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

Updated 2214 GMT+8 March 24, 2024

2024 spring, Complied by 李佳霖，心理与认知科学学院

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：macOS Sonoma

Python编程环境：VSCode

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：二叉搜索树具有如下定义：

1. 若任意节点的左子树不空，则左子树上所有节点的值均小于它的根节点的值；
2. 若任意节点的右子树不空，则右子树上所有节点的值均大于它的根节点的值；
3. 任意节点的左、右子树也分别为二叉查找树；

根据此定义，通过递归的方式遍历输入的节点，再以后序遍历的方式返回即可

代码

```
#
def preorder_to_postorder(preorder):
    if not preorder:
        return []

    root = preorder[0]
    left_preorder = [x for x in preorder[1:] if x < root]
    right_preorder = [x for x in preorder[1:] if x > root]

    left_postorder = preorder_to_postorder(left_preorder)
    right_postorder = preorder_to_postorder(right_preorder)

    return left_postorder + right_postorder + [root]

n = int(input())
preorder = list(map(int, input().split()))
postorder = preorder_to_postorder(preorder)
print(' '.join(map(str, postorder)))
```

代码运行截图 （至少包含有"Accepted"）

#44453235提交状态

查看 提交 统计 提问

状态: Accepted

源代码

```
def preorder_to_postorder(preorder):
    if not preorder:
        return []

    root = preorder[0]
    left_preorder = [x for x in preorder[1:] if x < root]
    right_preorder = [x for x in preorder[1:] if x > root]
```

基本信息

#: 44453235
题目: 22275
提交人: 李佳霖2000013713
内存: 3872kB
时间: 25ms
语言: Python3
提交时间: 2024-03-29 23:20:40

05455: 二叉搜索树的层次遍历

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

思路：来自老师教案

The problem is asking to construct a binary search tree (BST) from a sequence of numbers and then perform a level order traversal (also known as breadth-first search) on the BST.

Here is a step-by-step plan:

1. Create a TreeNode class to represent each node in the tree.
2. Create a function `insert` that takes a node and a value as input and inserts the value into the BST rooted at the node.
3. Create a function `level_order_traversal` that takes the root of the tree as input and returns the level order traversal of the tree.
    - Use a queue to store the nodes to be visited.
    - While the queue is not empty, dequeue a node, visit it, and enqueue its children.
4. Read the sequence of numbers from the input, construct the BST, perform the level order traversal, and output the result.

代码

```
#
from collections import deque

numbers = list(map(int, input().strip().split()))
numbers = list(dict.fromkeys(numbers)) # remove duplicates

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

# 永远从根开始往下插入，遇到比当前根节点值小的往左插，否则往右插
def insert(node, value):
  if node is None:
    return TreeNode(value)
  if value < node.val:
    node.left = insert(node.left, value)
  else:
    node.right = insert(node.right, value)
  return node

def traversal(root):
  queue = deque([root])
  travesal = []
  while queue:
    node = queue.popleft()
```

```
        travesal.append(node.val)
        if node.left:
            queue.append(node.left)
        if node.right:
            queue.append(node.right)
    return travesal

root = None
for number in numbers:
    root = insert(root, number)
traversal = traversal(root)
print(' '.join(map(str, traversal)))
```

代码运行截图 （至少包含有"Accepted"）

#44453511提交状态

查看 提交 统计 提问

状态: Accepted

源代码

```
from collections import deque

numbers = list(map(int, input().strip().split()))
numbers = list(dict.fromkeys(numbers)) # remove duplicates

class TreeNode:
    def __init__(self, x):
```

基本信息

#:	44453511
题目:	05455
提交人:	李佳霖2000013713
内存:	3672kB
时间:	25ms
语言:	Python3
提交时间:	2024-03-30 00:00:27

