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

Updated 2214 GMT+8 March 24, 2024

2024 spring, Complied by 周添 物理学院

1. 题目

22275: 二叉搜索树的遍历

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

代码

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


def build_tree(preorder):
    if not preorder:
        return None
    root = TreeNode(preorder[0])
    n = len(preorder)
    if n == 1:
        return root
    v = root.value
    i = 1
    while i < n and preorder[i] < v:
        i += 1
    root.left = build_tree(preorder[1:i])
    root.right = build_tree(preorder[i:])
    return root


def print_tree(node):
    if not node:
        return
    print_tree(node.left)
    print_tree(node.right)
    print(node.value, end=' ')


nn = int(input())
ppr = [int(i) for i in input().split()]
```

```
tree = build_tree(ppr)
print_tree(tree)
```

状态: Accepted

源代码

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


def build_tree(preorder):
    if not preorder:
        return None
    root = TreeNode(preorder[0])
    n = len(preorder)
    if n == 1:
        return root
    v = root.value
    i = 1
    while i < n and preorder[i] < v:
        i += 1
    root.left = build_tree(preorder[1:i])
    root.right = build_tree(preorder[i:])
    return root


def print_tree(node):
    if not node:
        return
    print_tree(node.left)
    print_tree(node.right)
    print(node.value, end=' ')
```

基本信息

#: 44395906
题目: 22275
提交人: 23n2300011538
内存: 4108kB
时间: 28ms
语言: Python3
提交时间: 2024-03-25 16:01:14

05455: 二叉搜索树的层次遍历

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

代码

```
from collections import deque


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


def add_leaves(s, k):
    if k == s.value:
        return
    elif k < s.value:
        if not s.left:
            s.left = TreeNode(k)
        else:
```

```
            add_leaves(s.left, k)
    else:
        if not s.right:
            s.right = TreeNode(k)
        else:
            add_leaves(s.right, k)


def build_tree(xxx):
    root = TreeNode(xxx[0])
    for i in xxx:
        add_leaves(root, i)
    return root


def print_tree_level_order(root):
    if not root:
        return
    queue = deque([root])
    while queue:
        node = queue.popleft()
        print(node.value, end=' ')

        if node.left:
            queue.append(node.left)
        if node.right:
            queue.append(node.right)


xs = [int(i) for i in input().split()]
tree = build_tree(xs)
print_tree_level_order(tree)
```

状态: Accepted

基本信息

#: 44396525
题目: 05455
提交人: 23n2300011538
内存: 3660kB
时间: 26ms
语言: Python3
提交时间: 2024-03-25 16:42:35

源代码

```
from collections import deque


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


def add_leaves(s, k):
    if k == s.value:
        return
    elif k < s.value:
        if not s.left:
            s.left = TreeNode(k)
        else:
            add_leaves(s.left, k)
    else:
        if not s.right:
            s.right = TreeNode(k)
        else:
            add_leaves(s.right, k)


def build_tree(xxx):
    root = TreeNode(xxx[0])
    for i in xxx:
```

04078: 实现堆结构

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

代码

```
class MinHeap:
    def __init__(self):
        self.heap = []

    def heap_push(self, item):
        self.heap.append(item)
        self.heapify_up(len(self.heap)-1)

    def heap_pop(self):
        if len(self.heap) == 0:
            return None
        print(self.heap[0])
        last_val = self.heap.pop()
        if len(self.heap) > 0:
            self.heap[0] = last_val
            self.heapify_down(0)

    def heapify_up(self, i):
        while i > 0:
            parent = (i - 1) // 2
            if self.heap[parent] > self.heap[i]:
                self.heap[parent], self.heap[i] = self.heap[i], self.heap[parent]
                i = parent
            else:
                break
```

```
    def heapify_down(self, i):
        left_child = 2 * i + 1
        right_child = 2 * i + 2
        smallest = i
        if left_child < len(self.heap) and self.heap[left_child] <
self.heap[smallest]:
            smallest = left_child
        if right_child < len(self.heap) and self.heap[right_child] <
self.heap[smallest]:
            smallest = right_child
        if smallest != i:
            self.heap[smallest], self.heap[i] = self.heap[i], self.heap[smallest]
            self.heapify_down(smallest)


n = int(input())
s = MinHeap()
for _ in range(n):
    c = input()
    if c[0] == '2':
        s.heap_pop()
    else:
        l, r = map(int, c.split())
        s.heap_push(r)
    #print('got it')
```

状态: Accepted

源代码

```
class MinHeap:
    def __init__(self):
        self.heap = []

    def heap_push(self, item):
        self.heap.append(item)
        self.heapify_up(len(self.heap)-1)

    def heap_pop(self):
        if len(self.heap) == 0:
            return None
        print(self.heap[0])
        last_val = self.heap.pop()
        if len(self.heap) > 0:
            self.heap[0] = last_val
            self.heapify_down(0)

    def heapify_up(self, i):
        while i > 0:
            parent = (i - 1) // 2
            if self.heap[parent] > self.heap[i]:
                self.heap[parent], self.heap[i] = self.heap[i], self.hea
                i = parent
            else:
                break

    def heapify_down(self, i):
        left_child = 2 * i + 1
        right_child = 2 * i + 2
        smallest = i
        if left_child < len(self.heap) and self.heap[left_child] < self
```

