|  |
| --- |
| 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: «**Структуры данных: список, очередь, стек**»** | | |
| Студент группы | ИКБО-10-18 | Минаев А. И. |
| 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**

Сформировать кольцевой однонаправленный список с заглавным элементом, составленный из целых чисел. Из каждого последующего элемента вычесть предыдущий.

* 1. **Описание используемых структур данных**

Кольцевым однонаправленным списком называется линейный список, в котором указатель из некоторого звена направлен на такое звено в списке, из которого данное звено может быть достигнуто вновь.

class Node:

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

self.value = value

self.next = next

class CircleList:

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

self.head = None

def push(self, value):

if self.head == None:

self.head = Node(value)

self.head.next = self.head

else:

current = self.head

while current.next != self.head:

current = current.next

current.next = Node(value, self.head)

* 1. **Пользовательский интерфейс**

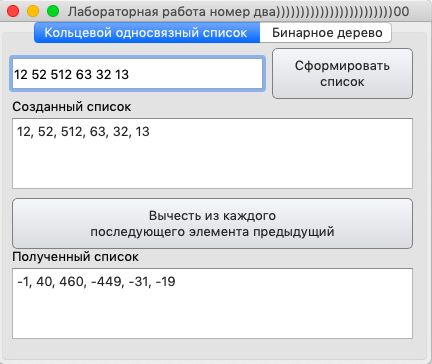


Рисунок 1.1 – Графический интерфейс для задания 1

* 1. **Описание алгоритма**

Сначала во временную переменную мы записываем значения первого элемента списка. Затем проходим по элементам по элементам списка в цикле while, пока первый элемент не станет следующим элементом выбранного, одновременно присваивая временной переменной значение следующего элемента относительно выбранного и вычитая из него старое значение временной переменной.

current = self.head

temp = current.value

while current.next != self.head:

current.next.value, temp = current.next.value - temp, current.next.value

current = current.next

current.next.value -= temp

* 1. **Тестирование**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Номер теста | Входные данные | Ожидаемый результат | Результат программы | Отметка о прохождении |
| 1. | 55, 42, 13, 25, 72 | -17, -13, -29, 12, 47 | -17, -13, -29, 12, 47 | Пройден |
| 2. | 76, 45, 13 | 63, -31, -32 | 63, -31, -32 | Пройден |
| 3. | 55 | 0 | 0 | Пройден |

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

Составить функцию, которая удаляет из непустого бинарного дерева T вершины, содержащие максимальный и минимальный элемент.

* 1. **Описание используемых структур данных**

Бинарное дерево — иерархическая структура данных, в которой каждый узел имеет не более двух потомков (детей). Как правило, первый называется родительским узлом, а дети называются левым и правым наследниками. В бинарном дереве поиска значение левого наследника всегда меньше, а правого больше.

class TreeNode:

def \_\_init\_\_(self, key = None, left = None, right = None):

self.key = key

self.left = left

self.right = right

def insert(self, key):

if not self.key:

self.key = key

elif key < self.key:

if not self.left:

self.left = TreeNode(key)

else:

self.left.insert(key)

elif key > self.key:

if not self.right:

self.right = TreeNode(key)

self.right.insert(key)

def printTree(self):

if self.left:

self.left.printTree()

print(self.key)

if self.right:

self.right.printTree()

* 1. **Пользовательский интерфейс**

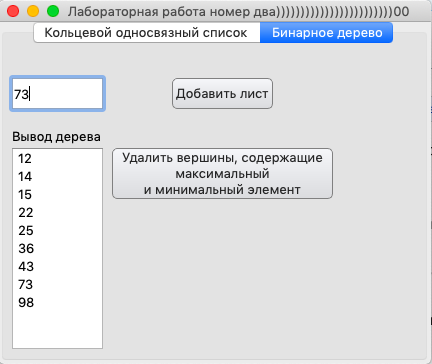


Рисунок 2.1 – Графический интерфейс для задания 2

* 1. **Описание алгоритма**

Сначала проверяется есть ли у первого листа минимальный/максимальный лист, и если есть только один лист, то функция возвращает пустое дерево. Дальше функции проверяют, есть ли у левых/правых листов листы дальше, и, основываясь на этом либо двигаются по дереву дальше, либо удаляют нужный лист и возвращают дерево.

