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 _____ Иванов Д. М.

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

Связный список - это структура данных, состоящая из узлов, где каждый узел содержит данные и ссылку (указатель) на следующий узел в списке. Последний узел может ссылаться на null, что означает конец списка.

Основные отличия связного списка от массива: хранение в памяти, доступ к элементам (не через индексы, а последовательно, начиная от головы), размер.

Рассмотрим алгоритм создания нужных нам классов.

Node:

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

2) get_data - Вывод поля self.__data.

3) __str__ - Вывод строкового представления экземпляра в соответсвии с шаблоном. Если следующий элемент не null, то передается значение через поле get_data.

LinkedList:

1) __init__ - Создание списка для данного экземпляра (self.lis). Если передается голова, класса Node, то он будет являться первым элементом списка.

2) __len__ - Вывод длины списка self.lis.

3) append - Добавление нового элемента класса Node. Помимо его добавления в сам список, в последнем элементе списка создается ссылка на этот элемент через поле next.

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. Помимо удаления из самого списка, менется поле ссыдки предыдущего элемента. Также идет проверка на пустоту списка.

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

7) clear - Очищение списка, путем создания пустого ([]).

Тестирование

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
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	Верный вывод

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 != None:
            return f"data: {self.__data}, next: {self.next.get_data()}"
        return f"data: {self.__data}, next: {self.next}"


class LinkedList:
    def __init__(self, head=None):
        if head == None:
            self.lis = []
        else:
            head_lis = Node(head)
            self.lis = [head_lis]

    def __len__(self):
        return len(self.lis)

    def append(self, element):
        el = Node(element)
        if len(self.lis) == 0:
            self.lis.append(el)
        else:
            self.lis[-1].next = el
            self.lis.append(el)

    def __str__(self):
        if len(self.lis) == 0:
            return "LinkedList[]"
        st = f"LinkedList[length = {len(self.lis)}, ["
        for i in self.lis:
            if i.next != None:
                st += f"data: {i.get_data()}, next: {i.next.get_data()}"
                st += '; '
            else:
                st += f"data: {i.get_data()}, next: {i.next}"
        st += ']]'
        return st

    def pop(self):
        if len(self.lis) == 0:
            raise IndexError("LinkedList is empty!")
        elif len(self.lis) == 1:
            el = self.lis[0].get_data()
            self.lis = []
```

```
            return el
        el = self.lis[-1].get_data()
        self.lis[-2].next = None
        self.lis.pop(len(self.lis) - 1)
        return el

    def delete_on_start(self, n):
        if len(self.lis) < n or n <= 0:
            raise KeyError("Element doesn't exist!")
        elif len(self.lis) == n:
            if n > 1:
                self.lis[n - 2].next = None
        else:
            if n > 1:
                self.lis[n - 2].next = self.lis[n]
        self.lis.pop(n - 1)

    def clear(self):
        self.lis = []
```