|  |  |
| --- | --- |
| |  | | --- | | МИНОБРНАУКИ РОССИИ  Федеральное государственное  бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | |

Институт Информационных технологий

Кафедра корпоративных информационных систем

**ЛАБОРАТОРНАЯ РАБОТА**

по дисциплине «Структуры и алгоритмы обработки данных»

**Тема лабораторной работы** Бинарные деревья.

**Студент группы** ИКБО-12-17 Лисовой А.А. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*(подпись студента)*

**Руководитель работы** Андрианова Е.Г. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*(подпись руководителя)*

Работа представлена к защите «2» марта 2019 г.

Допущен к защите «\_\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_2019 г.

Москва, 2019

**Оглавление**

[Постановка задачи 2](#_Toc2375733)

[Теоретическое введение 3](#_Toc2375734)

[Описание алгоритма решения задачи 4](#_Toc2375735)

[Тестирование программы 5](#_Toc2375736)

[Кодирование алгоритма программы 8](#_Toc2375737)

# Постановка задачи

1. Построить произвольное бинарное дерево с помощью рекурсивной процедуры.
2. Составить функцию или процедуру, которая заменяет в дереве значения всех отрицательных элементов вершин на их абсолютные величины.
3. Составить функцию, которая определяет максимальную глубину непустого бинарного дерева.

Так же необходимо создать консольное меню для удобного доступа к выполнению заданий лабораторной работы.

# Теоретическое введение

**Дерево:**

Дерево - связный ацикличный граф с не ориентированными и невзвешенными ребрами рёбрами. Дерево состоит из узлов, один из которых обязательно является корневым – то есть самым верхним узлом дерева. Внутренним узлом является любой узел дерева, имеющий потомков. Соответственно листом называется узел, потомков лишенный.

**Поддерево:**

Поддерево — часть древообразной структуры данных, которая может быть представлена в виде отдельного дерева. Любой узел дерева T вместе со всеми его узлами-потомками является поддеревом дерева T. Для любого узла поддерева либо должен быть путь в корневой узел этого поддерева, либо сам узел должен являться корневым. То есть поддерево связано с корневым узлом целым деревом, а отношения поддерева со всеми прочими узлами определяются через понятие соответствующее поддерево (по аналогии с термином «соответствующее подмножество»).

**Бинарное дерево:**

Бинарное дерево — это иерархическая структура данных, в которой каждый узел имеет значение и ссылки на левого и правого потомка. Узел, находящийся на самом верхнем уровне (не являющийся чьим-либо потомком) называется корнем. Узлы, не имеющие потомков (оба потомка которых равны NULL) называются листьями.

**Бинарное дерево поиска:**

Бинарное дерево поиска — это бинарное дерево, обладающее дополнительными свойствами: значение левого потомка меньше значения родителя, а значение правого потомка больше значения родителя для каждого узла дерева. То есть, данные в бинарном дереве поиска хранятся в отсортированном виде. При каждой операции вставки нового или удаления существующего узла отсортированный порядок дерева сохраняется. При поиске элемента сравнивается искомое значение с корнем. Если искомое больше корня, то поиск продолжается в правом потомке корня, если меньше, то в левом, если равно, то значение найдено и поиск прекращается. Как правило, реализация бинарного дерева поиска происходит с помощью рекурсивных алгоритмов.

# Описание алгоритма решения задачи

Алгоритм:

1. Ознакомиться с заданиями лабораторной работы и принципами организации работы указанных структур данных

2. Разработать класс-модуль бинарного дерева поиска, выполняющий условия поставленных задач.

3. Разработать класс-модуль интерфейса, реализующий удобную работу с классами-модулями заданий работы.

4. Подключить все классы-модули к главному файлу лабораторной работы, отладить работу интерфейса, провести тест работы остальных классов-модулей лабораторной работы.

# Тестирование программы

**Задание 1:**

Тест №1: Построить произвольное бинарное дерево с помощью рекурсивной процедуры.

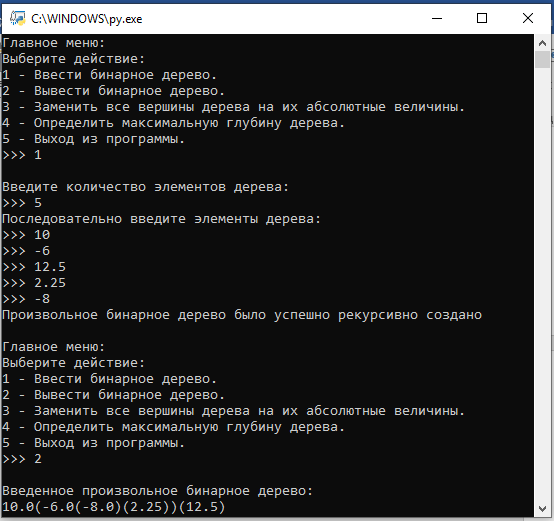


Рис. 1 Подтверждает корректную работу, так как данное дерево было успешно рекурсивно создано функцией tree\_recursive\_Creation класса Binary\_Tree.

