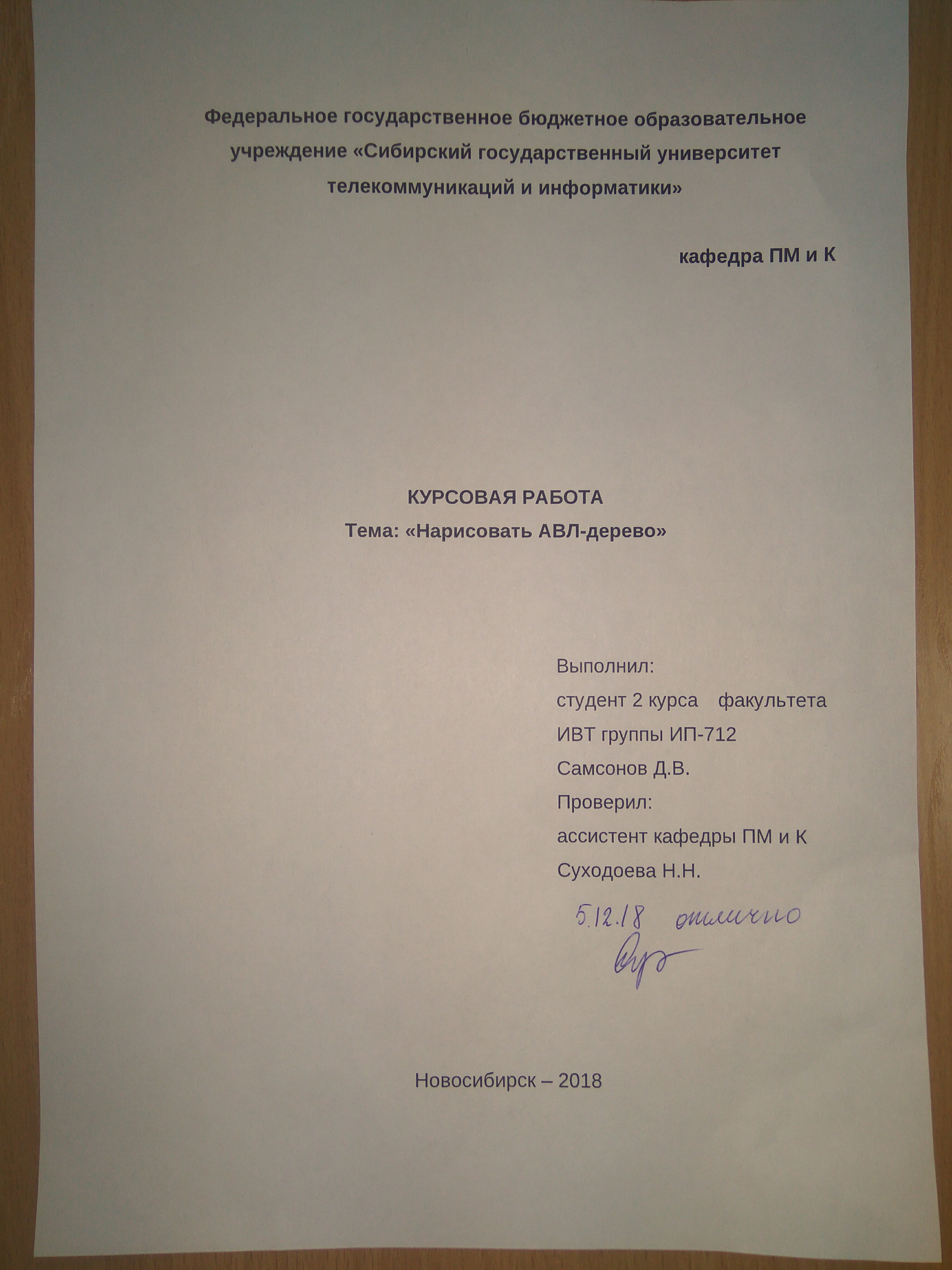
****

**Федеральное государственное бюджетное образовательное**

**учреждение «Сибирский государственный университет**

**телекоммуникаций и информатики»**

**кафедра ПМ и К**

**КУРСОВАЯ РАБОТА**

**Тема: «Нарисовать АВЛ-дерево»**

Выполнил:

студент 2 курса факультета ИВТ группы ИП-712

Самсонов Д.В.

Проверил:

ассистент кафедры ПМ и К

Суходоева Н.Н.