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：参照了郭炜老师的堆实现

代码

```
#
class Heap():
    def __init__(self, array = [], less = lambda x,y:x<y):
        self._a = array[:]
        self._size = len(array)
```

```
    self._less = less
    self.makeHeap()

  def top(self):
    return self._a[0]

  def pop(self):
    tmp = self._a[0]
    self._a[0] = self._a[-1]
    self._a.pop()
    self._size -= 1
    self._goDown(0)
    return tmp

  def append(self, x):
    self._size +=1
    self._a.append(x)
    self._goUp(self._size - 1)

  def _goUp(self, i):
    if i == 0:
      return
    f = (i-1)//2
    if self._less(self._a[i], self._a[f]):
      self._a[i], self._a[f] = self._a[f], self._a[i]
      self._goUp(f)

  def _goDown(self,i):
    if i*2 + 1 >= self._size:
      return
    L, R = i*2 + 1, i*2+2
    if R >= self._size or self._less(self._a[L], self._a[R]):
      s = L
    else:
      s = R

    if self._less(self._a[s], self._a[i]):
      self._a[i], self._a[s] = self._a[s], self._a[i]
      self._goDown(s)

  def makeHeap(self):
    i = (self._size - 1 - 1)// 2
    for k in range(i, -1, -1):
      self._goDown(k)

  def heapSort(self):
    for i in range(self._size-1, -1, -1):
```

```
            self._a[i], self._a[0] = self._a[0], self._a[i]
            self._size -= 1
            self._goDown(0)

        self._a.reverse()
        return self._a


n = int(input())
BinHeap = Heap()
for _ in range(n):
    op = list(map(int,input().split()))
    if op[0] == 1:
        BinHeap.append(op[1])

    if op[0] == 2:
        num = BinHeap.pop()
        print(num)
```

代码运行截图 （AC代码截图，至少包含有"Accepted"）

#44468517提交状态

查看 提交 统计 提问

状态: Accepted

源代码

```
class Heap():
    def __init__(self, array = [], less = lambda x,y:x<y):
        self._a = array[:]
        self._size = len(array)
        self._less = less
        self.makeHeap()
```

基本信息

#: 44468517
题目: 04078
提交人: 李佳霖2000013713
内存: 4704kB
时间: 893ms
语言: Python3
提交时间: 2024-03-30 18:04:04

22161: 哈夫曼编码树

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

思路：

heapq模块的操作见：https://docs.python.org/zh-cn/3/library/heapq.html

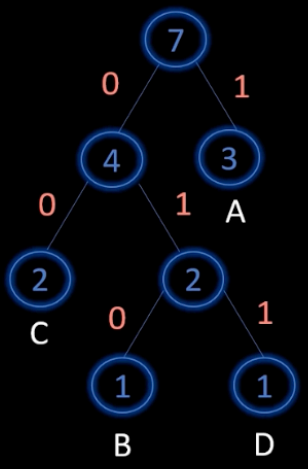
不断的合并最小两个。出现次数越多的字符，编的码越短（因为最后合并的），最早合并的在最下面