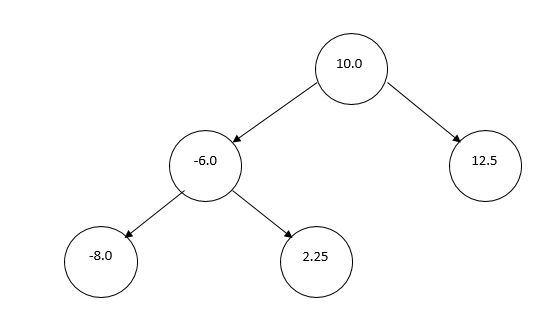


Рис. 2 Графическое отображение рекурсивно созданного выше бинарного дерева.

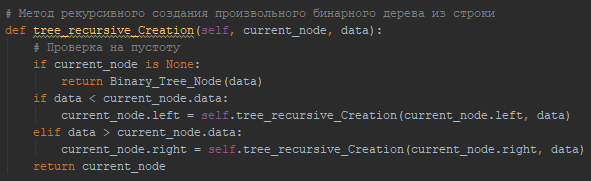


Рис. 3 Метод рекурсивного создания бинарного дерева tree\_recursive\_Creation класса Binary\_Tree.

**Задание 2:**

Тест №2: Заменить все отрицательные узлы бинарного дерева их абсолютными величинами:

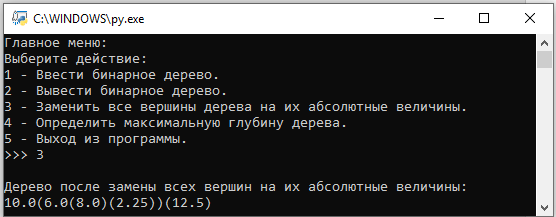


Рис. 4 Подтверждает корректную работу программы, так как с помощью функции abs\_Replace класса Binary\_Tree все отрицательные узлы дерева были заменены их абсолютными величинами.

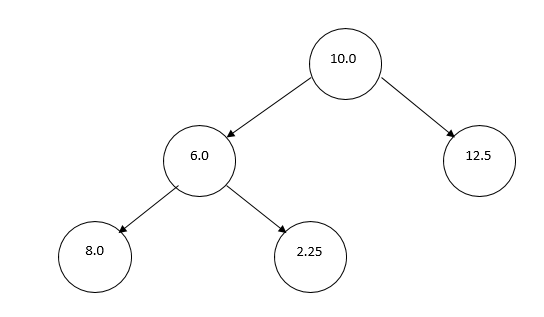


Рис. 5 Графическое отображение изменений в дереве после замены всех отрицательных узлов их абсолютными величинами.

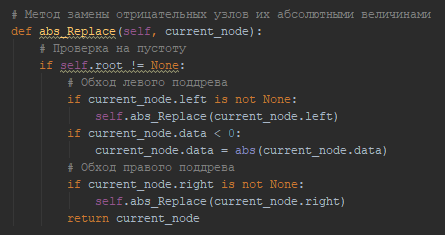


Рис. 6 Метод замены всех отрицательных узлов дерева их абсолютными величинами abs\_Replace класса Binary\_Tree.

**Задание 3:**

Тест №3: Определение максимальной глубины введенного бинарного дерева

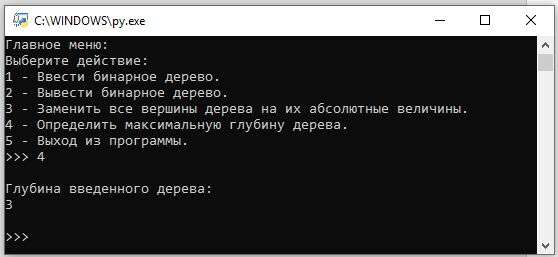


Рис. 7 Подтверждает корректную работу программы, так как так с помощью функции max\_Depth класса Binary\_Tree была верно определена глубина введенного бинарного дерева.

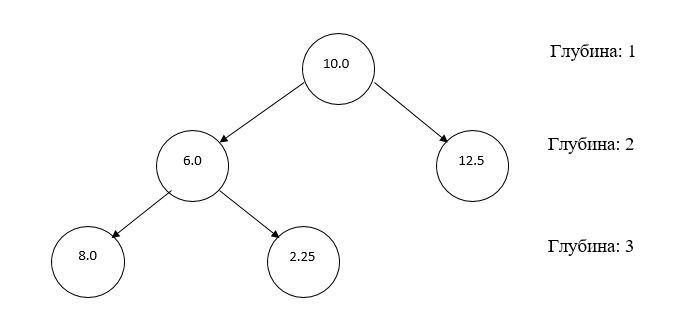


Рис.8 Графическая демонстрация уровней глубины введенного бинарного дерева:

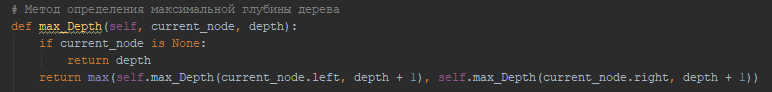


Рис. 9 Метод определения максимальной глубины бинарного дерева max\_Depth класса Binary\_Tree

