|  |
| --- |
| 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-08-18 | Валяев Д.А. |
| Принял | ассистент кафедры КИС | Габриелян Г.А. |
|  |  |  |
| Выполнено | «\_\_\_» \_\_\_\_\_\_ 201\_\_ г. | *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_* |
|  |  | *(подпись студента)* |
| Зачтено | «\_\_\_» \_\_\_\_\_\_ 201\_\_ г. | *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_* |
|  |  | *(подпись преподавателя)* |

Москва 2019

**Задача №1**

1. **Постановка задачи**

Сформировать кольцевой однонаправленный список с заглавным элементом. Составить процедуру или функцию, которая подсчитывает количество элементов списка, у которых равные «соседи».

1. **Описание используемых структур данных**

Кольцевой однонаправленный список - это структура данных, состоящая из элементов одного типа, связанных между собой последовательно посредством указателей. Каждый элемент списка имеет указатель на следующий элемент. Последний элемент списка указывает на первый.

class Node:

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

self.data = data

self.next = None

1. **Пользовательский интерфейс**

Пользовательский интерфейс содержит: list – для просмотра содержимого списка, entry – для ввода элементов, которые необходимо добавить, кнопка «Добавить элемент» - для добавления нового элемента, кнопка «Узнать результат» – для расчета количества элементов, у которых равные соседи.

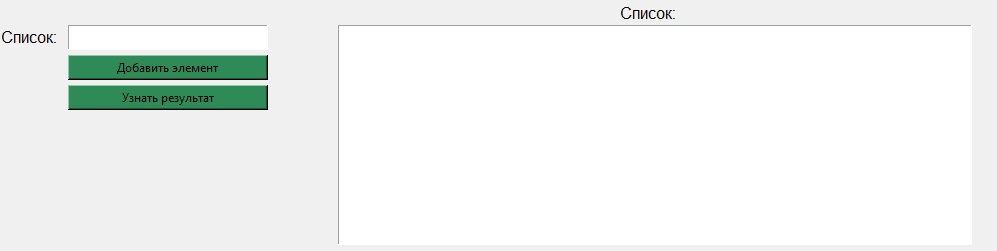


Рисунок 1.1 – Интерфейс кольцевого списка

1. **Описание алгоритма**

При нажатии на кнопку «Узнать результат», программа проходит по всему списку и проверяет, если текущий элемент равен элементу, который находится через один от текущего элемента, то программа добавляет единицу к счетчику чисел, у которых равные соседи.

1. **Тестирование**

На вход программы последовательно поступают числа: 3, 0, 9, 6, 3, 5, 3, 1

Теоретический результат работы программы: Элементов: 2

Практический результат работы программы: Элементов: 2

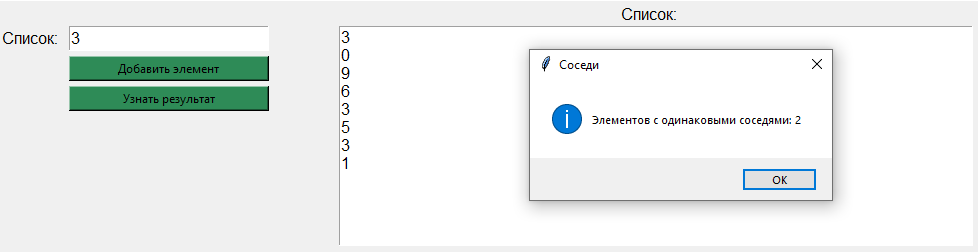


Рисунок 1.2 – Тестирование кольцевого списка

**Задача №2**

1. **Постановка задачи**

Составить процедуру Copy (T,T1), которая строит бинарное дерево T1 - копию дерева T.

1. **Описание используемых структур данных**

Двоичное дерево — иерархическая структура данных, в которой каждый узел имеет не более двух потомков. Как правило, первый называется родительским узлом, а потомки называются левым и правым наследниками.

class Node\_tree:

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

self.key = key

self.left = left

self.right = right

1. **Пользовательский интерфейс**

Пользовательский интерфейс содержит: list – для просмотра содержимого списка, кнопка «Вывести дерево» – для вывода начального дерева, кнопка «Вывести копию» – для вывода копии дерева.



Рисунок 1.3 – Интерфейс дерева

1. **Описание алгоритма**

При нажатии на кнопку «Вывести дерево», программа выводит дерево из 6-15 элементов, состоящее из чисел 0-50. При нажатии на кнопку “Вывести копию”, программа выводит копию начального дерева, составленную методом copy.

