

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

Updated 2214 GMT+8 March 24, 2024

2024 spring, Compiled by 尹柚鑫 光华管理学院 2100015878

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: win11

Python编程环境: jupyter notebook

1. 题目

22275: 二叉搜索树的遍历

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

耗时: 30mins

思路: 找第一个比第一个数大的数的位置，这个数的左边都比第一个数小，构成左子树，这个数和右边都比第一个数大，构成右子树，可以进行递归建树

代码

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

def find_first_larger(nums, target):
    for i, num in enumerate(nums):
```

```

        if num > target:
            return i
    return -1 # 如果没有找到比目标数大的数，则返回列表长度

def build_tree(num_list):
    if len(num_list)==0:
        return None
    if len(num_list)==1:
        node=TreeNode(num_list[0])
        return node
    median_index=find_first_larger(num_list,num_list[0])
    if median_index==-1:
        small=num_list[1:]
        big=[]
    else:
        small=num_list[1:median_index]
        big=num_list[median_index:]
    node=TreeNode(num_list[0])
    node.left=build_tree(small)
    node.right=build_tree(big)
    return node

def posttree(root):
    output=[]
    def post_helper(node):
        if node!=None:
            post_helper(node.left)
            post_helper(node.right)
            output.append(str(node.value))
    post_helper(root)

    return ' '.join(output)

def main():
    n=input()
    s=input().strip()
    num_list=[*map(int,s.split())]
    node=build_tree(num_list)
    print(posttree(node))

if __name__ == '__main__':
    main()

```

代码运行截图 == (至少包含有"Accepted") ==

状态: Accepted

源代码

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

    def find_first_larger(nums, target):
        for i, num in enumerate(nums):
            if num > target:
                return i
        return -1 # 如果没有找到比目标数大的数, 则返回列表长度

    def build_tree(num_list):
        if len(num_list) == 0:
            return None
        if len(num_list) == 1:
            node = TreeNode(num_list[0])
            return node
        median_index = find_first_larger(num_list, num_list[0])
        if median_index == -1:
```

基本信息

#: 44505618
题目: 22275
提交人: 尹柚鑫(2100015878)
内存: 4136kB
时间: 28ms
语言: Python3
提交时间: 2024-04-02 14:44:19

05455: 二叉搜索树的层次遍历

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

耗时: 20mins

思路: 遍历每一个数, 不断比较值的大小关系, 确定是向左子树去还是右子树去, 从而建立这个树。层次遍历的部分用队列的想法很容易实现

代码

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

    def build_tree(numlist):
        if len(numlist) == 0:
            return ''
        root = TreeNode(numlist[0])
        for i in numlist[1:]:
            node = root
            while True:
                if i > node.value:
                    if node.right != None:
                        node = node.right
                    else:
                        node.right = TreeNode(i)
                        break
                elif i < node.value:
                    if node.left != None:
                        node = node.left
                    else:
                        node.left = TreeNode(i)
```

```

        break
    else:
        break
    return root

def level_output(root):
    queue=[root]
    output=[]
    while queue:
        node=queue.pop(0)
        output.append(node.value)
        if node.left:
            queue.append(node.left)
        if node.right:
            queue.append(node.right)
    return output

def main():
    num=input().strip()
    numlist=[*map(int,num.split())]
    root=build_tree(numlist)
    output=level_output(root)
    s=' '.join(map(str,output))
    print(s)

if __name__ == '__main__':
    main()

```

代码运行截图 == (至少包含有"Accepted") ==

#44506073提交状态

[查看](#) [提交](#) [统计](#) [提问](#)

状态: **Accepted**

源代码

```

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

def build_tree(numlist):
    if len(numlist)==0:

```

基本信息

#: 44506073
 题目: 05455
 提交人: 尹柚鑫(2100015878)
 内存: 3688kB
 时间: 26ms
 语言: Python3
 提交时间: 2024-04-02 15:19:31

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。

耗时: 30mins

思路: 手搓最小二叉堆

