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 №1
по дисциплине
«Структуры и алгоритмы обработки данных»

Тема лабораторной работы: «Структуры данных: список, очередь, стек»

Студент группы	ИКБО-07-18	Фроленко М.Д
Принял	ассистент кафедры КИС	Габриелян Г.А.

Выполнено «___» _____ 201__ г. _____
(подпись студента)

Зачтено «___» _____ 201__ г. _____
(подпись преподавателя)

Москва 2019

1. Задача №1

1.1. Постановка задачи

Сформируйте два двунаправленных списка. Составить процедуру для добавления перевернутого первого списка на k-ую позицию второго двунаправленного списка.

1.2. Описание используемых структур данных

Линейный двунаправленный список – структура данных , представляющая из себя набор узлов и связей между ними. В данном случае у каждого узла есть хранимое значение ,а так же связи между предыдущим и следующим узлом.

1.3. Пользовательский интерфейс

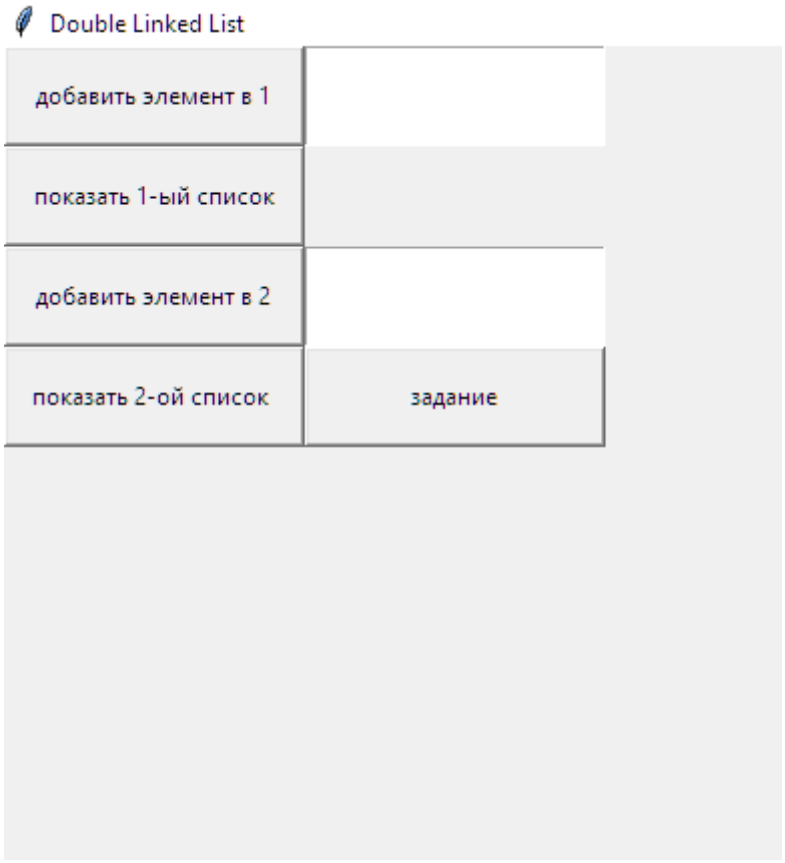

Рисунок 1.3.1 - Интерфейс задания 1

1.4. Описание алгоритма

Для необходимого списка , который нужно перевернуть , мы проходимся по всем узлам и меняем связи на противоположные. Затем создаем копию перевернутого списка и добавляем на k позицию каждый узел списка