1. **Тестирование**

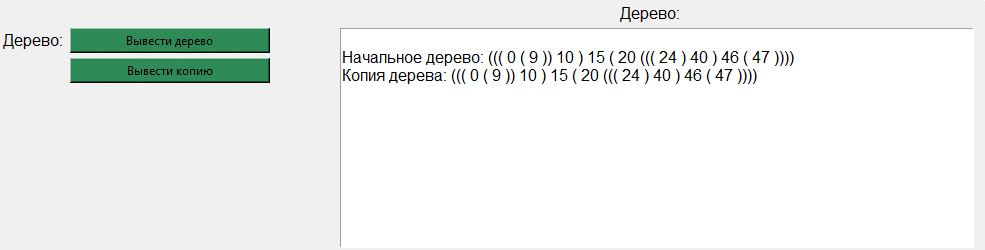


Рисунок 1.4 – Тестирование дерева

1. **Листинг программы**

import tkinter as tk

from tkinter import messagebox

import random

root = tk.Tk()

text\_list = tk.Text(root, font=12, width=70, height=12)

text\_list.configure(state="disabled")

text\_list.place(relx=0.34, rely=0.05)

text\_tree = tk.Text(root, font=12, width=70, height=12)

text\_tree.configure(state="disabled")

text\_tree.place(relx=0.34, rely=0.55)

tree = None

temp = []

class Node\_tree:

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

self.key = key

self.left = left

self.right = right

def \_\_str\_\_(self):

return "(%s %s %s)" % (self.left or "", self.key, self.right or "")

def insert(tree, key):

if not tree:

tree = Node\_tree(key)

elif key < tree.key:

tree = Node\_tree(tree.key, insert(tree.left, key), tree.right)

elif key > tree.key:

tree = Node\_tree(tree.key, tree.left, insert(tree.right, key))

return tree

def copy(tree):

if tree:

temp.append(tree.key)

copy(tree.left)

copy(tree.right)

class Node:

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

self.data = data

self.next = None

class LinkedList:

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

self.head = None

def append(self, data):

if not self.head:

self.head = Node(data)

self.head.next = self.head

else:

newNode = Node(data)

cur = self.head

while cur.next != self.head:

cur = cur.next

cur.next = newNode

newNode.next = self.head

def check(self):

if self.head is None:

messagebox.showerror("Count error", "Список пуст")

return

cur = self.head

curr = self.head

count = 0

while curr.next != cur:

if curr.data == curr.next.next.data:

count += 1

curr = curr.next

else:

curr = curr.next

else:

if curr.data == curr.next.next.data:

count += 1

messagebox.showinfo("Соседи", "Элементов с одинаковыми соседями: " + str(count))

list\_ = LinkedList()

def addList():

text\_list.configure(state="normal")

if len(entry\_list.get()) != 0:

list\_.append(entry\_list.get())

text\_list.insert(0.0, entry\_list.get() + '\n')

else:

messagebox.showerror("Entry error", "Поле ввода не должно быть пустым")

text\_list.configure(state="disabled")

times = random.randint(6, 15)

while times:

element = random.randint(0, 50)

tree = insert(tree, element)

times += -1

def copyTree():

tree2 = None

copy(tree)

for i in range(len(temp)):

tree2 = insert(tree2, temp[i])

text\_tree.configure(state="normal")

text\_tree.insert(tk.END, '\nКопия дерева: ')

text\_tree.insert(tk.END, tree2)

text\_tree.configure(state="disabled")

def printTree(tree):

text\_tree.configure(state="normal")

text\_tree.insert(tk.END, '\nНачальное дерево: ')

text\_tree.insert(tk.END, tree)

text\_tree.configure(state="disabled")

# region UI создание графического интерфейса

label\_list = tk.Label(root, font=12, text="Список:", fg='black', bg=None)

label\_list.place(relx=0.4, rely=0.01, relwidth=0.50, relheight=0.03)

label\_tree = tk.Label(root, font=12, text="Дерево:", fg='black', bg=None)

label\_tree.place(relx=0.4, rely=0.5, relwidth=0.50, relheight=0.04)

button\_add = tk.Button(root, text="Добавить элемент", bg='#2E8B57', command=lambda: addList())

button\_add.place(relx=0.07, rely=0.11, relwidth=0.20, relheight=0.05)

button\_check = tk.Button(root, text="Узнать результат", bg='#2E8B57', command=lambda: list\_.check())

button\_check.place(relx=0.07, rely=0.17, relwidth=0.20, relheight=0.05)

button\_check\_tree = tk.Button(root, text="Вывести дерево", bg='#2E8B57', command=lambda: printTree(tree))

button\_check\_tree.place(relx=0.07, rely=0.55, relwidth=0.20, relheight=0.05)

button\_check\_tre = tk.Button(root, text="Вывести копию", bg='#2E8B57', command=lambda: copyTree())

button\_check\_tre.place(relx=0.07, rely=0.61, relwidth=0.20, relheight=0.05)

entry\_list = tk.Entry(root, font=12)

entry\_list.place(relx=0.07, rely=0.05, relwidth=0.20, relheight=0.05)

label\_list = tk.Label(root, font=12, text="Список:", fg='black')

label\_list.place(relx=0, rely=0.05)

label\_tree = tk.Label(root, font=12, text="Дерево:", fg='black')

label\_tree.place(relx=0, rely=0.55)

# endregion

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

root.title("Кольцевой список и дерево")

root.geometry("1000x500")

root.resizable(False, False)

root.mainloop()