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

Институт информационных технологий

Кафедра корпоративных информационных систем

|  |  |  |
| --- | --- | --- |
| **ОТЧЕТ**  **по лабораторной работе №2** | | |
| **по дисциплине** | | |
| **«Структуры и алгоритмы обработки данных»**  **Тема лабораторной работы: «**Списки сложной структуры, бинарные деревья**»** | | |
| Студент группы | ИКБО-06-18 | Литвиненко Б.В. |
| Принял | ассистент кафедры КИС | Габриелян Г.А. |
|  |  |  |
| Выполнено | «\_\_\_» \_\_\_\_\_\_ 201\_\_ г. | *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_* |
|  |  | *(подпись студента)* |
| Зачтено | «\_\_\_» \_\_\_\_\_\_ 201\_\_ г. | *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_* |
|  |  | *(подпись преподавателя)* |

Москва 2019

1. **Задача №1**
   1. **Постановка задачи**

Составить программу, которая каждый элемент кольцевого однонаправленного списка с заглавным элементом увеличивает на 3, а затем вычеркивает те из них, которые будут простыми

* 1. **Описание используемых структур данных**

# класс Узел хранит значение узла и ссылку на следующий элемент

class Node:

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

self.value = value

self.next = None

# класс Кольцевой список. Позволяет связанно хранить элементы

# Хранит ссылку на заглавный элемент, при этом последний

# последний элемент ссылается на первый

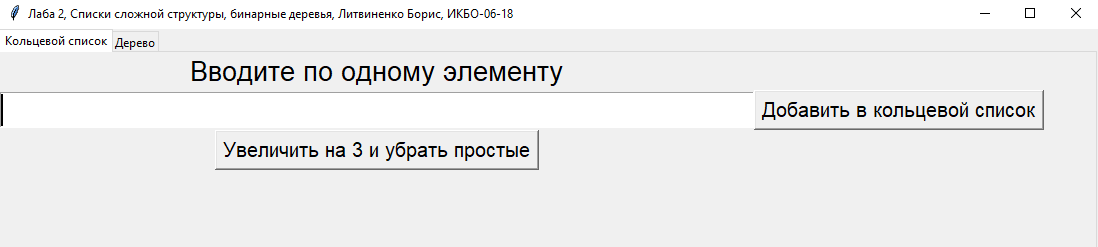
class CircularLinkedList:

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

self.head = None # First element

* 1. **Пользовательский интерфейс**

Содержит информационный виджет Label, поле ввода и кнопки: «Добавить элемент в кольцевой список», «Увеличить элементы на 3 и затем убрать простые».

  
Рисунок 1.1 – Вкладка «Кольцевой список»

* 1. **Описание алгоритма**

Процедура для выполнения задания task() реализована так: сначала с помощью цикла while каждый элемент списка увеличивается на 3. Второй раз цикл while перебирает все элементы и каждое значение проверяет на простоту с помощью метода prime(). Если число оказалось простым, то его метод убирает из списка. Делается это так: если значение узла оказалось простым, то в предыдущем узле ссылка на следующий узел изменяется на последующий узел.

При этом метод генерирует исключения с информационными сообщениями в случаях, если список пуст или входные данные не верны.

* 1. **Тестирование**

|  |  |
| --- | --- |
| Введенные данные | Результат выполнения программы |
| 1 2 3 4 5 6 7 8 | 4 6 8 9 10 |
| 12 tik 34 #42 | Неверные входные данные |
| 2 2 8 2 | Нет элементов |
| 2 2 3 | 6 |
| 6 -7 90 | 9 -4 93 |

Таблица 1.1 – Тестирование «Кольцевой список»

* 1. **Листинг программы**

import math

def prime(n):

if n < 1:

return False

for i in range(2, int(math.sqrt(n) + 1)):

if n % i == 0:

return False

return True

# List Node

class Node:

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

self.value = value

self.next = None

class CircularLinkedList:

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

self.head = None # First element

def append(self, value):

""" New elements """

node = Node(value)

if self.head is None:

self.head = node

self.head.next = self.head

else:

curr = self.head

while curr.next is not self.head:

curr = curr.next

node.next = self.head

curr.next = node

def get\_list(self):

if self.head is None:

return "Нет элементов"

string = ""

curr = self.head

while curr.next is not self.head:

string = string + str(curr.value) + " "

curr = curr.next

string = string + str(curr.value)

return string

def task(self):

if self.head is None:

raise IOError()

curr = self.head

try:

