МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №2
по дисциплине «Информатика»
Тема: Алгоритмы и структуры данных в Python

Студент гр. 3342 _____ Корниенко А. Е.

Преподаватель _____ Иванов Д. В.

Санкт-Петербург
2024

Цель работы

Изучить алгоритмы и структуры данных и их реализацию на языке Python. С их помощью написать программу, создающую однонаправленный список.

Задание

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:

1) __init__(self, data, next) - конструктор, у которого значения по умолчанию для аргумента next равно None.

2) get_data(self) - метод возвращает значение поля data (это необходимо, потому что в идеале пользователь класса не должен трогать поля класса Node).

3) __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.

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:

1) __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.

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

3) 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.

4) __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.

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

6) clear(self) - очищение списка.

7) 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.

Выполнение работы

Node:

1) __init__ - Создается два поля экземпляра self.__data - приватное поле, хранившее данные; self.next - ссылка на следующий элемент класса Node.

2) get_data – геттер для self.__data.

3) __str__ - Вывод строкового представления экземпляра в соответсвии с шаблоном.

LinkedList:

1) __init__ - констуктор, если передаётся Node элемент, то он становится головой, иначе создаёт пустой LinkedList.

2) __len__ - Вывод длины списка.

3) append - Добавление нового элемента класса Node.

4) __str__ - Строковое представление списка в соответсвии с шаблоном. Циклом происходит проход по элементам и добавление к итоговое строке описание каждого элемента.

5) pop - Удаление последнего элемента из списка.

6) delete_on_start - По аналогии с pop. Только идет удаление по значению элемента.

7) clear - Очищение списка, используется pop().

Тестирование

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные
1.	linked_list = LinkedList() print(linked_list) print(len(linked_list)) linked_list.append(10) print(linked_list) print(len(linked_list)) linked_list.append(20) print(linked_list) print(len(linked_list)) linked_list.pop() print(linked_list) print(len(linked_list))	LinkedList[] 0 LinkedList[length = 1, [data: 10, next: None]] 1 LinkedList[length = 2, [data: 10, next:20; data: 20, next: None]] 2 LinkedList[length = 1, [data: 10, next: None]] 1

Выводы

Была разработана программа, содержащая классы элемента однонаправленного списка и сам список. Написаны методы для каждого из них и протестирована их работа.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: 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 None:
            return f"data: {self.__data}, next: None"
        else:
            return       f"data:       {self.__data},       next:
{self.next.get_data()}"


class LinkedList:
    def __init__(self, head=None):
        self.head = None
        if head is not None:
            self.length = 1
            self.head = Node(head)
        else:
            self.length = 0

    def __len__(self):
        return self.length

    def append(self, element):
        if self.length != 0:
            self.length += 1
            node = Node(element)
            current = self.head
            while current.next is not None:
                current = current.next
            current.next = node
        else:
            self.length += 1
            self.head = Node(element)

    def __str__(self):
        if self.length == 0:
            return "LinkedList[]"
        else:
            string = f"LinkedList[length = {self.length}, ["
            current = self.head
            while current is not None:
                if current.next is None:
                    string += f"data: {current.get_data()}, next:
None; "
                else:
```

```
                        string += f"data: {current.get_data()}, next:
{current.next.get_data()}; "
                    current = current.next

                string = string[:len(string) - 2]
                string += "]]"
                return string

        def pop(self):
            if self.length == 0:
                raise IndexError("LinkedList is empty!")
            else:
                if self.head.next is not None:
                    self.length -= 1
                    current = self.head
                    while current.next.next is not None:
                        current = current.next

                    current.next = None
                else:
                    self.head = None
                    self.length = 0

        def delete_on_start(self, n):
            if self.length < n or n <= 0:
                raise KeyError("Element doesn't exist!")
            elif n == self.length:
                self.pop()
            elif n == 1:
                self.length -= 1
                self.head = self.head.next
            else:
                self.length -= 1
                current = self.head
                for i in range(n - 2):
                    current = current.next

                current.next = current.next.next

        def clear(self):
            while self.length != 0:
                self.pop()
```