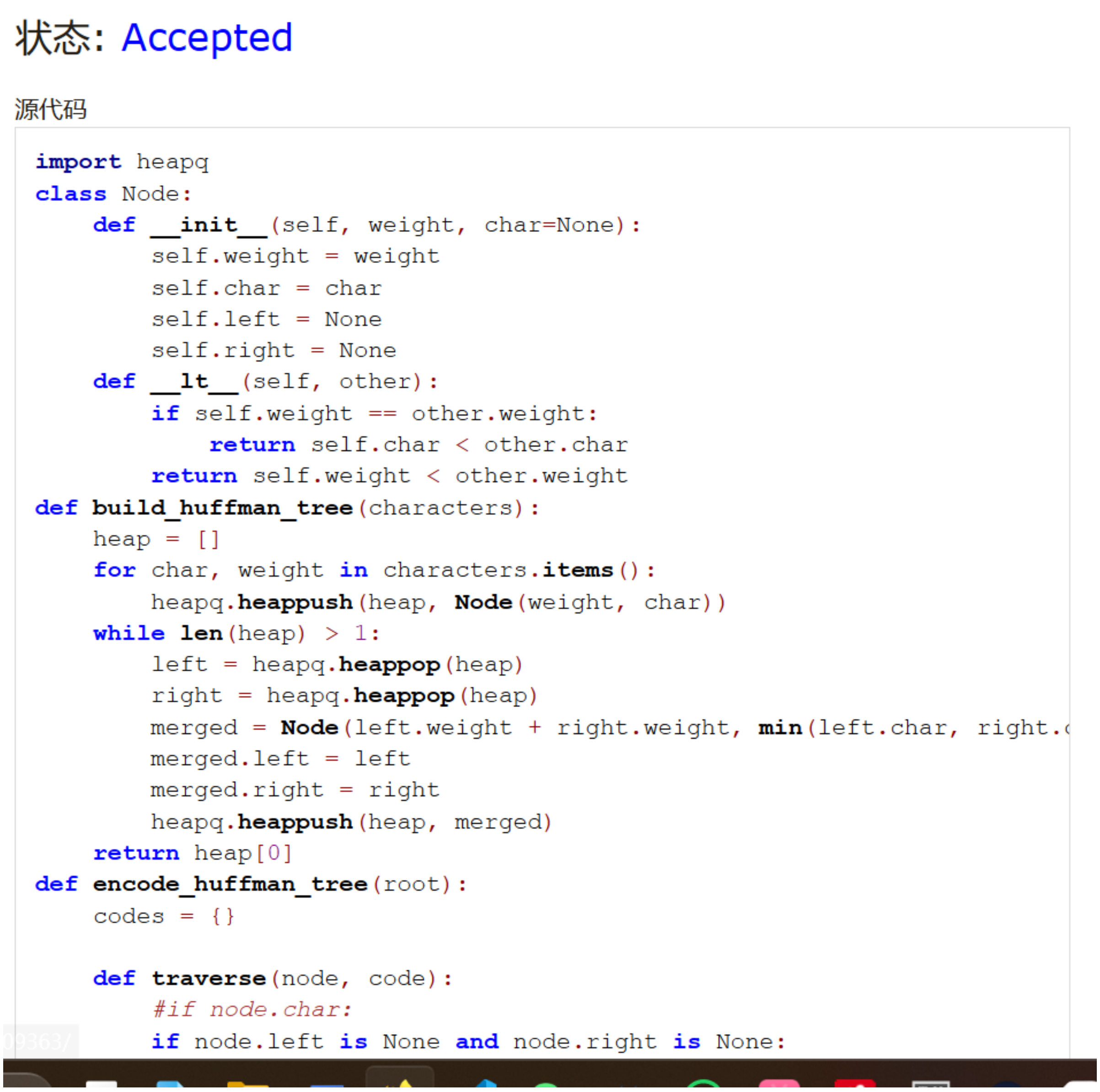
http://cs101.openjudge.cn/practice/22161/

思路:

代码

```python
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.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.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



状态: Accepted

源代码