Новосибирск – 2018

Оглавление

[Постановка задачи 2](#_Toc531290180)

[Технологии ООП 3](#_Toc531290181)

[Структура классов 4](#_Toc531290182)

[Алгоритм и результат работы 6](#_Toc531290183)

[Заключение 7](#_Toc531290184)

[Приложение. Листинг 8](#_Toc531290185)

## Постановка задачи

Необходимо написать программу, используя объектно-ориентированный подход, которая будет реализовать отображение графического представления АВЛ-дерева.

Программа написана на языке программирования Python3 при помощи графической библиотеки Tkinter.

# Технологии ООП

В курсовой работе были применены такие принципы объектно-ориентированного программирования, как:

1. Инкапсуляция
2. Полиморфизм
3. Наследование

Из технологий ООП были применены:

1. Абстрактный класс
2. Абстрактные(виртуальные) методы
3. Статические методы
4. Инициализатор (конструктор) с параметрами по умолчанию

Дополнительные использованные технологии:

1. Свойство (@property)
2. Замыкания

### Структура классов

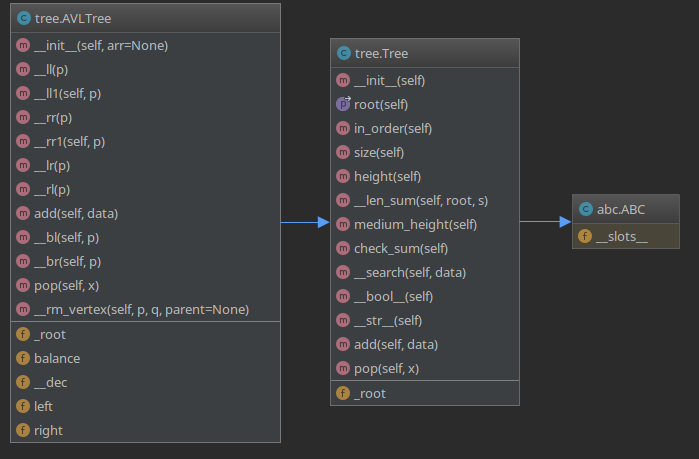


Рисунок 1. Структура классов Tree, AVLTree

В программе реализован абстрактный класс (*Tree*), (рис. 1) который содержит в себе 2 абстрактных метода (add и pop) и множество общих для всех деревьев методов, а также защищённое поле root.

Этот класс является родительским для класса *AVLTree*, который содержит в себе методы, присущие именно АВЛ-дереву и определение всех абстрактных методов.

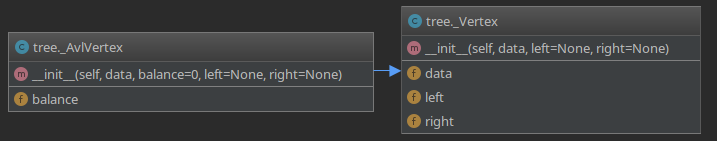
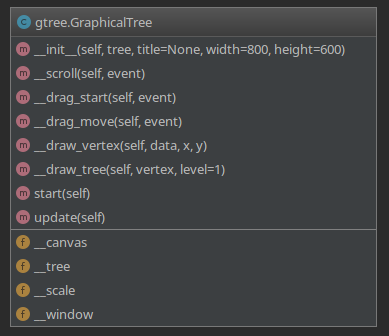


Рисунок 2. Структура классов Vertex и AvlVertex

В качестве вершин для дерева используются объекты класса AvlVertex, (рис. 2) который унаследован от класса Vertex. В классе AvlVertex определено поле balance, которое необходимо для АВЛ-дерева.

Рисунок 3. Структура класса GraphicalTree

Для графического отображения дерева используется класс GraphicalTree, (рис. 3) который содержит методы для отрисовки дерева.

# Алгоритм и результат работы

В программе используются 4 класса, описанные выше.