如何保证没有歧义：叶子结点不可能在另外一个叶子结点的路上

哈夫曼编码

蓝不过海呀 bilibili

A B A A C D C

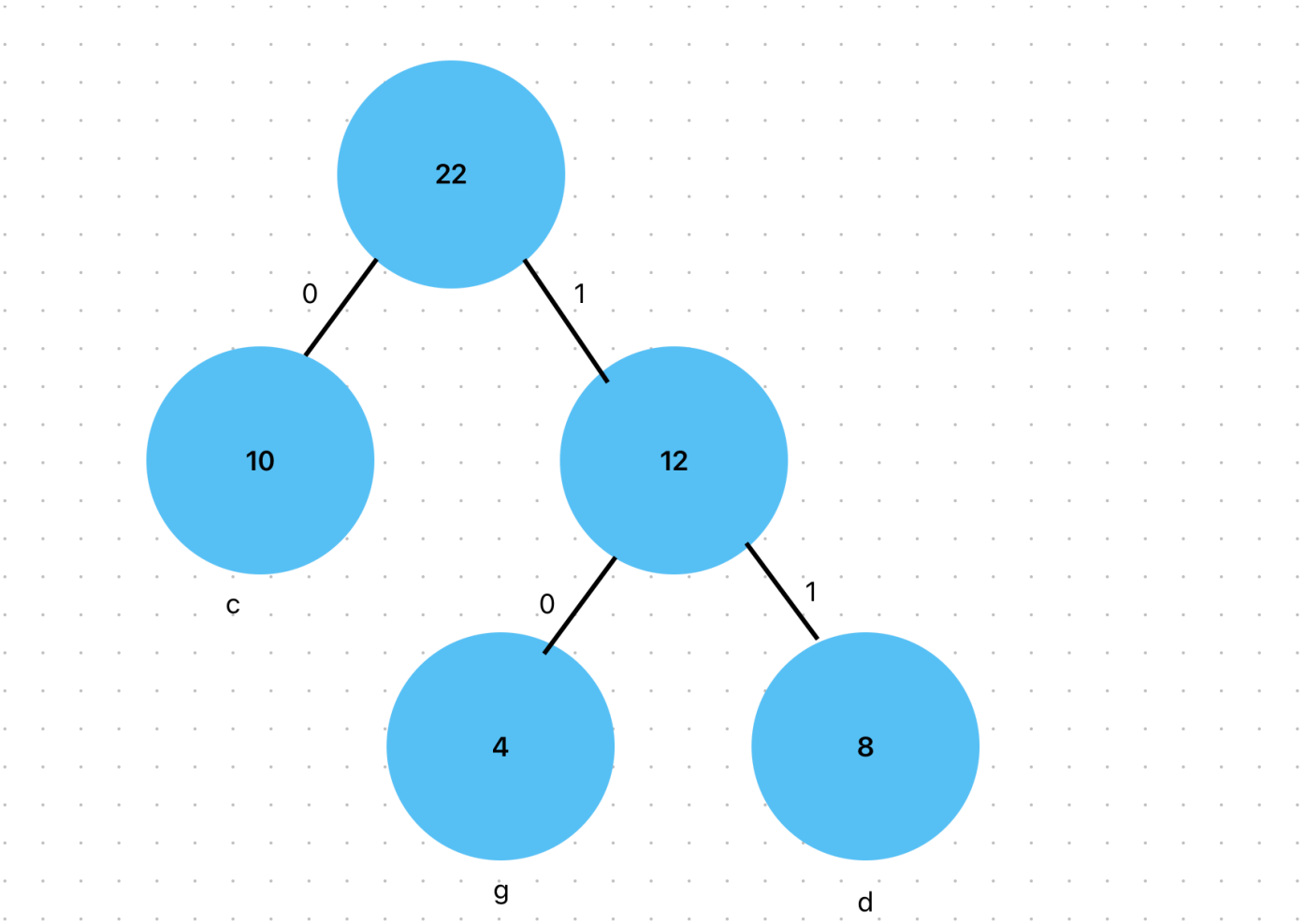


A	1
B	010
C	00
D	011

出现次数越多的字符编的码越短

1010110001100

不出现歧义的情况下编出来的码长度最短



代码

```
#
import heapq

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

    # 如果两个Node对象的权重相同，则按照它们的字符顺序进行比较。这意味着如果两个节点的权重相同，则会根据它们的字符（char属性）的字典顺序进行排序。
    def __lt__(self, other):
        if self.weight == other.weight:
            return self.char < other.char
        return self.weight < other.weight
```

```python
def build_huffman_tree(characters):
    heap = []

    # 将结点加入到堆中，保持堆的不变性。
    for char, weight in characters.items():
        heapq.heappush(heap, Node(weight, char))

    # 合并堆，直至只剩一个
    while len(heap) > 1:

        # 弹出并返回 heap 的最小的两个元素，在本题中即为出现次数最少的两个元素，并保持堆的不变性。
        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
        # 合并两个最小的元素，算出其父结点的值，字符值用左结点和右结点中小的代替（为什么？）
        merged = Node(left.weight + right.weight, min(left.char, right.char))
        merged.left = left
        merged.right = right
        heapq.heappush(heap, merged)

    return heap[0]

# 对哈夫曼树的结点进行编码，用递归对每个结点值进行编码
def encode_huffman_tree(root):
    codes = {}

    def traverse(node, code):