```python
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.char:
        if node.left is None and node.right is None:
```

## 晴问9.5: 平衡二叉树的建立

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

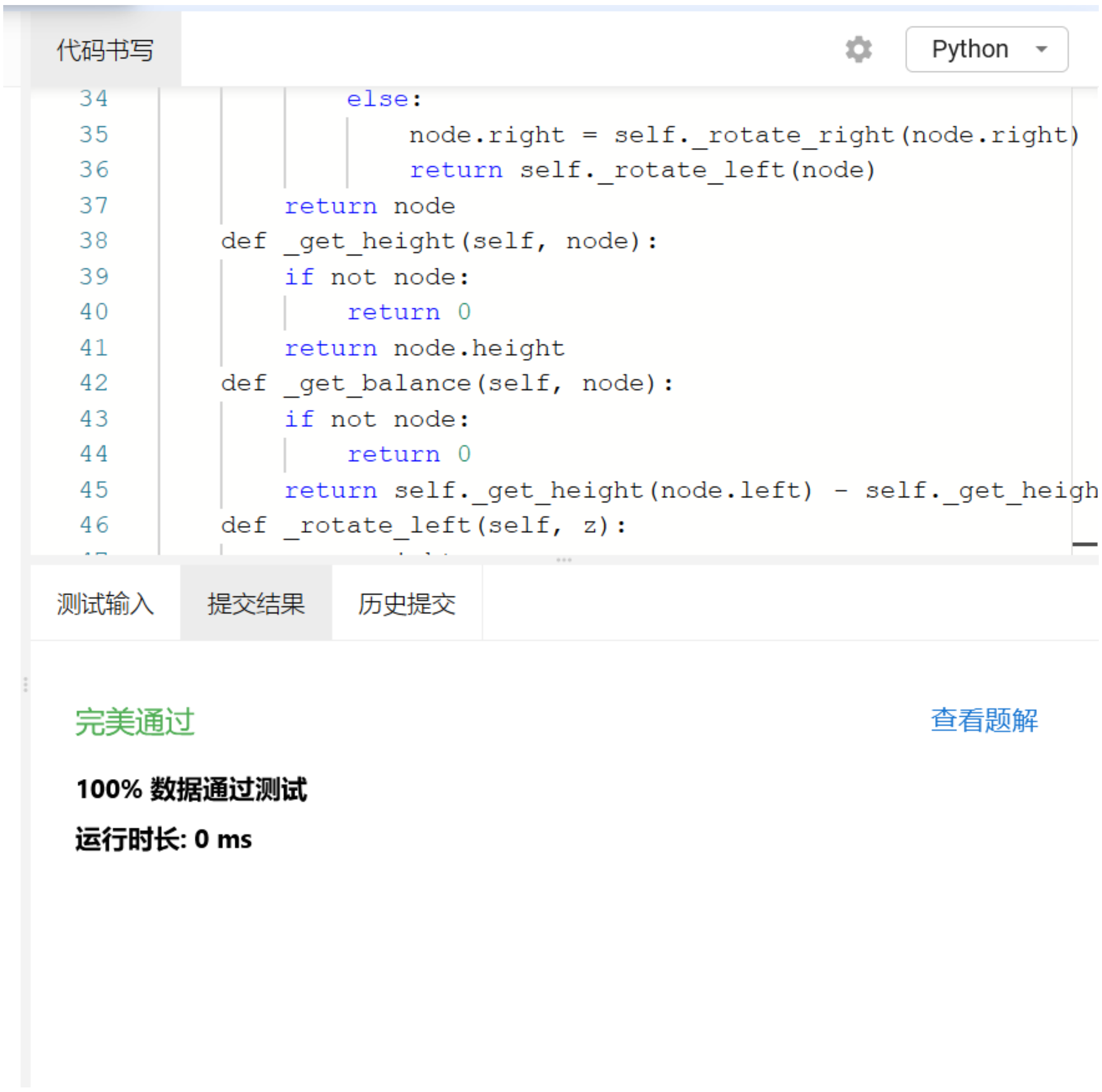
思路:

照猫画虎跟着讲义写了一遍代码，确实感觉复用才是精髓

代码

```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)
    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, 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



```
            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_heigh
    def _rotate_left(self, z):
```

