**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МОЭВМ**

отчет

**по лабораторной работе №2**

**по дисциплине «Информатика»**

Тема: Алгоритмы и структуры данных в Python

|  |  |  |
| --- | --- | --- |
| Студент гр. 3341 |  | Костромитин М.М. |
| Преподаватель |  | Иванов Д.В. |

Санкт-Петербург

2024

## Цель работы

Реализация связного однонаправленного списка на языке Python с использованием двух зависимых классов Node и Linked List. В результате выполнения лабораторной работы должно быть создано и протестировано функционирование списка, включая добавление элементов, удаление элементов, поиск элементов и вывод всего списка в консоль.

## Задание

### Вариант 2

В данной лабораторной работе Вам предстоит реализовать связный однонаправленный список. Для этого необходимо реализовать 2 зависимых класса:

Node

Класс, который описывает элемент списка.

Он должен иметь 2 поля:

o data # Данные элемента списка, приватное поле.

o next # Ссылка на следующий элемент списка.

И следующие методы:

o \_\_init\_\_(self, data, next) - конструктор, у которого значения по умолчанию для аргумента next равно None.

o get\_data(self) - метод возвращает значение поля data (это необходимо, потому что в идеале пользователь класса не должен трогать поля класса Node).

o \_\_str\_\_(self) - перегрузка стандартного метода \_\_str\_\_, который преобразует объект в строковое представление. Для данной лабораторной необходимо реализовать следующий формат перевода объекта класса Node в строку:

“data: <node\_data>, next: <node\_next>”,

где <node\_data> - это значение поля data объекта Node, <node\_next> - это значение поля next объекта, на который мы ссылаемся, если он есть, иначе None.

Пример того, как должен выглядеть результат реализации \_\_str\_\_ см. ниже.

Пример того, как должен выглядеть вывод объекта:

node = Node(1)

print(node) # data: 1, next: None

node.next = Node(2, None)

print(node) # data: 1, next: 2

Linked List

Класс, который описывает связный однонаправленный список.

Он должен иметь 2 поля:

o head # Данные первого элемента списка.

o length # Количество элементов в списке.

И следующие методы:

o \_\_init\_\_(self, head) - конструктор, у которого значения по умолчанию для аргумента head равно None.

· Если значение переменной head равна None, метод должен создавать пустой список.

· Если значение head не равно None, необходимо создать список из одного элемента.

o \_\_len\_\_(self) - перегрузка метода \_\_len\_\_, он должен возвращать длину списка (этот стандартный метод, например, используется в функции len).

o append(self, element) - добавление элемента в конец списка. Метод должен создать объект класса Node, у которого значение поля data будет равно element и добавить этот объект в конец списка.

o \_\_str\_\_(self) - перегрузка стандартного метода \_\_str\_\_, который преобразует объект в строковое представление. Для данной лабораторной необходимо реализовать следующий формат перевода объекта класса однонаправленного списка в строку:

· Если список пустой, то строковое представление:

“LinkedList[]”

· Если не пустой, то формат представления следующий:

“LinkedList[length = <len>, [data:<first\_node>.data, next: <first\_node>.data; data:<second\_node>.data, next:<second\_node>.data; … ; data:<last\_node>.data, next: <last\_node>.data]”,

где <len> - длина связного списка, <first\_node>, <second\_node>, <third\_node>, … , <last\_node> - элементы однонаправленного списка.

Пример того, как должен выглядеть результат реализации см. ниже.

o pop(self) - удаление последнего элемента. Метод должен выбрасывать исключение IndexError с сообщением "LinkedList is empty!", если список пустой.

o clear(self) - очищение списка.

o delete\_on\_start(self, n) - удаление n-того элемента с НАЧАЛА списка. Метод должен выбрасывать исключение KeyError, с сообщением "Element doesn't exist!", если количество элементов меньше n.