```

#
class MinHeap:
    def __init__(self):
        self.heaplist = [0]
        self.currentSize = 0

    def insert(self, num):
        self.heaplist.append(num)
        self.currentSize += 1
        self.perUp(self.currentSize)

    def perUp(self, i):
        while i // 2 > 0:
            if self.heaplist[i] < self.heaplist[i // 2]:
                self.heaplist[i], self.heaplist[i // 2] = self.heaplist[i // 2],
self.heaplist[i]
                i = i // 2
            else:
                break

    def perDown(self, i):
        while 2 * i <= self.currentSize:
            mc = self.minChild(i)
            if self.heaplist[i] > self.heaplist[mc]:
                self.heaplist[i], self.heaplist[mc] = self.heaplist[mc],
self.heaplist[i]
                i = mc
            else:
                break

    def minChild(self, i):
        if 2 * i + 1 > self.currentSize:
            return 2 * i
        else:
            if self.heaplist[2 * i + 1] < self.heaplist[2 * i]:
                return 2 * i + 1
            else:
                return 2 * i

    def delMin(self):
        if self.currentSize == 0:
            return None
        output = self.heaplist[1]
        self.heaplist[1] = self.heaplist[self.currentSize]
        self.currentSize -= 1
        self.heaplist.pop()
        self.perDown(1)
        return output

def main():
    n = int(input())
    heap = MinHeap()
    for _ in range(n):

```

```

op, *args = map(int, input().split())
if op == 1:
    heap.insert(args[0])
elif op == 2:
    print(heap.delMin())

if __name__ == '__main__':
    main()

```

代码运行截图 == (AC代码截图, 至少包含有"Accepted") ==

#44506576提交状态

查看 提交 统计 提问

状态: Accepted

源代码

```

class MinHeap:
    def __init__(self):
        self.heaplist = [0]
        self.currentSize = 0

    def insert(self, num):
        self.heaplist.append(num)
        self.currentSize += 1
        self.perUp(self.currentSize)

    def perUp(self, i):
        while i // 2 > 0:
            if self.heaplist[i] < self.heaplist[i // 2]:

```

基本信息

#: 44506576
 题目: 04078
 提交人: 尹柚鑫(2100015878)
 内存: 4692kB
 时间: 676ms
 语言: Python3
 提交时间: 2024-04-02 16:16:17

22161: 哈夫曼编码树

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

思路:

利用上一题写的最小堆, 不断获取最小的数, 从而构建huffman树, 之后进行编码和解码 (这一段理解不太到位)

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) #note: 合并后, char 字段默认值是空
    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.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:
    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") ==

#44509990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

    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:

```

基本信息

#: 44509990
 题目: 22161
 提交人: 尹柚鑫(2100015878)
 内存: 3688kB
 时间: 25ms
 语言: Python3
 提交时间: 2024-04-02 20:47:55

晴问9.5: 平衡二叉树的建立

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

思路:

代码

```
#
```

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:

代码

```
#
```

代码运行截图 == (AC代码截图, 至少包含有"Accepted") ==

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。==

1.

for _ in range(n):

 op, *args = map(int, input().split())



ChatGPT

在Python中，星号`*`可以用来解包(iterable unpacking)。在你提供的代码中，`*args`用来接收输入的除第一个元素外的其他所有元素。这样做的好处是，不管输入中有多少个额外的参数，它们都会被收集到`args`列表中，方便后续处理。

让我来解释一下：

1. `input().split()` 用于将输入的字符串按空格分割成多个部分，然后返回一个列表。
2. `map(int, input().split())` 将这个列表中的每个元素转换为整数，返回一个整数迭代器。
3. `op, *args = map(int, input().split())` 将迭代器中的第一个元素赋值给`op`，而其他所有元素被解包并赋值给`args`列表。这意味着`args`包含了除了第一个元素外的其他所有元素。

这种语法在处理不定数量参数的情况下很有用，使得代码更加灵活和简洁。

这个在接受不定数量的参数时很有用