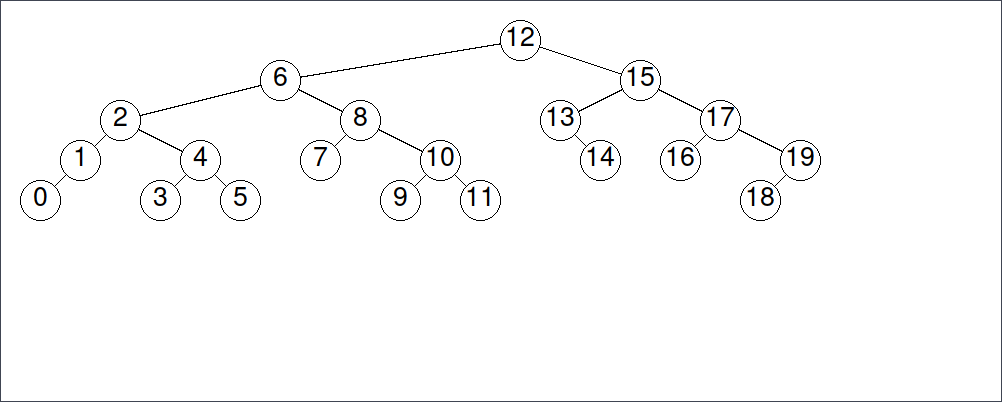
В основной части программы АВЛ-дерево заполняется случайными не повторяющимися числами при помощи методов, определённых в классе AVLTree, затем создаётся объект класса GraphicalTree и вызывается метод start, который запускает окно с деревом (рис. 4).

Рисунок 4. Результат работы программы

Дерево можно масштабировать при помощи колёсика мыши и перемещать левой кнопкой мыши при помощи приватных методов scroll, drag\_start, drag\_move.

# Заключение

При работе использовались такие принципы ООП, как наследование, полиморфизм и инкапсуляция. Наследование – конструирование новых, более сложных производных классов-потомков, из уже имеющихся базовых классов-родителей, с помощью добавления новых полей и/или методов. Полиморфизм - механизм, позволяющий функциям обрабатывать разные типы данных, которые объединены общим классом родителем. Инкапсуляция – объединение данных и методов, обрабатывающих эти данные в одном классе, а также сокрытие данных от прямого воздействия на них за пределами класса.

Для создания объектов классов в программе существуют конструкторы.

Для создания абстрактного класса использовалось наследование от AbstractBaseClass, а для создания абстрактных методов использовались декораторы @abstractmethod.

Таким образом было реализовано отображение АВЛ-дерева с использованием объектно-ориентированных технологий.

# Приложение. Листинг

Листинг кода из файла tree.py:

from abc import ABC, abstractmethod  
  
  
class \_Vertex:  
 def \_\_init\_\_(self, data, left=None, right=None):  
 self.data = data  
 self.left = left  
 self.right = right  
  
  
class \_AvlVertex(\_Vertex):  
 def \_\_init\_\_(self, data, balance=0, left=None, right=None):  
 super().\_\_init\_\_(data, left, right)  
 self.balance = balance  
  
  
class Tree(ABC):  
 def \_\_init\_\_(self):  
 self.\_root = None  
  
 @property  
 def root(self):  
 return self.\_root  
  
 def in\_order(self):  
 def in\_order(root):  
 array = []  
 if root:  
 array += in\_order(root.left)  
 array.append(root.data)  
 array += in\_order(root.right)  
 return array  
  
 return in\_order(self.\_root)  
  
 def size(self):  
 def size(root):  
 return 1 + size(root.left) + size(root.right) if root else 0  
  
 return size(self.\_root)  
  
 def height(self):  
 def height(root):  
 return 1 + max(height(root.left), height(root.right)) if root else 0  
  
 return height(self.\_root)  
  
 def \_\_len\_sum(self, root, s):  
 return s + self.\_\_len\_sum(root.left, s + 1) + self.\_\_len\_sum(root.right, s + 1) if root else 0  
  
 def medium\_height(self):  
 return self.\_\_len\_sum(self.\_root, 1) / self.size()  
  
 def check\_sum(self):  
 def check\_sum(root):  
 return root.data + check\_sum(root.left) + check\_sum(root.right) if root else 0  
  
 return check\_sum(self.\_root)  
  
 def \_\_search(self, data):  
 p = self.\_root  
 while p:  
 if data < p.data:  
 p = p.left  
 elif data > p.data:  
 p = p.right  
 else:  
 break  
 return p  
  
 def \_\_bool\_\_(self):  
 return bool(self.\_root)  
  
 def \_\_str\_\_(self):  
 return str(self.in\_order())  
  
 @abstractmethod  
 def add(self, data):  
 pass  
  
 @abstractmethod  
 def pop(self, x):  
 pass  
  
  
class AVLTree(Tree):  
 def \_\_init\_\_(self, arr=None):  
 super().\_\_init\_\_()  
 self.\_\_dec = False  
 if arr:  
 for element in arr:  
 self.add(element)  
  