        # if node.char: 可能和上面改动的地方对应
        # 如果左右子结点均为None，编码即为自身
        if node.left is None and node.right is None:
            codes[node.char] = code
        # 左结点赋0，右结点赋1
        else:
            traverse(node.left, code + '0')
            traverse(node.right, code + '1')

    # 从根结点开始编码
    traverse(root, '')
    return codes

# 对输入的字符进行哈夫曼编码，直接索引字典即可
def huffman_encoding(codes, string):
    encoded = ''
    for char in string:
        encoded += codes[char]
    return encoded

# 对输入的哈夫曼编码进行解码
```

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

        # 只有左右子结点的值均为None时也就是无叶子结点时，才会把解码的字符添加到解码字符串中
        # 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:
    # 如果输入的是哈夫曼编码
    if string[0] in ('0','1'):
        results.append(huffman_decoding(huffman_tree, string))
    # 如果输入的是哈夫曼字符
```

```
    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"）

#44474715提交状态 查看 提交 统计 提问

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
```

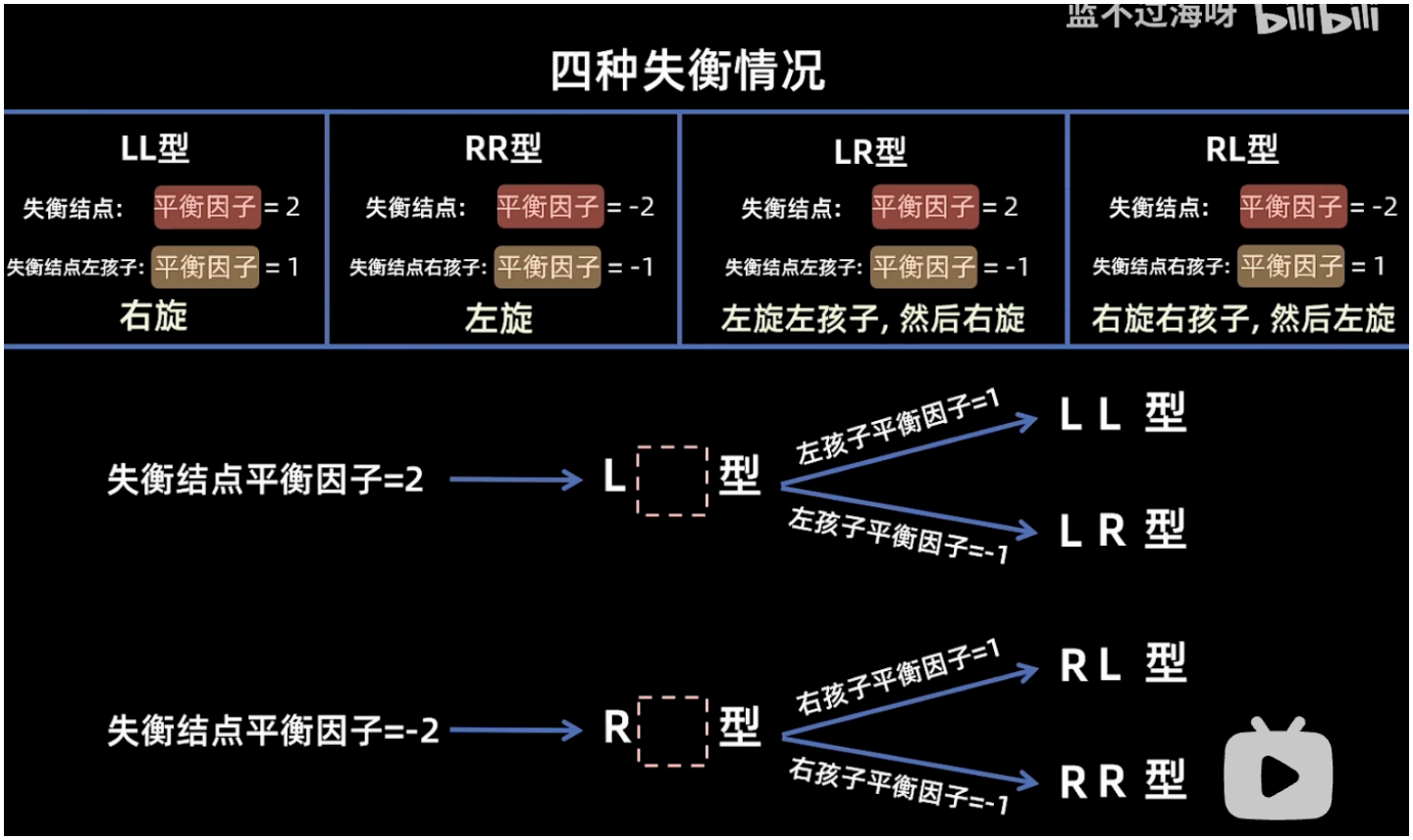
基本信息

#:	44474715
题目:	22161
提交人:	李佳霖2000013713
内存:	3744kB
时间:	28ms
语言:	Python3
提交时间:	2024-03-31 01:36:34

晴问9.5: 平衡二叉树的建立

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

思路：



代码