1.5. Тестирование

Тестирование проведено на различных входных данных. Ошибок не найдено

1.6. Листинг программы

1.6.1. DLL.py :

```
class Node:


    def __init__(self, data):
        self.data = data
        self.next = None
        self.prev = None

    def __str__(self):
        return str(self.data)

class DoublyLinkedList:

    def __init__(self):
        self.head = None

    def size(self):
        count = 0
        node_iterator = self.head
        while node_iterator is not None:
            node_iterator = node_iterator.next
            count+=1
        return count

    def reverse(self):
        temp = None
        current = self.head


        while current is not None:
            temp = current.prev
            current.prev = current.next
            current.next = temp
            current = current.prev


        if temp is not None:
            self.head = temp.prev


    def push(self, new_data):


        new_node = Node(new_data)
        if self.head is None:
            self.head = new_node
        else:
            node_iterator = self.head
            while node_iterator.next is not None:
                node_iterator = node_iterator.next
            node_iterator.next = new_node
            new_node.prev = node_iterator
```

```
    def pushToK(self, value , k):
        new_node = Node(value)
        node_iterator = self.head
        count = 0
        if(k > self.size() or k < 0):
             return "Нет позиции"
        if(k !=0):
            while count != k-1:
                node_iterator = node_iterator.next
                count+=1
            if(node_iterator.next is None):
                node_iterator.next = new_node
                new_node.prev = node_iterator
            else:
                new_node.next = node_iterator.next
                new_node.prev = node_iterator
                node_iterator.next = new_node
                new_node.next.prev = new_node
        else:
            new_node.next = self.head
            self.head.prev = new_node
            self.head = new_node

    def printList(self, node):
        s = ""
        while(node is not None):
            s += "{} ".format(str(node))
            node = node.next
        return s
```

1.6.2. DLLwin.py:

```
from tkinter import *
from DLL import *
import copy

dllFirst = DoublyLinkedList()
dllSecond = DoublyLinkedList()

def addToFirst():
    dllFirst.push(int(entry1.get()))

def addToSecond():
    dllSecond.push(int(entry2.get()))

def showFirst():
    show.set(dllFirst.printList(dllFirst.head))

def showSecond():
    show.set(dllSecond.printList(dllSecond.head))

def Task(k):
    cursor = k
    dllSecond_copy = copy.deepcopy(dllSecond)
    dllSecond_copy.reverse()
    node_iterator = dllSecond_copy.head
    while node_iterator is not None:
        dllFirst.pushToK(node_iterator.data , cursor)
        node_iterator = node_iterator.next
        cursor += 1

root = Tk()
root.title("Double Linked List")
```

```
root.geometry("300x300")

show = StringVar()
entry1 = StringVar()
entry2 = StringVar()

addButton1 = Button(root, text="добавить элемент в 1",command =
addToFirst)
addButton1.place(x=0, y=0, width=150, height=50 )

addEntry = Entry(root , textvariable = entry1)
addEntry.place(x=150, y=0, width=150, height=50)
showButton1 = Button(root, text="показать 1-ый список" , command =
showFirst )
showButton1.place(x=0, y=50, width=150, height=50)

addButton2 = Button(root, text="добавить элемент в 2" , command =
addToSecond)
addButton2.place(x=0, y=100, width=150, height=50)

addEntry = Entry(root , textvariable = entry2)
addEntry.place(x=150, y=100, width=150, height=50)
showButton2 = Button(root, text="показать 2-ой список " , command =
showSecond)
showButton2.place(x=0, y=150, width=150, height=50)


showButton = Button(root, text="задание" , command = lambda : Task(1))
showButton.place(x=150, y=150, width=150, height=50)

showLabel = Label(root, textvariable=show)
showLabel.place(x=0, y=300, width=300, height=150)

root.mainloop()
```

2. Задача №2

2.1. Постановка задачи

Выписать все вершины дерева поиска, находящиеся на заданном уровне k, используя функцию определения пути от данной вершины до корня.

2.2. Описание используемых структур данных

Дерево поиска- бинарное дерево , в котором при добавлении нового элемента он отправляется в левый узел , если он меньше родителя , и в правый , если больше. Каждый узел дерева содержит значение , ссылку на левый и правый подузел.