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/

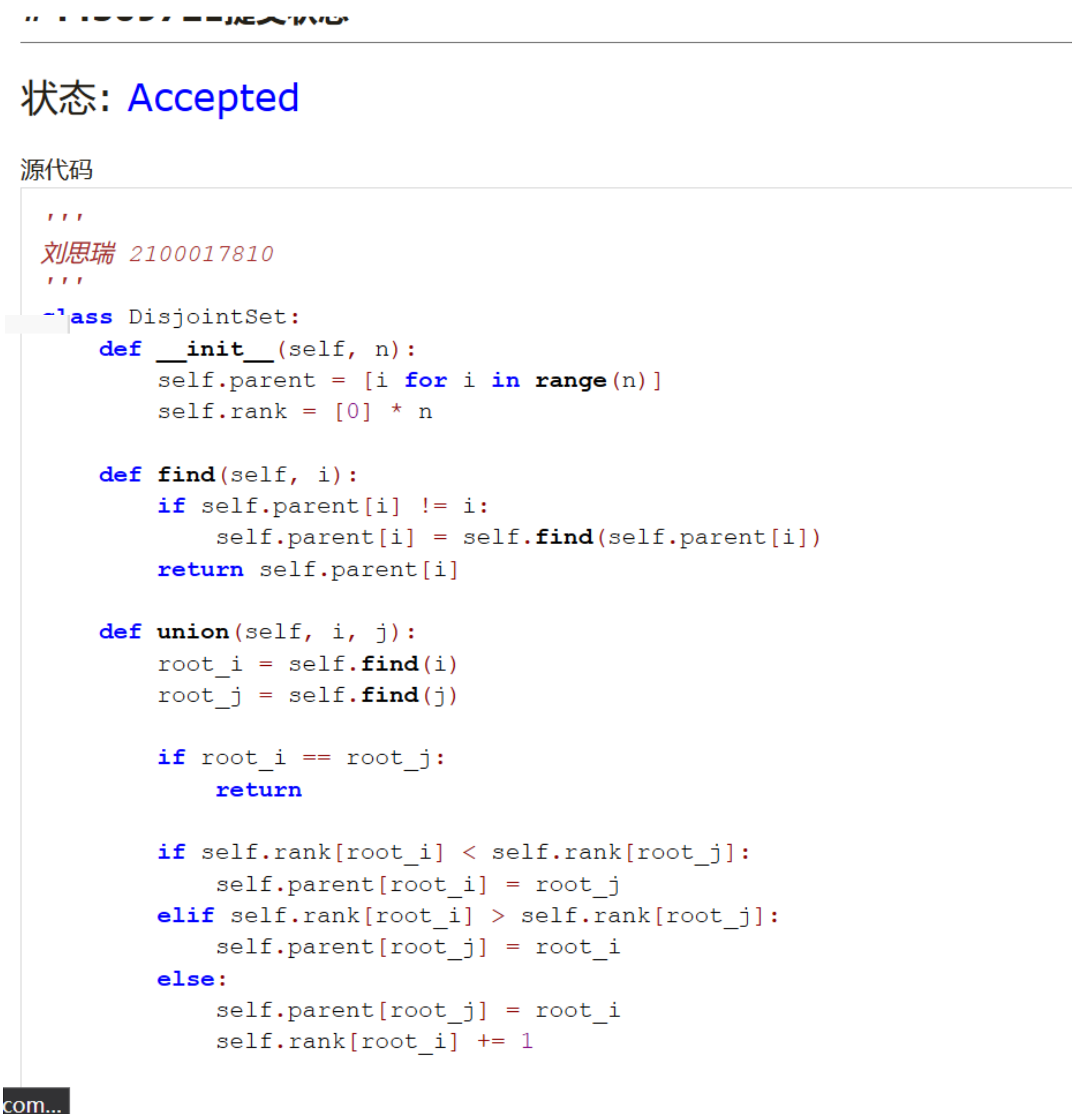
思路：

并查集

代码

```python
'''
刘思瑞 2100017810
'''
class DisjointSet:
    def __init__(self, n):
        self.parent = [i for i in range(n)]
        self.rank = [0] * n

    def find(self, i):
        if self.parent[i] != i:
            self.parent[i] = self.find(self.parent[i])
        return self.parent[i]

    def union(self, i, j):
        root_i = self.find(i)
        root_j = self.find(j)

        if root_i == root_j:
            return

        if self.rank[root_i] < self.rank[root_j]:
            self.parent[root_i] = root_j
        elif self.rank[root_i] > self.rank[root_j]:
            self.parent[root_j] = root_i
        else:
            self.parent[root_j] = root_i
            self.rank[root_i] += 1


def max_religions(n, m, edges):
    ds = DisjointSet(n)

    for edge in edges:
        ds.union(edge[0] - 1, edge[1] - 1)

    distinct_sets = set()
    for i in range(n):
        distinct_sets.add(ds.find(i))

    return len(distinct_sets)


def main():
    case = 1
    while True:
        n, m = map(int, input().split())
        if n == 0 and m == 0:
            break

        edges = []
        for _ in range(m):
            i, j = map(int, input().split())
            edges.append((i, j))


        result = max_religions(n, m, edges)
        print("Case {}: {}".format(case, result))
        case += 1


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

代码运行截图



状态: Accepted

源代码

```
'''
刘思瑞 2100017810
'''
class DisjointSet:
    def __init__(self, n):
        self.parent = [i for i in range(n)]
        self.rank = [0] * n

    def find(self, i):
        if self.parent[i] != i:
            self.parent[i] = self.find(self.parent[i])
        return self.parent[i]

    def union(self, i, j):
        root_i = self.find(i)
        root_j = self.find(j)

        if root_i == root_j:
            return

        if self.rank[root_i] < self.rank[root_j]:
            self.parent[root_i] = root_j
        elif self.rank[root_i] > self.rank[root_j]:
            self.parent[root_j] = root_i
        else:
            self.parent[root_j] = root_i
            self.rank[root_i] += 1
```

com...

## 2. 学习总结和收获

这次作业耗时非常长，感觉对树和堆有了新的理解，之前拿到树的结构不会操作不会实现算法现在觉得是因为没有意识到树的根节点的结构，我觉得根节点是树结构非常重要的部分，构建出根节点接下来不管是递归还是搜索都会有思路。另一点是关于代码的复用，合理的copy真的很有用，这样在平时的练习中可以更多关注算法，思路也会跟会更加清晰