# увеличиваем на 3

while curr.next is not self.head:

curr.value += 3

curr = curr.next

curr.value += 3

# убираем простые

old = self.head

curr = self.head.next

while old.next is not self.head:

if prime(curr.value) is True:

old.next = curr.next

old = old.next

curr = old.next

if (prime(curr.value) and (self.head.next is not self.head)) is True:

old.next = curr.next

self.head = curr.next

elif prime(curr.value) is True:

self.head = None

except IOError:

raise IOError()

except Exception:

raise Exception("Неверные входные параметры")

1. **Задача №2**
   1. **Постановка задачи**

Составить функцию, которая заменяет в дереве T значения всех отрицательных элементов вершин на их абсолютные величины.

* 1. **Описание используемых структур данных**

# класс Узел хранит значение узла, а так же ссылки на два дочерних узла

class Node:

def \_\_init\_\_(self, value=None):

self.value = value

self.left = None

self.right = None

# класс дерево связывает узлы между собой. Хранит корень дерева – root, от # которого далее идут потомки (другие узлы)

class Tree:

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

self.root = None

* 1. **Пользовательский интерфейс**

Содержит информационный виджет Label, поле ввода и кнопки: добавить элемент в очередь, вытащить последний элемент из очереди, посчитать сумму и произведение элементов.

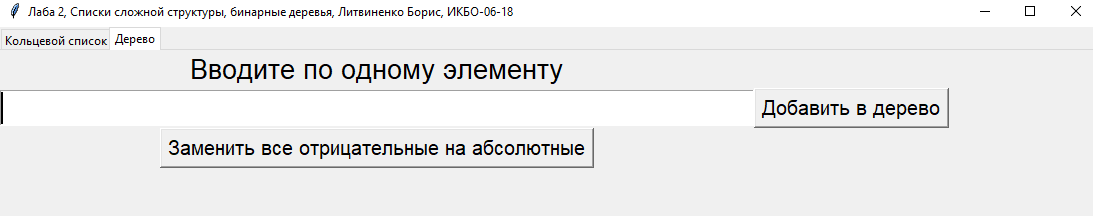


Рисунок 2.1 - Вкладка «Дерево».

* 1. **Описание алгоритма**

Функция task(), которая заменяет в дереве T значения всех отрицательных элементов вершин на их абсолютные величины, устроена так: проделывается прямой обход (NLR). К значению каждого узла применяется функция fabs() библиотеки math.

* 1. **Тестирование**

|  |  |
| --- | --- |
| Введенные данные | Вывод программы |
| -12 2 12 2 2 -4 -5 | 12 2 12 2 2 4 5 |
| 1 2 к 4 5 | Неверные входные данные |
| 1 -2 3 -4 5 -6 7 | 1 2 3 4 5 6 7 |
| -1 2 -3 4 -5 6 -7 | 1 2 3 4 5 6 7 |
| -1 -2 -3 -4 -5 -6 -7 | 1 2 3 4 5 6 7 |

Таблица 2.1 – «Тестирование дерева»

* 1. **Листинг программы**

from math import fabs

class Node:

def \_\_init\_\_(self, value=None):

self.value = value

self.left = None

self.right = None

class Tree:

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

self.root = None

self.result = ""

def insert(self, value):

if self.root is None:

self.root = Node(value)

else:

self.\_insert(value, self.root)

def \_insert(self, value, cur\_node):

if value<cur\_node.value:

if cur\_node.left is None:

cur\_node.left = Node(value)

else:

self.\_insert(value, cur\_node.left)

else:

if cur\_node.right is None:

cur\_node.right = Node(value)

else:

self.\_insert(value, cur\_node.right)

def print\_tree(self):

self.result = ""

if self.root is not None:

self.\_print\_tree(self.root)

return self.result

def \_print\_tree(self, cur\_node):

if cur\_node is not None:

self.\_print\_tree(cur\_node.left)

self.result += ((str(cur\_node.value))+" ")

self.\_print\_tree(cur\_node.right)

def task(self):

if self.root is not None:

self.\_task(self.root)

def \_task(self, cur\_node):

if cur\_node is not None:

self.\_task(cur\_node.left)

cur\_node.value = int(fabs(cur\_node.value))

self.\_task(cur\_node.right)