基本信息

#:	44401323
题目:	04078
提交人:	23n2300011538
内存:	4688kB
时间:	637ms
语言:	Python3
提交时间:	2024-03-25 21:44:07

22161: 哈夫曼编码树

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

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

huffman_tree = build_huffman_tree(chars)
codes = encode_huffman_tree(huffman_tree)

while True:
    try:
        line = input()
        if line[0] in ('0', '1'):
            print(huffman_decoding(huffman_tree, line))
        else:
            print(huffman_encoding(codes, line))
    except EOFError:
        break
```

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(chars_and_weights):
    heap = []
    for char, weight in chars_and_weights.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.c
        merged.left = left
        merged.right = right
        heapq.heappush(heap, merged)

    return heap[0]
```

基本信息

#:	44411563
题目:	22161
提交人:	23n2300011538
内存:	3616kB
时间:	22ms
语言:	Python3
提交时间:	2024-03-26 19:52:59

晴问9.5: 平衡二叉树的建立

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

代码

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


class AVLTree:
    def insert(self, root, key):
        if not root:
            return AVLNode(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.get_height(root.left),
self.get_height(root.right))

        balance = self.get_balance(root)

        # 左旋转
        if balance > 1 and key < root.left.key:
```

```
            return self.rotate_right(root)
        # 右旋转
        if balance < -1 and key > root.right.key:
            return self.rotate_left(root)
        # 左-右旋转
        if balance > 1 and key > root.left.key:
            root.left = self.rotate_left(root.left)
            return self.rotate_right(root)
        # 右-左旋转
        if balance < -1 and key < root.right.key:
            root.right = self.rotate_right(root.right)
            return self.rotate_left(root)

        return root

    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_right(self, z):
        y = z.left
        T3 = y.right

        y.right = z
        z.left = T3

        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_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 pre_order_traversal(self, root):
        result = []
        if root:
            result.append(str(root.key))
            result += self.pre_order_traversal(root.left)
            result += self.pre_order_traversal(root.right)
        return result
```

```
n = int(input())
values = list(map(int, input().split()))

avl_tree = AVLTree()
root = None
for value in values:
    root = avl_tree.insert(root, value)

result = avl_tree.pre_order_traversal(root)
print(' '.join(result))
```

```
54
55          z.height = 1 + max(self.get_height(z.left), self.get
56          y.height = 1 + max(self.get_height(y.left), self.get
57
58          return y
59
60      def rotate_left(self, z):
61          y = z.right
62          T2 = y.left
63
64          y.left = z
65          z.right = T2
66
67          z.height = 1 + max(self.get_height(z.left), self.get
```

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/

代码

```
def find(parent, i):
    if parent[i] == i:
```

```
        return i
    return find(parent, parent[i])

def union(parent, rank, x, y):
    x_root = find(parent, x)
    y_root = find(parent, y)

    if rank[x_root] < rank[y_root]:
        parent[x_root] = y_root
    elif rank[x_root] > rank[y_root]:
        parent[y_root] = x_root
    else:
        parent[y_root] = x_root
        rank[x_root] += 1

def estimate_religions(n, edges):
    parent = [i for i in range(n+1)]
    rank = [0] * (n+1)

    for edge in edges:
        x, y = edge
        union(parent, rank, x, y)

    religions = set()
    for i in range(1, n+1):
        religions.add(find(parent, i))

    num_religions = len(religions)
    return num_religions


case_num = 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))

    max_religions = estimate_religions(n, edges)
    print(f"Case {case_num}: {max_religions}")
    case_num += 1
```

状态：Accepted

源代码

```python
def find(parent, i):
    if parent[i] == i:
        return i
    return find(parent, parent[i])

def union(parent, rank, x, y):
    x_root = find(parent, x)
    y_root = find(parent, y)

    if rank[x_root] < rank[y_root]:
        parent[x_root] = y_root
    elif rank[x_root] > rank[y_root]:
        parent[y_root] = x_root
    else:
        parent[y_root] = x_root
        rank[x_root] += 1

def estimate_religions(n, edges):
    parent = [i for i in range(n+1)]
    rank = [0] * (n+1)

    for edge in edges:
        x, y = edge
        union(parent, rank, x, y)

    religions = set()
    for i in range(1, n+1):
        religions.add(find(parent, i))

    num_religions = len(religions)
    return num_religions
```

基本信息

#：	44405716
题目：	02524
提交人：	23n2300011538
内存：	20028kB
时间：	1506ms
语言：	Python3
提交时间：	2024-03-26 13:52:47

2. 学习总结和收获

手搓有点难呜呜呜

huffman和avl是抄的，但是是手动敲的而不是Ctrl c + Ctrl v，敲了一遍以后感觉理解更好一点

额外题目的话，帮室友做了几道动态规划