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:

- data # Данные элемента списка, приватное поле.
- next # Ссылка на следующий элемент списка.

И следующие методы:

- __init__(self, data, next) - конструктор, у которого значения по умолчанию для аргумента next равно None.
- get_data(self) - метод возвращает значение поля data (это необходимо, потому что в идеале пользователь класса не должен трогать поля класса Node).
- __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:

- head # Данные первого элемента списка.
- length # Количество элементов в списке.

И следующие методы:

- __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.
- __len__(self) - перегрузка метода __len__, он должен возвращать длину списка (этот стандартный метод, например, используется в функции len).
- 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.
- __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.

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