МІНІСТЕРСТВО ОСВІТИ І НАУКИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМ.ІГОРЯ СІКОРСЬКОГО»

НАВЧАЛЬНО-НАУКОВИЙ ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ

Лабораторна робота №5

«Дерева»

Варіант 2

Виконав:
Студент 2 курсу
Групи ФІ-21
Голуб Михайло

Перевірив:
Лавренюк А. М.

Київ 2024

ЗАВДАННЯ

Побудувати двійкове дерево пошуку з цілих чисел, що вводяться. Вивести його на екран у вигляді дерева. Знайти вершину, яка містить задане число. Визначити максимальний елемент в цьому дереві.

ХІД РОБОТИ

Створимо клас Node, який має методи рекурсивні методи depth, append та find:

```
class Node:
    def __init__(self, value, parent=None, right_arm=None,
left_arm=None):
        self.value = value
        self.parent = parent
        self.right_arm = right_arm
        self.left_arm = left_arm

    def __str__(self):
        return str(self.value)

    def depth(self):
        if self.parent is None:
            return 0
        else:
            return self.parent.depth() + 1

    def append(self, value):
        if self.value >= value:
            if self.left_arm is None:
                node = Node(value, self)
                self.left_arm = node
                return
            else:
                self.left_arm.append(value)
        else:
            if self.right_arm is None:
                node = Node(value, self)
                self.right_arm = node
                return
            else:
                self.right_arm.append(value)
                return

    def find(self, key):
        if key < self.value:
            if self.left_arm is None:
                return 0
            else:
                return self.left_arm.find(key)
        elif key > self.value:
            if self.right_arm is None:
                return 0
            else:
                return self.right_arm.find(key)
        else:
            return self
```

Створимо клас Tree, який має методи find, draw, append, find_max:

```
class Tree:
    def __init__(self, root):
        if isinstance(root, Node):
            self.root = root
        else:
            self.root = Node(root)

    def find(self, key):
        return self.root.find(key)

    def draw(self, x_spacing=25, y_spacing=30):
        turtle.speed(0)
        previous_level = []
        level = [self.root]
        level_counter = 0
        while True:
            # draw level
            turtle.penup()
            turtle.pencolor("black")
            turtle.goto(-x_spacing * (len(level) - 1) / 2, -
level_counter * y_spacing)
            for node in level:
                if not node is None:
                    turtle.write(node.value, align="center",
font=("Verdana", 15, "normal"))
                turtle.setheading(0)
                turtle.forward(x_spacing)

            # draw connections
            for node in level:
                if not node is None:
                    if not node.parent is None:
                        turtle.penup()
                        turtle.goto(-x_spacing * (len(level) - 1)
/ 2 + x_spacing * level.index(node),
                                    -level_counter * y_spacing)
                        turtle.pendown()
                        if node.parent.left_arm == node:
                            turtle.pencolor("red")
                        else:
                            turtle.pencolor("black")
                        turtle.goto(
                            -x_spacing * (len(previous_level) - 1)
/ 2 + x_spacing * previous_level.index(node.parent),
                            -level_counter * y_spacing +
y_spacing)

            # iterate
            level_counter += 1
            nodes_exist = False
            new_level = []
            for node in level:
                if not node is None:
```

```
                    new_level.append(node.left_arm)
                    new_level.append(node.right_arm)
                    nodes_exist = True
                else:
                    new_level.append(None)
                    new_level.append(None)
            previous_level = copy.copy(level)
            level = copy.copy(new_level)
            if not nodes_exist:
                return 0

    def append(self, value):
        self.root.append(value)

    def find_max(self):
        nodes = [self.root]
        values = []
        nodes_detected = True
        while nodes_detected:
            new_nodes = []
            nodes_detected = False
            for node in nodes:
                if not node is None:
                    nodes_detected = True
                    values.append(node.value)
                    new_nodes.append(node.right_arm)
                    new_nodes.append(node.left_arm)
            nodes = copy.copy(new_nodes)
        maximum = max(values)
        print(maximum)
        return self.find(maximum)
```

