Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Владимирский государственный университет

имени Александра Григорьевича и Николая Григорьевича Столетовых»

(ВлГУ)

Колледж инновационных технологий и предпринимательства

**Лабораторная работа №2**

на тему: «Обходы деревья»

Вариант 21

Выполнила:

ст.гр. ИРсп-121

Орлова. С. В.

Проверил:

Ларин Е. С.

Владимир 2023

**Цель работы**

Изучить методы построения бинарных деревьев, методы графического представления деревьев. Изучить методы обхода деревьев.

**Задание для выполнения:**

1. Организовать ввод данных для построения дерева поиска из текстового файла.
2. Написать метод отображения дерева на экране.
3. Реализовать метод DFS. Вывести на экран список обхода вершин методом DFS любой модификации.
4. С помощью метода DFS определить и вывести на экран списки вершин каждой ветви и длину каждой ветви.
5. Вершины самой длинной ветви закрасить красным цветом.
6. С помощью метода BFS определить и вывести на экран списки вершин каждого уровня дерева и ширину каждого уровня.
7. Вершины самого широкого уровня закрасить синим цветом.
8. \* Написать метод отображения дерева таким образом, чтобы вершины не закрывали друг друга.

**Ход работы**

Листинг кода:

from random import \*

from PIL import Image, ImageDraw

class Node:

def \_\_init\_\_(self, value: int, name: int):

self.data: int = value

self.left: Node | None = None

self.right: Node | None = None

self.id: int = name

self.color = "white"

class BinTree:

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

self.root: Node | None = None

self.count: int = 0

def \_\_str\_\_(self):

if self.root is not None:

return " ".join(self.\_get\_tree(self.root))

return ''

def insert(self, val: int, name: int):

if self.root is not None:

self.\_insert(val, self.root, name)

else:

self.root = Node(val, name)

self.count += 1

def paint\_tree(self):

img = Image.new("RGB", (1000, 1000), "white")

draw = ImageDraw.Draw(img)

if self.root is not None:

self.\_drawTree(self.root,500,25,250,50,draw)

return img

def deleteTree(self, node: Node):

if node is not None:

self.deleteTree(node.left)

self.deleteTree(node.right)

del node

# private

def \_insert(self, value: int, node: Node, name: int):

if value < node.data:

if node.left is not None:

self.\_insert(value, node.left, name)

else:

node.left = Node(value, name)

else:

if node.right is not None:

self.\_insert(value, node.right, name)

else:

node.right = Node(value, name)

def \_drawTree(self,node,x,y,dx,dy,draw):

node.x = x

node.y = y

if node.left is not None :

newx,newy = x-dx,y+dy

draw.line([(x,y),(newx,newy)],width=2,fill = 'black')

self.\_drawTree(node.left,newx,newy,dx//2,dy,draw)

if node.right is not None:

newx,newy = x+dx,y+dy

draw.line([(x,y),(newx,newy)],width=2,fill = 'black')

self.\_drawTree(node.right,newx,newy,dx//2,dy,draw)

draw.ellipse( (x-25,y-25,x+25,y+25),fill=node.color, outline ='black' )

if node.data < 10 :

draw.text((x-7,y-7),str(node.data),(0,0,0))

else:

draw.text((x-10,y-7),str(node.data),(0,0,0))

def \_get\_tree(self, node: Node):

if node:

return self.\_get\_tree(node.left) + [str(node.data)] \

+ self.\_get\_tree(node.right)

return []

def tree\_depth(self, node: Node, height=0):

if node:

height += 1

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

if self.height < height:

self.height = height

self.tree\_depth(node.left, height)

self.tree\_depth(node.right, height)

def dfs(self, node: Node):

if node :

print(node.data, end =' ')

self.dfs(node.left)

self.dfs(node.right)

def dfs\_branch\_left(self, node: Node, nodes=[], flag=True):

if flag:

if node.left:

flag = False

self.dfs\_branch\_left(node.left, nodes, flag)

else:

if node.left:

self.dfs\_branch\_left(node.left, nodes, flag)

if node.right:

self.dfs\_branch\_left(node.right, nodes, flag)

nodes.append(node.data)

def dfs\_branch\_right(self, node: Node, nodes=[], flag=True):

if flag:

if node.right:

flag = False

self.dfs\_branch\_right(node.right, nodes, flag)

else:

if node.right:

self.dfs\_branch\_right(node.right, nodes, flag)

if node.left:

self.dfs\_branch\_right(node.left, nodes, flag)

nodes.append(node.data)

def longest\_to\_red(self, node: Node):

def find\_longest(node, branch=[]):

if node is None:

return branch

branch.append(node)

left\_branch = find\_longest(node.left, branch.copy())

right\_branch = find\_longest(node.right, branch.copy())

return left\_branch if len(left\_branch) > len(right\_branch) else right\_branch

longest\_branch = find\_longest(node)

for node in longest\_branch:

node.color = "red"

def bfs(self, node: Node):

def \_bfs(node):

result = []

if not node:

return result

queue = [node]

while queue:

level = []

next\_level = []

for n in queue:

level.append(n.data)

if n.left:

next\_level.append(n.left)

if n.right:

next\_level.append(n.right)

result.append(level)

queue = next\_level

print("BFS:")

for i, level in enumerate(result):

print(f"Вершины стоки {i+1}: {level} (Ширина: {len(level)})")