def delMin(self):

if not self.left:

return self.right if self.right else TreeNode()

if self.left.left:

tree = self.left

else:

self.left = self.left.right

return self

while (tree.left.left):

tree = tree.left

if tree.left.right:

tree.left.key = tree.left.right.key

tree.left.left = tree.left.right.left

tree.left.right = tree.left.right.right

else:

tree.left = None

return self

def delMax(self):

if not self.right:

return self.left if self.left else TreeNode()

if self.right.right:

tree = self.right

else:

self.right = self.right.left

return self

while (tree.right.right):

tree = tree.right

if tree.right.left:

tree.right.key = tree.right.left.key

tree.right.right = tree.right.left.right

tree.right.left = tree.right.left.left

else:

tree.right = None

return self

* 1. **Тестирование**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Номер теста | Входные данные | Ожидаемый результат | Результат программы | Отметка о прохождении |
| 1. | 1 2 3 4 5 | 2 3 4 | 2 3 4 | Пройден |
| 2. | 36 75 | None | None | Пройден |
| 3. | 12 45 72 | 45 | 45 | Пройден |

**Листинг программы**

from PyQt5 import QtCore, QtGui, QtWidgets

from circle import CircleList

from binarytree import TreeNode

class Ui\_MainWindow(object):

def setupUi(self, MainWindow):

MainWindow.setObjectName("MainWindow")

MainWindow.resize(432, 342)

MainWindow.setCursor(QtGui.QCursor(QtCore.Qt.ArrowCursor))

MainWindow.setStyleSheet("QPushButton {border: 1px solid grey; border-radius: 5px; background-color: qlineargradient(x1: 0, y1: 0, x2: 0, y2: 1, stop: 0 #f6f7fa, stop: 1 #dadbde)}\n"

"QPushButton:pressed { background-color: qlineargradient(x1: 0, y1: 0, x2: 0, y2: 1, stop: 0 #dadbde, stop: 1 #E5CCFF) }")

self.centralwidget = QtWidgets.QWidget(MainWindow)

self.centralwidget.setObjectName("centralwidget")

self.tabWidget = QtWidgets.QTabWidget(self.centralwidget)

self.tabWidget.setGeometry(QtCore.QRect(0, 0, 431, 341))

self.tabWidget.setObjectName("tabWidget")

self.tab = QtWidgets.QWidget()

self.tab.setObjectName("tab")

self.lineEdit = QtWidgets.QLineEdit(self.tab)

self.lineEdit.setGeometry(QtCore.QRect(10, 10, 251, 31))

self.lineEdit.setObjectName("lineEdit")

self.pushButton = QtWidgets.QPushButton(self.tab)

self.pushButton.setGeometry(QtCore.QRect(270, 0, 141, 51))

self.pushButton.setObjectName("pushButton")

self.pushButton.clicked.connect(self.btn1Clicked)

self.label = QtWidgets.QLabel(self.tab)

self.label.setGeometry(QtCore.QRect(10, 50, 121, 16))

self.label.setObjectName("label")

self.label\_2 = QtWidgets.QLabel(self.tab)

self.label\_2.setGeometry(QtCore.QRect(10, 200, 131, 16))

self.label\_2.setObjectName("label\_2")

self.pushButton\_2 = QtWidgets.QPushButton(self.tab)

self.pushButton\_2.setGeometry(QtCore.QRect(10, 150, 401, 51))

self.pushButton\_2.setObjectName("pushButton\_2")

self.pushButton\_2.clicked.connect(self.btn2Clicked)

self.plainTextEdit = QtWidgets.QPlainTextEdit(self.tab)

self.plainTextEdit.setGeometry(QtCore.QRect(10, 70, 401, 71))

self.plainTextEdit.setReadOnly(True)

self.plainTextEdit.setObjectName("plainTextEdit")

self.plainTextEdit.viewport().setCursor(QtCore.Qt.ArrowCursor)

self.plainTextEdit\_2 = QtWidgets.QPlainTextEdit(self.tab)

self.plainTextEdit\_2.setGeometry(QtCore.QRect(10, 220, 401, 71))

self.plainTextEdit\_2.setReadOnly(True)

self.plainTextEdit\_2.setObjectName("plainTextEdit\_2")

self.plainTextEdit\_2.viewport().setCursor(QtCore.Qt.ArrowCursor)

self.tabWidget.addTab(self.tab, "")

self.tab\_2 = QtWidgets.QWidget()

self.tab\_2.setObjectName("tab\_2")

self.tabWidget.addTab(self.tab\_2, "")

self.lineEdit\_2 = QtWidgets.QLineEdit(self.tab\_2)

self.lineEdit\_2.setGeometry(QtCore.QRect(10, 30, 91, 31))

self.lineEdit\_2.setObjectName("lineEdit\_2")

self.pushButton\_3 = QtWidgets.QPushButton(self.tab\_2)

self.pushButton\_3.setGeometry(QtCore.QRect(170, 30, 101, 31))

self.pushButton\_3.setObjectName("pushButton\_3")

self.pushButton\_3.clicked.connect(self.btn3Clicked)

self.pushButton\_4 = QtWidgets.QPushButton(self.tab\_2)