Пример того, как должно выглядеть взаимодействие с Вашим связным списком:

linked\_list = LinkedList()

print(linked\_list) # LinkedList[]

print(len(linked\_list)) # 0

linked\_list.append(10)

print(linked\_list) # LinkedList[length = 1, [data: 10, next: None]]

print(len(linked\_list)) # 1

linked\_list.append(20)

print(linked\_list) # LinkedList[length = 2, [data: 10, next:20; data: 20, next: None]]

print(len(linked\_list)) # 2

linked\_list.pop()

print(linked\_list)

print(linked\_list) # LinkedList[length = 1, [data: 10, next: None]]

print(len(linked\_list)) # 1

## Выполнение работы

Шаги выполнения лабораторной работы:

1. Создание класса Node с полями data и next, и методами \_\_init\_\_, get\_data и str в соответствии с заданием.

2. Создание класса LinkedList с полями head и length, и методами \_\_init\_\_, len, append, \_\_str\_\_, pop, clear, delete\_on\_start в соответствии с заданием.

3. Реализация конструктор \_\_init\_\_ класса LinkedList для создания связанного списка.

4. Реализация метода len для определения длины списка.

5. Реализация метода append для добавления элемента в конец списка.

6. Реализация метода \_\_str\_\_ для представления списка в строковом формате.

7. Реализация метода pop для удаления последнего элемента списка.

8. Реализация метода clear для очистки списка.

9. Реализация метода delete\_on\_start для удаления n-того элемента с начала списка.

## Тестирование

Результаты тестирования представлены в Таблице 1.

Таблица 1.

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Входные данные | Выходные данные | Комментарии |
| 1 | node1 = Node(1)  node2 = Node(2)  node3 = Node(3)  llist = LinkedList(node1)  llist.append(node2)  llist.append(node3)  # Вывод длины списка  print(len(llist)) # 3  # Вывод списка в строковом формате  print(llist) # LinkedList[length = 3, [data:1, next:2; data:2, next:3; data:3, next:None]]  # Удаление последнего элемента  llist.pop()  print(llist) # LinkedList[length = 2, [data:1, next:2; data:2, next:None]]  # Удаление второго элемента  llist.delete\_on\_start(2)  print(llist) # LinkedList[length = 1, [data:1, next:None]]  # Очистка списка  llist.clear()  print(llist) # LinkedList[] | 3  LinkedList[length = 3, [data:1, next:2; data:2, next:3; data:3, next:None]]  LinkedList[length = 2, [data:1, next:2; data:2, next:None]]  LinkedList[length = 1, [data:1, next:None]]  LinkedList[] | Проверка работы всех основных методов |

## Выводы

В ходе выполнения данной лабораторной работы был создан и реализован класс Node для представления элемента списка и класс LinkedList для работы со связанным однонаправленным списком.

# Приложение А Исходный код программы

Исходный файл: main.py

class Node:

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

self.\_\_data = data

self.next = next

def get\_data(self):

return self.\_\_data

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

if self.next is not None:

return f"data: {self.\_\_data}, next: {(self.next).\_\_data}"

return f"data: {self.\_\_data}, next: None"

class LinkedList:

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

self.head = head

self.length = len(self)

def \_\_len\_\_(self):

length = 0

head = self.head

while head is not None:

length += 1

head = head.next

return length

def append(self, element):

if self.length == 0:

self.head = Node(element)

self.length = 1

return

current = self.head

while current.next is not None:

current = current.next

current.next = Node(element)

self.length += 1

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

if self.length == 0:

return "LinkedList[]"

elif self.length == 1:

if self.length == 1:

return f'LinkedList[length = {self.length}, [data: {self.head.get\_data()}, next: None]]'

else:

current = self.head

result = f'data: {current.get\_data()}, next: {current.next.get\_data()}'

current = current.next

while current.next is not None:

result = result + f'; data: {current.get\_data()}, next: {current.next.get\_data()}'

current = current.next

result = result + f'; data: {current.get\_data()}, next: None'

return f'LinkedList[length = {self.length}, [{result}]]'

def pop(self):

try:

if self.length == 0:

raise IndexError("LinkedList is empty!")

if self.length == 1:

self.head = None

else:

current = self.head

while current.next.next is not None:

current = current.next

current.next = None

self.length -= 1

except IndexError as error:

return error

def delete\_on\_start(self, n):

try:

if self.length < n or n < 1:

raise KeyError("OK")

if n == 1:

self.head = self.head.next

else:

current = self.head

for i in range(n - 2):

current = current.next

current.next = current.next.next

self.length -= 1

except KeyError as error:

raise KeyError

def clear(self):

self.head = None

self.length = 0