\_bfs(node)

def max\_width\_color(self, node: Node):

def calc(node, level=0, level\_width={}):

if node is not None:

if level not in level\_width:

level\_width[level] = 0

level\_width[level] += 1

calc(node.left, level + 1, level\_width)

calc(node.right, level + 1, level\_width)

level\_width = {}

calc(node, 0, level\_width)

max\_width = max(level\_width.values())

for level, width in level\_width.items():

if width == max\_width:

self.\_color\_level(node, level, "aqua")

def \_color\_level(self, node: Node, level, color):

if node is not None:

if level == 0:

node.color = color

self.\_color\_level(node.left, level-1, color)

self.\_color\_level(node.right, level-1, color)

def get\_from\_file(filename):

file = open(filename, 'r', encoding='utf-8')

lines = file.read().splitlines()

nums = []

for line in lines:

nums += list(map(int, line.split(' ')))

file.close()

return nums

def main():

tree = BinTree()

nums = get\_from\_file(r'F:\(((\3\трпо\Lb2.txt')

for i in range(len(nums)):

tree.insert(nums[i], i)

tree.longest\_to\_red(tree.root)

tree.max\_width\_color(tree.root)

img1 = tree.paint\_tree()

img1.show()

print("\nDFS")

tree.dfs(tree.root)

print()

stack = []

tree.dfs\_branch\_left(tree.root, stack)

print(f'Левая: {stack}\nГлубина левой ветки: {len(stack)}')

stack = []

tree.dfs\_branch\_right(tree.root, stack)

print(f'Правая: {stack}\nГлубина правой ветки: {len(stack)}')

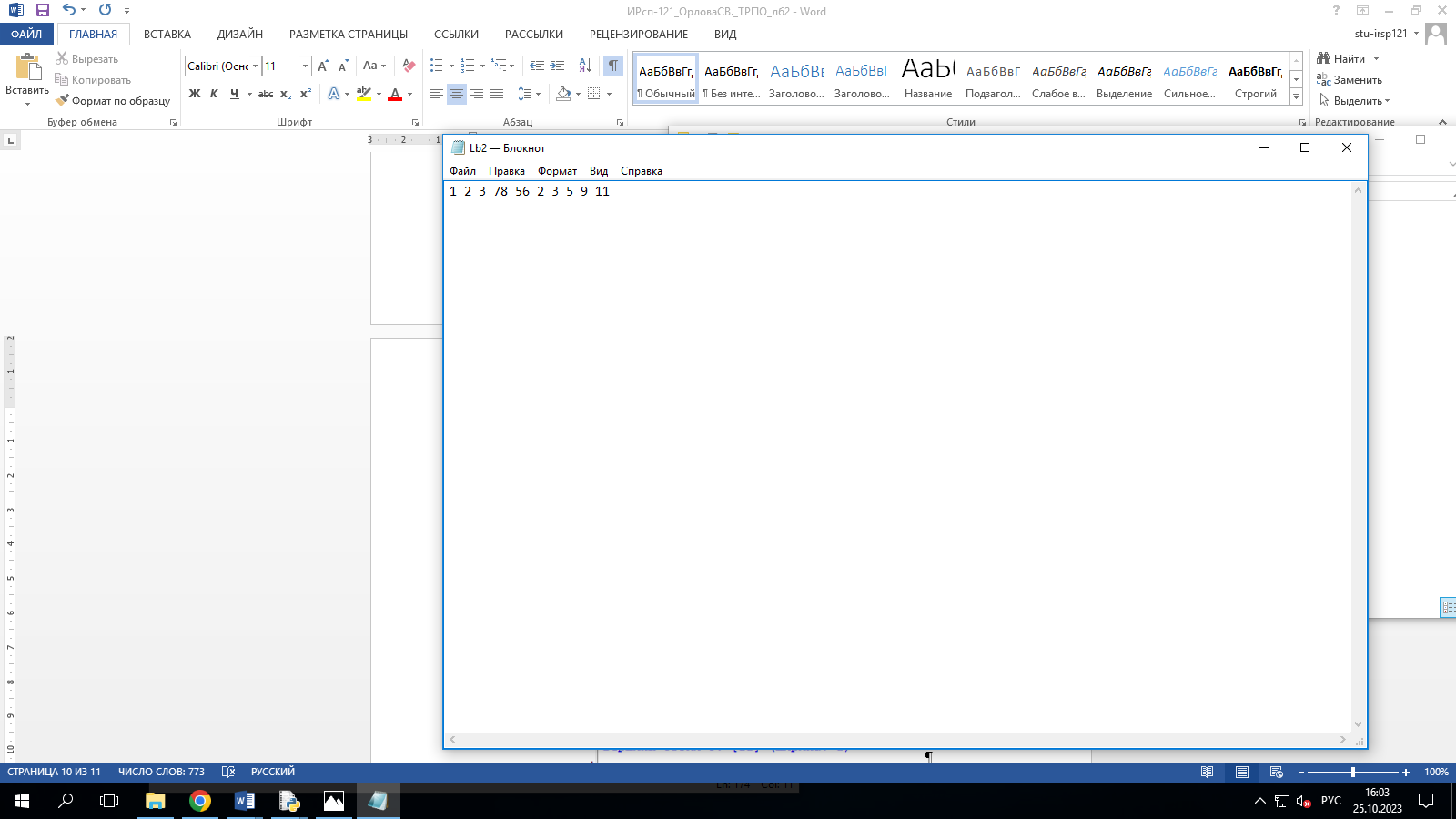
print('\n')