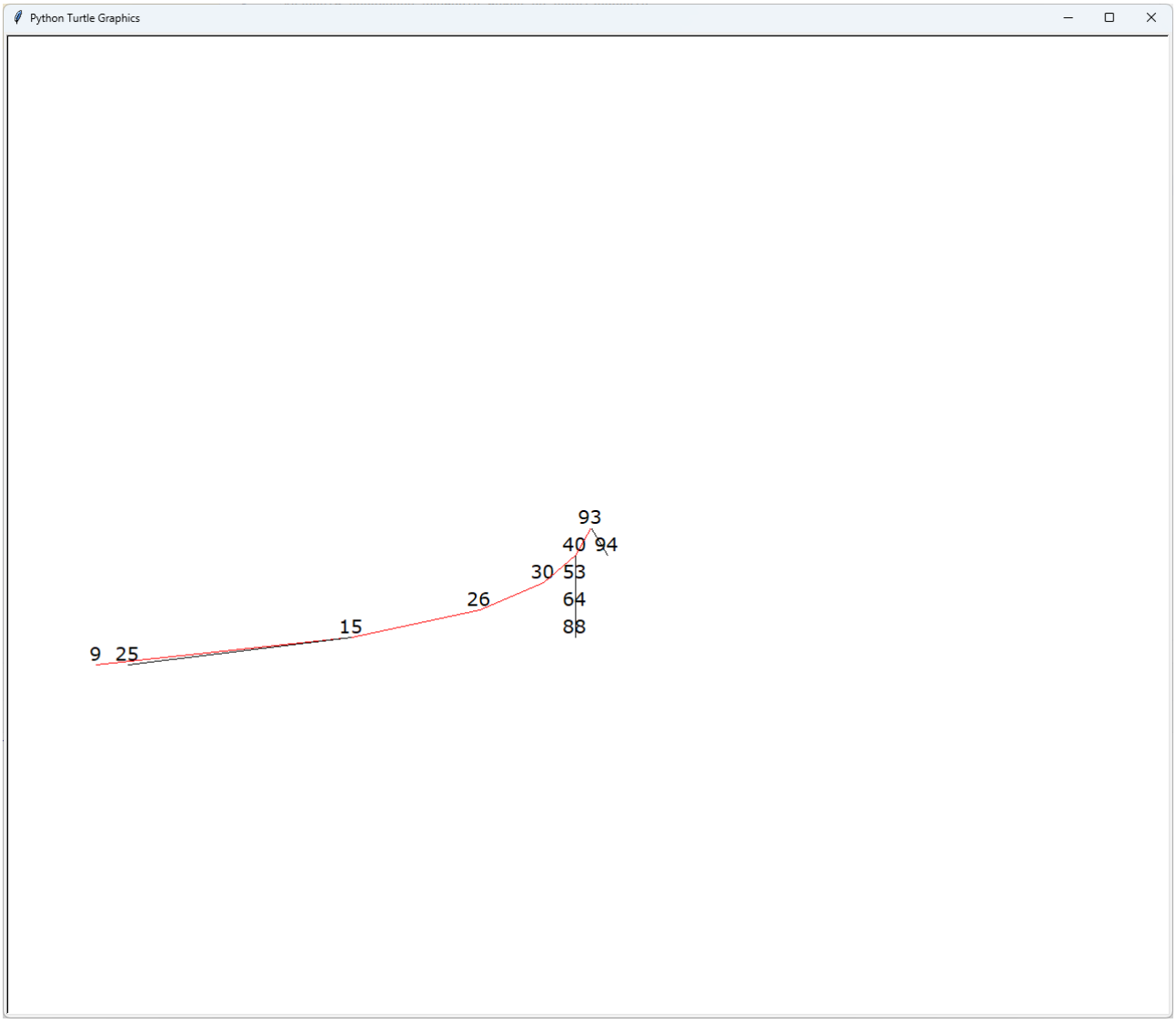
Створимо функцію tree_random, яка випадковим чином генерує дерево із заданою кількістю вершин:

```
def random_tree(nodes_n, min=0, max=99):
    width = max - min
    tree = Tree(int(random() * width + min))
    for n in range(nodes_n):
        value = int(random() * width + min)
        tree.append(value)
    return tree
```

Отже, створюємо випадкове дерево з 10 вершин та виводимо значення його найбільшого елемента:

```
test = random_tree(10)
print(test.find_max())
test.draw(35)
input()
```

Приклад відображення дерева (ліві зв’язки червоним, праві зв’язки чорним):



Результат пошуку максимального елемента в дереві: 94