self.pushButton\_4.setGeometry(QtCore.QRect(110, 100, 221, 51))

self.pushButton\_4.setObjectName("pushButton\_4")

self.pushButton\_4.clicked.connect(self.btn4Clicked)

self.listWidget = QtWidgets.QListWidget(self.tab\_2)

self.listWidget.setGeometry(QtCore.QRect(10, 100, 91, 201))

self.listWidget.setObjectName("listWidget")

self.label\_3 = QtWidgets.QLabel(self.tab\_2)

self.label\_3.setGeometry(QtCore.QRect(10, 80, 91, 16))

self.label\_3.setObjectName("label\_3")

self.label\_4 = QtWidgets.QLabel(self.tab\_2)

self.label\_4.setGeometry(QtCore.QRect(110, 160, 221, 141))

self.label\_4.setText("")

self.label\_4.setAlignment(QtCore.Qt.AlignHCenter|QtCore.Qt.AlignTop)

self.label\_4.setWordWrap(True)

self.label\_4.setObjectName("label\_4")

self.tabWidget.addTab(self.tab\_2, "")

MainWindow.setCentralWidget(self.centralwidget)

self.retranslateUi(MainWindow)

self.tabWidget.setCurrentIndex(0)

QtCore.QMetaObject.connectSlotsByName(MainWindow)

def retranslateUi(self, MainWindow):

\_translate = QtCore.QCoreApplication.translate

MainWindow.setWindowTitle(\_translate("MainWindow", "Лабораторная работа номер два))))))))))))))))))))))))00"))

self.pushButton.setText(\_translate("MainWindow", "Сформировать\n"

"список"))

self.label.setText(\_translate("MainWindow", "Созданный список"))

self.label\_2.setText(\_translate("MainWindow", "Полученный список"))

self.pushButton\_2.setText(\_translate("MainWindow", "Вычесть из каждого\n"

"последующего элемента предыдущий"))

self.tabWidget.setTabText(self.tabWidget.indexOf(self.tab), \_translate("MainWindow", "Кольцевой односвязный список"))

self.pushButton\_3.setText(\_translate("MainWindow", "Добавить лист"))

self.pushButton\_4.setText(\_translate("MainWindow", "Удалить вершины, содержащие\n"

"максимальный\n"

"и минимальный элемент"))

self.label\_3.setText(\_translate("MainWindow", "Вывод дерева"))

self.tabWidget.setTabText(self.tabWidget.indexOf(self.tab\_2), \_translate("MainWindow", "Бинарное дерево"))

crcl = None

def btn1Clicked(self):

try:

self.crcl = CircleList()

s = self.lineEdit.text()

if s == "":

raise Exception("ничего не введено")

arr = list(map(int, s.split()))

s = ""

for i in arr:

self.crcl.push(i)

s += str(i) + ", "

s = s[:-2]

self.plainTextEdit.setPlainText(s)

except ValueError:

self.plainTextEdit.setPlainText("Список должен состоять из натуриальных чисел, введенных через пробел")

except Exception as e:

self.plainTextEdit.setPlainText("Ошибка: " + str(e))

def btn2Clicked(self):

try:

if not self.crcl:

raise Exception("список не заполнен")

self.plainTextEdit\_2.setPlainText(self.crcl.action())

except AttributeError:

self.plainTextEdit\_2.setPlainText("Ошибка: список не заполнен")

except Exception as e:

self.plainTextEdit\_2.setPlainText("Ошибка: " + str(e))

tree = TreeNode()

arr = []

def outTree(self, tree):

if tree.left:

self.outTree(tree.left)

self.arr.append(tree.key)

if tree.right:

self.outTree(tree.right)

def btn3Clicked(self):

try:

self.tree.insert(int(self.lineEdit\_2.text().strip()))

self.arr = []

self.outTree(self.tree)

self.listWidget.clear()

for i in self.arr:

self.listWidget.addItem(str(i))

self.label\_4.setText("")

except Exception:

self.label\_4.setText("Ошибка: неправильно введен лист дерева")

def btn4Clicked(self):

try:

if len(self.arr) == 0:

raise Exception("в дереве нет элементов")

self.tree = self.tree.delMin()

self.tree = self.tree.delMax()

self.arr = []

self.outTree(self.tree)

self.listWidget.clear()

for i in self.arr:

self.listWidget.addItem(str(i))

self.label\_4.setText("")

except Exception as e:

self.label\_4.setText("Ошибка: " + str(e))

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

import sys

app = QtWidgets.QApplication(sys.argv)

MainWindow = QtWidgets.QMainWindow()

ui = Ui\_MainWindow()

ui.setupUi(MainWindow)

MainWindow.show()

sys.exit(app.exec\_())