|  |
| --- |
| 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. **Постановка задачи**

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

* 1. **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. **Пользовательский интерфейс**

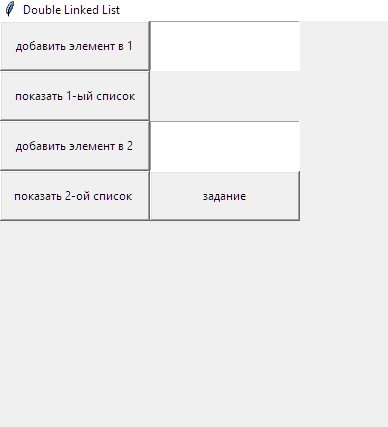


Рисунок 1.3.1 - Интерфейс задания 1

* 1. **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. **Тестирование**

Тестирование проведено на различных входных данных. Ошибок не найдено

* 1. **Листинг программы**
     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. **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()

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

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

* 1. **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.

* 1. **Пользовательский интерфейс**

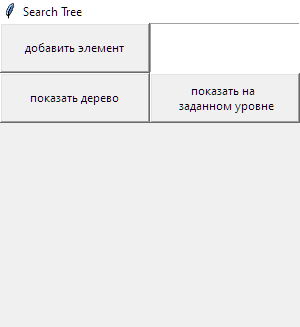


Рисунок 2.3.1 - Интерфейс задания 2

* 1. **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 явлется числом.

* 1. **Тестирование**

Тестирование проведено на различных входных данных. Ошибок не найдено

* 1. **Листинг программы**
     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)]

* + 1. **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)

1. root.mainloop()