# Кодирование алгоритма программы

**Модуль LR\_2.py:**

# Список импортов

from Binary\_Tree import Binary\_Tree

from Interface import Interface

# Консольное меню программы

Menu = Interface()

# Вызов метода отображения главного меню

Menu.Interface\_Main()

# Создание объекта класса Дерево

tree = Binary\_Tree()

# Строка вывода

output\_string = ""

# Цикл обработки команд

while True:

# Ввод выбранного действия

Step = Menu.input\_Controller()

# Ввод дерева

if Step == 1:

print("")

print("Введите количество элементов дерева: ")

tree.root = None

length = Menu.input\_Controller()

print("Последовательно введите элементы дерева:")

for i in range(length):

value = Menu.input\_Controller2()

tree.root = tree.tree\_recursive\_Creation(tree.root, value)

print("Произвольное бинарное дерево было успешно рекурсивно

создано")

print("")

Menu.Interface\_Main()

# Вывод дерева

elif Step == 2:

print("")

output\_string = tree.tree\_Output(tree.root)

if output\_string is "":

print("Исходное дерево пустое")

print("")

else:

print("Введенное произвольное бинарное дерево:")

print(output\_string)

print("")

Menu.Interface\_Main()

# Замена на абсолютные величины

elif Step == 3:

if tree.root is not None:

print("")

print("Дерево после замены всех вершин на их абсолютные

величины:")

tree.root = tree.abs\_Replace(tree.root)

print(tree.tree\_Output(tree.root))

print("")

else:

print("Исходное дерево пустое")

print("")

Menu.Interface\_Main()

# Определение максимальной глубины

elif Step == 4:

print("")

print("Глубина введенного дерева:")

print(tree.max\_Depth(tree.root, 0))

print("")

Menu.Interface\_Main()

# Выход из программы

elif Step == 5:

print("Программа завершена")

break

# Остальные случаи

else:

print("Ошибка ввода")

print("")

**Модуль Interface.py:**

# Класс интерфейса

class Interface:

# Конструктор

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

pass

# Главное меню:

def Interface\_Main(self):

print ("Главное меню:")

print ("Выберите действие:")

print ("1 - Ввести бинарное дерево.")

print ("2 - Вывести бинарное дерево.")

print ("3 - Заменить все вершины дерева на их абсолютные

величины.")

print ("4 - Определить максимальную глубину дерева.")

print ("5 - Выход из программы.")

pass

# Защита от некорректного ввода:

def input\_Controller(self):

while True:

try:

Value = int(input(">>> "))

if Value > 0:

return Value

else:

print("Ошибка ввода!")

except ValueError:

print("Ошибка ввода!")

print("")

# Защита от некорректного ввода:

def input\_Controller2(self):

while True:

try:

Value = float(input(">>> "))

return Value

except ValueError:

print("Ошибка ввода!")

print("")

**Модуль Binary\_Tree\_Node.py**

# Класс узла бинарного дерева

class Binary\_Tree\_Node(object):

# Конструктор

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

self.data = data

self.left = self.right = None

**Модуль Binary\_Tree.py**

# Импорт класса узла бинарного дерева

from Binary\_Tree\_Node import Binary\_Tree\_Node

# Класс бинарного дерева

class Binary\_Tree(object):

# Конструктор

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

self.root = None

# Метод рекурсивного создания произвольного бинарного дерева из строки

def tree\_recursive\_Creation(self, current\_node, data):

# Проверка на пустоту

if current\_node is None:

return Binary\_Tree\_Node(data)

if data < current\_node.data:

current\_node.left = self.tree\_recursive\_Creation

(current\_node.left, data)

elif data > current\_node.data:

current\_node.right = self.tree\_recursive\_Creation

(current\_node.right, data)

return current\_node

# Метод рекурсивного вывода дерева

def tree\_Output(self, current\_node):

# Проверка на пустоту

if current\_node is None:

return ""

if current\_node.left is None and current\_node.right is None:

return str(current\_node.data) + ""

if current\_node.right is None:

return str(current\_node.data) + "(" + self.tree\_Output

(current\_node.left) + ")"

#if current\_node.left is None:

#return str(current\_node.data) + "(" + self.tree\_Output

(current\_node.right) + ")"

return str(current\_node.data) + "(" + self.tree\_Output

(current\_node.left) + ")(" + self.tree\_Output(current\_node.right)

+ ")"

# Метод замены отрицательных узлов их абсолютными величинами

def abs\_Replace(self, current\_node):

# Проверка на пустоту

if self.root != None:

# Обход левого поддрева

if current\_node.left is not None:

self.abs\_Replace(current\_node.left)

if current\_node.data < 0:

current\_node.data = abs(current\_node.data)

# Обход правого поддрева

if current\_node.right is not None:

self.abs\_Replace(current\_node.right)

return current\_node

# Метод определения максимальной глубины дерева

def max\_Depth(self, current\_node, depth):

if current\_node is None:

return depth

return max(self.max\_Depth(current\_node.left, depth + 1),

self.max\_Depth(current\_node.right, depth + 1))