tree.bfs(tree.root)

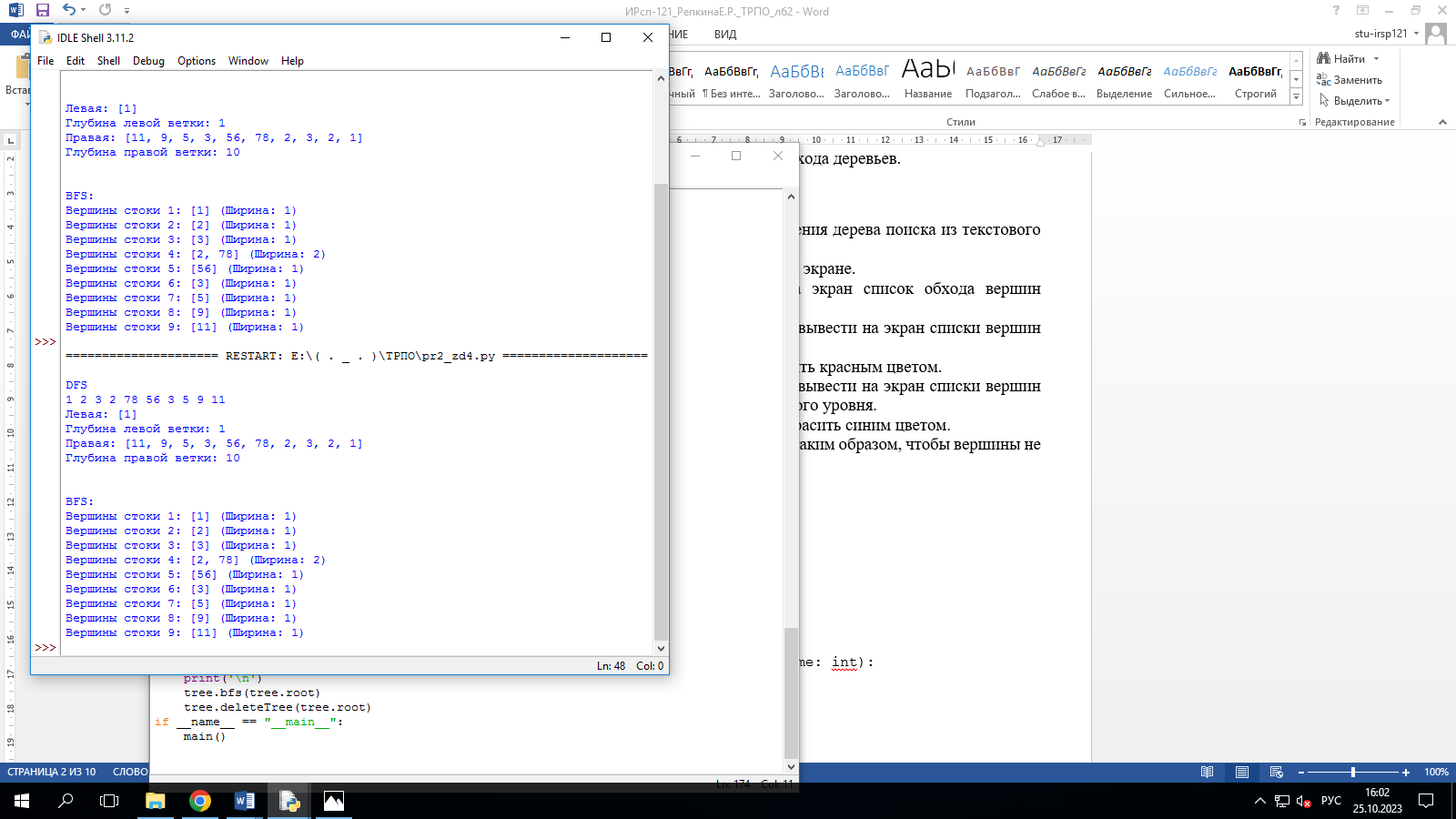
tree.deleteTree(tree.root)

if \_\_name\_\_ == "\_\_main\_\_":

