МІНІСТЕРСТВО ОСВІТИ I НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. Ігоря Сікорського»

ФАКУЛЬТЕТ БІОМЕДИЧНОЇ ІНЖЕНЕРІЇ

КАФЕДРА БІОМЕДИЧНОЇ КІБЕРНЕТИКИ

**Звіт до домашньої контрольної роботи**

з дисципліни «Проектування та аналіз обчислювальних алгоритмів»

на тему: «Програмна реалізація розкладу функції»

**Виконав:**

студент групи БС-81

Гладкий Я.В.

**Перевірив:**

Доц. каф. БМК, к.т.н.

Яковенко А.В.

Зараховано від \_\_.\_\_.\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис викладача)

Київ-2019

**Мета роботи:** робота з бінарними деревами та бінарними деревами пошуку.

**Завдання:**

1. На вхід подається деяке бінарне дерево, із фіксованою структурою (тобто зв’язками між вузлами, їх батьком та нащадками). Необхідно переписати значення вузлів дерева таким чином, щоб:

а) їх нові значення брались тільки з того набору, який присутній у вхідному дереві;

б) зберігалась внутрішня структура дерева (зберігались зв’язки між кожним батьківським вузлом та його вузлами нащадками).

2. Після того, як вхідне дерево перетворене на бінарне дерево пошуку, необхідно в отриманому бінарному дереві пошуку знайти всі такі монотонні шляхи (які не обов'язково йдуть від кореня, але всі прямують згори донизу), що сума значень вузлів, які належать знайденим шляхам, дорівнює числу S.

**Лістинг програми:**

class Node:

def \_\_init\_\_(self, val, space=1, l=None, r=None):

self.left = l

self.right = r

self.space=space

self.v = val

if self.v == 0:

self.space=0

def display(self):

lines, \_, \_, \_ = self.\_display\_aux()

for line in lines:

print(line)

def \_display\_aux(self):

"""Returns list of strings, width, height, and horizontal coordinate of the root."""

# No child.

if self.right is None and self.left is None:

line = '%s' % self.v

width = len(line)

height = 1

middle = width // 2

return [line], width, height, middle

# Only left child.

if self.right is None:

lines, n, p, x = self.left.\_display\_aux()

s = '%s' % self.v

u = len(s)

first\_line = (x + 1) \* ' ' + (n - x - 1) \* '\_' + s

second\_line = x \* ' ' + '/' + (n - x - 1 + u) \* ' '

shifted\_lines = [line + u \* ' ' for line in lines]

return [first\_line, second\_line] + shifted\_lines, n + u, p + 2, n + u // 2

# Only right child.

if self.left is None:

lines, n, p, x = self.right.\_display\_aux()

s = '%s' % self.v

u = len(s)

first\_line = s + x \* '\_' + (n - x) \* ' '

second\_line = (u + x) \* ' ' + '\\' + (n - x - 1) \* ' '

shifted\_lines = [u \* ' ' + line for line in lines]

return [first\_line, second\_line] + shifted\_lines, n + u, p + 2, u // 2

# Two children.

left, n, p, x = self.left.\_display\_aux()

right, m, q, y = self.right.\_display\_aux()

s = '%s' % self.v

u = len(s)

first\_line = (x + 1) \* ' ' + (n - x - 1) \* '\_' + s + y \* '\_' + (m - y) \* ' '

second\_line = x \* ' ' + '/' + (n - x - 1 + u + y) \* ' ' + '\\' + (m - y - 1) \* ' '

if p < q:

left += [n \* ' '] \* (q - p)

elif q < p:

right += [m \* ' '] \* (p - q)

zipped\_lines = zip(left, right)

lines = [first\_line, second\_line] + [a + u \* ' ' + b for a, b in zipped\_lines]

return lines, n + m + u, max(p, q) + 2, n + u // 2

class Tree:

def \_\_init\_\_(self):

self.root = None

def getroot(self):

print(self.root.v)

def add(self, val):

if(self.root == None):

self.root = Node(val)

else:

self.\_add(val, self.root)

def \_add(self, val, node):

if node.left==None:

node.left=Node(val)

elif node.left.space==0 and node.right==None:

node.right=Node(val)

else:

if node.left.space:

self.\_add(val, node.left)

elif node.right.space:

self.\_add(val, node.right)

self.isspace(node)

def newadd(self, array):

self.\_newadd(array, self.root)

def \_newadd(self, array, node):

if node.v!=0:

self.\_newadd(array, node.left)

node.v=array[0]

del array[0]

self.\_newadd(array, node.right)

def isspace(self, node):

if node.left!=None and node.right!=None:

if node.left.space==0 and node.right.space==0:

node.space=0

def clean(self):

if(self.root != None):

self.\_clean(self.root)

def \_clean(self, node):

if(node != None):

if node.left!=None and node.left.v==0:

node.left=None

if node.right!=None and node.right.v==0:

node.right=None

self.\_clean(node.left)

self.\_clean(node.right)

def pathsum(self, s):

return self.path\_sum(self.root, s)

def path\_sum(self, node, s, l=[], finalans=[]):

if (sum(l)+node.v)==s:

l.append(node.v)

if l not in finalans:

finalans+=[l]

if node.left!=None:

self.path\_sum(node.left, s, [])

else:

l.append(node.v)

l2=l.copy()

if node.left != None:

self.path\_sum(node.left, s, l, finalans)

self.path\_sum(node.left, s, [], finalans)

if node.right != None:

self.path\_sum(node.right, s, l2, finalans)

self.path\_sum(node.right, s, [], finalans)

return finalans

with open("input\_10b.txt") as f:

file=f.read()

file = list(map(int, file.split()))

NewTree = Tree()

for i in file:

NewTree.add(i)

print('INPUT:')

NewTree.root.display()

l=[]

for i in file:

if i!=0:

l.append(i)

l.sort()

NewTree.newadd(l)

NewTree.clean()

print('\nOUTPUT:')

NewTree.root.display()

wanted=(int(input('\nWANTED PATHSUM: ')))

pathsums = NewTree.pathsum(wanted)

if len(pathsums)==0:

print ("THERE IS NO WAY TO GET THIS SUM.")

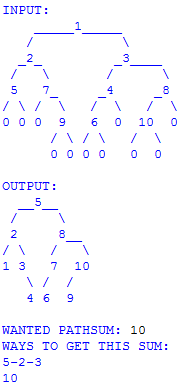
else:

print ("WAYS TO GET THIS SUM:")

for i in pathsums:

i=list(map(str, i))

print('-'.join(i))

**Результат роботи:**