```
#
class TreeNode:
    def __init__(self, val):
        self.val = val
        self.left = None
        self.right = None
        self.height = 1

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

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

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

```
    balance = self.getBalance(root)

    # Left Left Case
    if balance > 1 and val < root.left.val:
        return self.rightRotate(root)

    # Right Right Case
    if balance < -1 and val > root.right.val:
        return self.leftRotate(root)

    # Left Right Case
    if balance > 1 and val > root.left.val:
        root.left = self.leftRotate(root.left)
        return self.rightRotate(root)

    # Right Left Case
    if balance < -1 and val < root.right.val:
        root.right = self.rightRotate(root.right)
        return self.leftRotate(root)

    return root

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)

# 左旋操作，其中z为失衡结点，y为失衡结点的左子树，T2为失衡结点的左孩
# 右旋就是把失衡结点z连给y的左子结点，把T2连接给失衡结点z的右结点
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

# 右旋操作，其中y为失衡结点，x为失衡结点的左子树，T2为失衡结点的右孩
```

```
    # 右旋就是把y连给x的右结点，把T2连接给失衡结点y的左结点
    def rightRotate(self, y):
        x = y.left
        T2 = x.right

        x.right = y
        y.left = T2

        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 preOrderTraversal(self, root):
        result = []
        if root:
            result.append(root.val)
            result += self.preOrderTraversal(root.left)
            result += self.preOrderTraversal(root.right)
        return result

# 输入
n = int(input())
elements = list(map(int, input().split()))

# 构建 AVL 树
avl_tree = AVLTree()
for element in elements:
    avl_tree.root = avl_tree.insert(avl_tree.root, element)

# 输出先序遍历序列
pre_order_result = avl_tree.preOrderTraversal(avl_tree.root)
# for val in pre_order_result:
#     print(val, end=" ")
print(' '.join(str(val) for val in pre_order_result))
```