 @staticmethod  
 def \_\_ll(p):  
 q = p.left  
 p.balance = q.balance = 0  
 p.left = q.right  
 q.right = p  
 return q  
  
 def \_\_ll1(self, p):  
 q = p.left  
 if q.balance == 0:  
 q.balance = 1  
 p.balance = -1  
 self.\_\_dec = False  
 else:  
 q.balance = p.balance = 0  
 p.left = q.right  
 q.right = p  
 return q  
  
 @staticmethod  
 def \_\_rr(p):  
 q = p.right  
 p.balance = q.balance = 0  
 p.right = q.left  
 q.left = p  
 return q  
  
 def \_\_rr1(self, p):  
 q = p.right  
 if q.balance == 0:  
 q.balance = -1  
 p.balance = 1  
 self.\_\_dec = False  
 else:  
 q.balance = p.balance = 0  
 p.right = q.left  
 q.left = p  
 return q  
  
 @staticmethod  
 def \_\_lr(p):  
 q = p.left  
 r = q.right  
 if r.balance < 0:  
 p.balance = 1  
 else:  
 p.balance = 0  
 if r.balance > 0:  
 q.balance = -1  
 else:  
 q.balance = 0  
 r.balance = 0  
 q.right = r.left  
 p.left = r.right  
 r.left = q  
 r.right = p  
 return r  
  
 @staticmethod  
 def \_\_rl(p):  
 q = p.right  
 r = q.left  
 if r.balance > 0:  
 p.balance = -1  
 else:  
 p.balance = 0  
 if r.balance < 0:  
 q.balance = 1  
 else:  
 q.balance = 0  
 r.balance = 0  
 q.left = r.right  
 p.right = r.left  
 r.right = q  
 r.left = p  
 return r  
  
 def add(self, data):  
 def add(root):  
 if root is None:  
 self.\_\_grow = True  
 return \_AvlVertex(data)  
 elif root.data > data:  
 root.left = add(root.left)  
 if self.\_\_grow:  
 if root.balance > 0:  
 root.balance = 0  
 self.\_\_grow = False  
 elif root.balance == 0:  
 root.balance = -1  
 self.\_\_grow = True  
 else:  
 if root.left.balance < 0:  
 root = self.\_\_ll(root)  
 self.\_\_grow = False  
 else:  
 root = self.\_\_lr(root)  
 self.\_\_grow = False  
 elif root.data < data:  
 root.right = add(root.right)  
 if self.\_\_grow:  
 if root.balance < 0:  
 root.balance = 0  
 self.\_\_grow = False  
 elif root.balance == 0:  
 root.balance = 1  
 self.\_\_grow = True  
 else:  
 if root.right.balance > 0:  
 root = self.\_\_rr(root)  
 self.\_\_grow = False  
 else:  
 root = self.\_\_rl(root)  
 self.\_\_grow = False  
 else:  
 print("Данные уже в дереве")  
 return root  
  
 self.\_root = add(self.\_root)  
  
 def \_\_bl(self, p):  
 if p.balance == -1:  
 p.balance = 0  
 return p  
 elif p.balance == 0:  
 p.balance = 1  
 self.\_\_dec = False  
 return p  
 elif p.balance == 1:  
 if p.right.balance >= 0:  
 return self.\_\_rr1(p)  
 else:  
 return self.\_\_rl(p)  
  
 def \_\_br(self, p):  
 if p.balance == 1:  
 p.balance = 0  
 return p  
 elif p.balance == 0:  
 p.balance = -1  
 self.\_\_dec = False  
 return p  
 elif p.balance == -1:  
 if p.left.balance <= 0:  
 return self.\_\_ll1(p)  
 else:  
 return self.\_\_lr(p)  
  
 def pop(self, x):  
 def pop(p, parent):  
 if p is None:  
 print("Вершины нет")  
 elif x < p.data:  
 p.left = pop(p.left, p)  
 if self.\_\_dec:  
 p = self.\_\_bl(p)  
 elif x > p.data:  
 p.right = pop(p.right, p)  
 if self.\_\_dec:  
 p = self.\_\_br(p)  
 else:  
 q = p  
 if q.left is None:  
 p = q.right  
 self.\_\_dec = True  
 elif q.right is None:  
 p = q.left  
 self.\_\_dec = True  
 else:  
 if x > parent.data:  
 parent.right = self.\_\_rm\_vertex(q.left, q)  
 elif x < parent.data:  
 parent.left = self.\_\_rm\_vertex(q.left, q)  
 else:  
 tmp = self.\_\_rm\_vertex(q.left, q)  
 if self.\_root is tmp:  
 self.\_root = tmp  
 else:  
 parent.left = tmp  
  
 if self.\_\_dec:  
 p = self.\_\_bl(p)  
 return p  
  
 self.\_root = pop(self.\_root, self.\_root)  
  
 def \_\_rm\_vertex(self, p, q, parent=None):  
 if p.right:  
 self.\_\_rm\_vertex(p.right, q, p)  
 if self.\_\_dec:  
 return self.\_\_br(p)  
 else:  
 q.data = p.data  
 self.\_\_dec = True  
 if parent:  
 parent.right = None  
 else:  
 q.left = p.left  
 return q