2.3. Пользовательский интерфейс

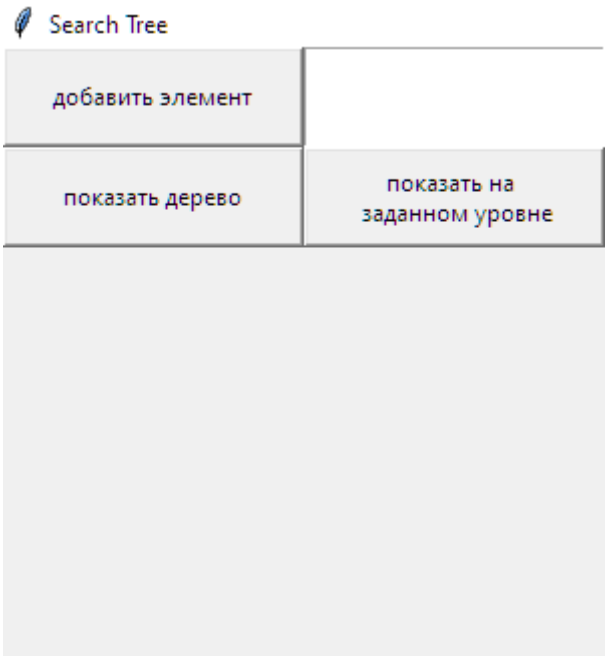

Рисунок 2.3.1 - Интерфейс задания 2

2.4. Описание алгоритма

Рассматривается каждый узел , если его уровень больше того , что подается на вход функции , то мы смотрим в левое и правое поддерево до тех пор , пока не найдем узел с нужным значением уровня . Затем вернем строку ., разделенную запятыми . И вернем каждое значение , которое является числом.

2.5. Тестирование

Тестирование проведено на различных входных данных. Ошибок не найдено

2.6. Листинг программы

2.6.1. BST.py:

```
class Node:

      def __init__(self,info):

          self.info = info
          self.left = None
          self.right = None
          self.level = None

      def __str__(self):

          return str(self.info)


class searchTree:

      def __init__(self):
```

```
    self.root = None


def create(self,val):

    if self.root == None:

       self.root = Node(val)

    else:

       current = self.root

       while 1:

           if val < current.info:

             if current.left:
                current = current.left
             else:
                current.left = Node(val)
                break;

           elif val > current.info:

              if current.right:
                 current = current.right
              else:
                 current.right = Node(val)
                 break;

           else:
              break

def bft(self):

    self.root.level = 0
    queue = [self.root]
    out = []
    current_level = self.root.level

    while len(queue) > 0:

       current_node = queue.pop(0)

       if current_node.level > current_level:
          current_level += 1
          out.append("\n")

       out.append(str(current_node.info) + " ")

       if current_node.left:

          current_node.left.level = current_level + 1
          queue.append(current_node.left)


       if current_node.right:

          current_node.right.level = current_level + 1
          queue.append(current_node.right)
```

```
            print ("".join(out))


      def inorder(self,node):

           if node is not None:

              self.inorder(node.left)
              print (node.info)
              self.inorder(node.right)


      def preorder(self,node):
           if node is None:
               return ""
           if node is not None:

              return str(node.info) +"( "+
str(self.preorder(node.left)) +" , "+str(self.preorder(node.right)) + "
)"


      def postorder(self,node):

           if node is not None:

              self.postorder(node.left)
              self.postorder(node.right)
              print (node.info)

      def searchAtLevel(self , node  , level):
           s = ""
           if node is None :
              return
           elif level == 1:
               return node.info
           elif level > 1 :
               return [self.searchAtLevel(node.left , level - 1)  ,
self.searchAtLevel(node.right , level - 1)]
```

2.6.2. BSTwin.py:

```
from tkinter import *
from BST import *

tree = searchTree()

def add():
    tree.create(int(entry.get()))

def showTree():
    show.set(tree.preorder(tree.root))

def Task(level):
    res = ""
    string = ",".join(str(tree.searchAtLevel(tree.root , level)))
    for x in string.split(","):
        if x.isdigit() :
            res += "{} ".format(x)
    show.set(res)


root = Tk()
```

```
root.title("Search Tree")
root.geometry("300x300")
show = StringVar()
entry = StringVar()



addButton = Button(root, text="добавить элемент" , command = add)
addButton.place(x=0, y=0, width=150, height=50)

addEntry = Entry(root , textvariable = entry)
addEntry.place(x=150, y=0, width=150, height=50)
showButton = Button(root, text="показать дерево" , command = showTree)
showButton.place(x=0, y=50, width=150, height=50)

showButton = Button(root, text="показать на \n заданном уровне"  ,
command = lambda : Task(2))
showButton.place(x=150, y=50, width=150, height=50)

showLabel = Label(root, textvariable=show)
showLabel.place(x=0, y=100, width=300, height=200)
3.
4. root.mainloop()
```