代码运行截图 （AC代码截图，至少包含有"Accepted"）

02524: 宗教信仰

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

思路：通过列表实现并查集结构，通过迭代或递归地访问元素的父节点，直到找到根节点为止，并且在路径上进行路径压缩，以提高后续查找的效率。

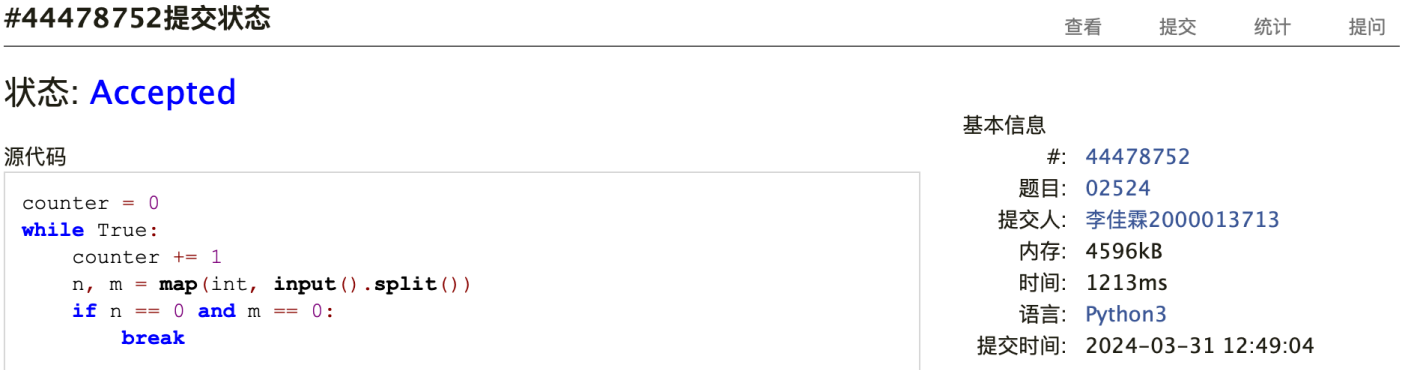
```
def find(parent, x):
    if parent[x] != x: # 如果当前节点的父节点不是自己，则继续向上找
        parent[x] = find(parent, parent[x]) # 路径压缩：将当前节点的父节点直接指向根节点
    return parent[x] # 返回根节点的索引值
```

代码

```
#
counter = 0
while True:
    counter += 1
    n, m = map(int, input().split())
    if n == 0 and m == 0:
        break

    if m == 0: # 处理无有效宗教连接的情况
        print('Case {}:'.format(counter), n)
```

```
    continue

num = n  # 最坏情况下每个学生信仰不同宗教
religion = [-1] * n

def find(rel, x):
    if rel[x] == -1:
        return x
    rel[x] = find(rel, rel[x])
    return rel[x]

for _ in range(m):
    i, j = map(int, input().split())
    root_i = find(religion, i-1)
    root_j = find(religion, j-1)
    if root_i != root_j:
        religion[root_i] = root_j
        num -= 1  # 连接了不同宗教的学生，宗教数目减少1

print('Case {}:'.format(counter), num)
```

代码运行截图 （AC代码截图，至少包含有"Accepted"）



#44478752提交状态

查看 提交 统计 提问

状态: Accepted

源代码

```
counter = 0
while True:
    counter += 1
    n, m = map(int, input().split())
    if n == 0 and m == 0:
        break
```

基本信息

#: 44478752
题目: 02524
提交人: 李佳霖2000013713
内存: 4596kB
时间: 1213ms
语言: Python3
提交时间: 2024-03-31 12:49:04

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。

这周的作业对我来说做起来还是相当费劲的，每道题从开始学习到自己尝试到看题解的时长都在一小时以上。虽然感觉看题解时理解了，但真正到自己复现代码时还是相当困难。这里强烈推荐b站Up主‘蓝不过海呀’的视频，动画做的非常清晰直接。假期的时候打算再过一遍。