Листинг кода из файла gtree.py:

import tkinter as tk  
  
  
class GraphicalTree:  
 def \_\_init\_\_(self, tree, title=None, width=800, height=600):  
 self.\_\_tree = tree  
 self.\_\_window = tk.Tk()  
 self.\_\_window.title(title)  
 self.\_\_canvas = tk.Canvas(self.\_\_window, width=width, height=height)  
 self.\_\_canvas.configure(background="white")  
 self.\_\_canvas.pack()  
 self.\_\_canvas.bind("<Button-4>", self.\_\_scroll)  
 self.\_\_canvas.bind("<Button-5>", self.\_\_scroll)  
 self.\_\_canvas.bind("<ButtonPress-1>", self.\_\_drag\_start)  
 self.\_\_canvas.bind("<B1-Motion>", self.\_\_drag\_move)  
 self.\_\_scale = 1  
  
 def \_\_scroll(self, event):  
 if event.num == 4 or event.delta == 120:  
 self.\_\_canvas.scale("all", self.\_\_canvas.canvasx(event.x), self.\_\_canvas.canvasy(event.y), 1.1, 1.1)  
 self.\_\_scale \*= 1.1  
 elif event.num == 5 or event.delta == -120:  
 self.\_\_canvas.scale("all", self.\_\_canvas.canvasx(event.x), self.\_\_canvas.canvasy(event.y), 0.9, 0.9)  
 self.\_\_scale \*= 0.9  
 self.\_\_canvas.itemconfigure("text\_data", font=(None, round(self.\_\_scale \* 20)))  
  
 def \_\_drag\_start(self, event):  
 self.\_\_canvas.scan\_mark(event.x, event.y)  
  
 def \_\_drag\_move(self, event):  
 self.\_\_canvas.scan\_dragto(event.x, event.y, gain=1)  
  
 def \_\_draw\_vertex(self, data, x, y):  
 radius = 20  
 self.\_\_canvas.create\_oval(x - radius, y - radius, x + radius, y + radius, fill="white")  
 self.\_\_canvas.create\_text(x, y, text=data, font=(None, round(self.\_\_scale \* 20)), tag="text\_data")  
  
 def \_\_draw\_tree(self, vertex, level=1):  
 cell\_size = 40  
 if vertex:  
 if vertex.left:  
 self.\_\_canvas.create\_line((vertex.data + 1) \* cell\_size, level \* cell\_size,  
 (vertex.left.data + 1) \* cell\_size, (level + 1) \* cell\_size)  
 if vertex.right:  
 self.\_\_canvas.create\_line((vertex.data + 1) \* cell\_size, level \* cell\_size,  
 (vertex.right.data + 1) \* cell\_size, (level + 1) \* cell\_size)  
  
 self.\_\_draw\_vertex(vertex.data, (vertex.data + 1) \* cell\_size, level \* cell\_size)  
 self.\_\_draw\_tree(vertex.left, level + 1)  
 self.\_\_draw\_tree(vertex.right, level + 1)  
  
 def start(self):  
 self.update()  
 self.\_\_window.mainloop()  
  
 def update(self):  
 self.\_\_canvas.delete("all")  
 self.\_\_draw\_tree(self.\_\_tree.root)  
 self.\_\_canvas.update()

Листинг кода из файла main.py:

import random  
  
from tree import AVLTree  
from gtree import GraphicalTree  
  
n = 20  
random.seed(0)  
arr = random.sample(range(n), n)  
avl\_tree = AVLTree(arr)  
  
print("n={0} Размер Контр. сумма Высота Средн.высота".format(n))  
print("АВЛ {0} {1} {2} {3}".format(avl\_tree.size(), avl\_tree.check\_sum(),  
 avl\_tree.height(), avl\_tree.medium\_height()))  
  
g\_tree = GraphicalTree(avl\_tree, "AVL", 1000, 400).start()