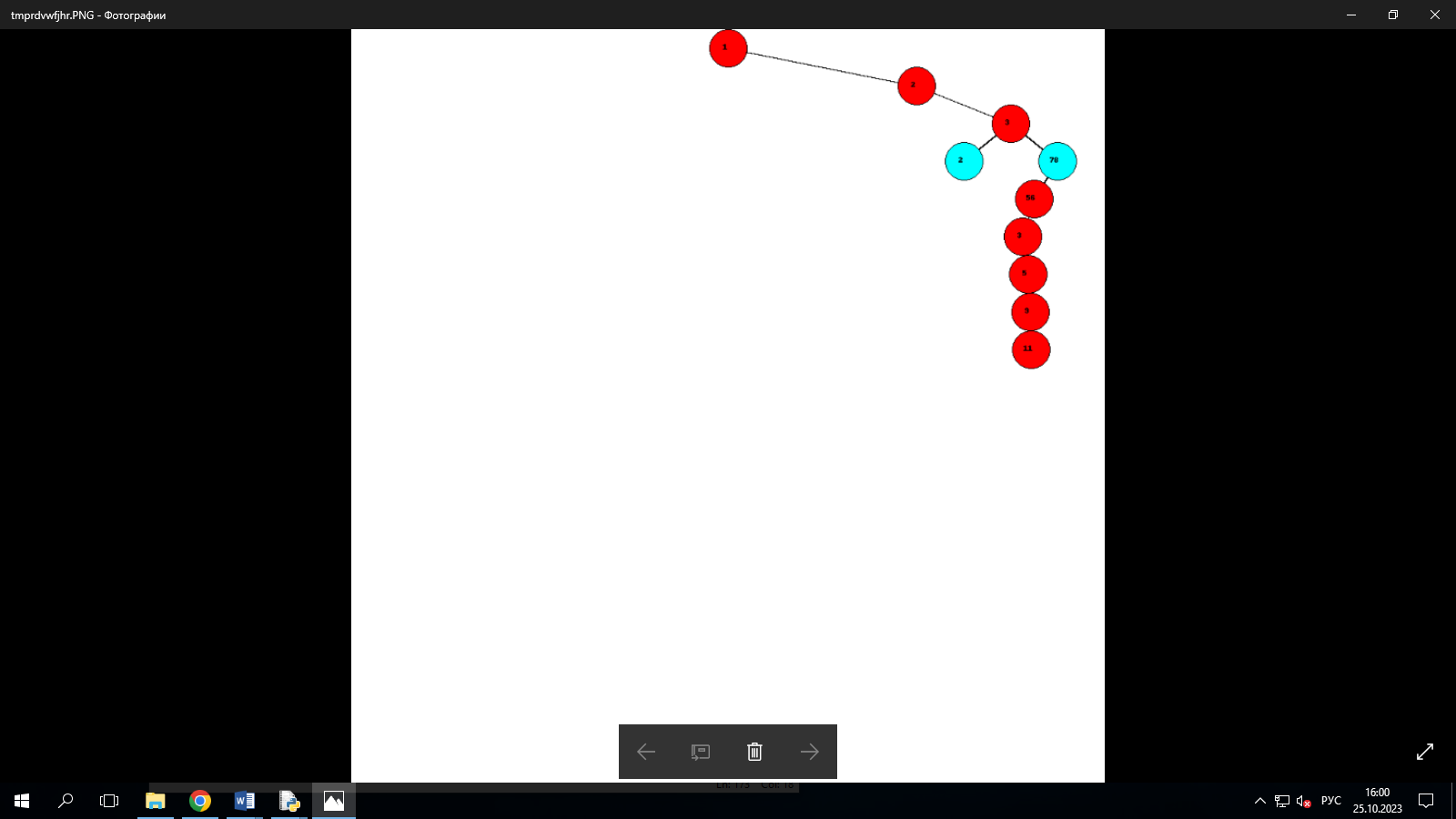
main()



*Рисунок 1 – Информация в тестовом файле «Lb2.txt»*



*Рисунок 2 – Результат*



*Рисунок 3 - Результат*

**Вывод**

В ходе данной лабораторной работы изучены методы построения бинарных деревьев, методы графического представления деревьев, также изучены методы обхода деревьев.