

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.Тест

Студент гр. 3341

Ягудин Д. Р.

Преподаватель

Иванов Д. В.

Санкт-Петербург

2024

Цель работы

Цель работы к данному заданию заключается в реализации связанного однонаправленного списка через создание двух зависимых классов: Node и LinkedList. Цель включает следующие задачи:

1. Создание класса Node, который описывает элемент списка с полями данных и ссылкой на следующий элемент, а также методами для инициализации объекта, получения данных и перегрузки метода str для удобного вывода информации объекта.

2. Создание класса LinkedList, который описывает связанный однонаправленный список с полями головного элемента и количеством элементов списка. Также требуется реализовать методы инициализации объекта, получения длины списка, добавления элемента в конец списка, перегрузки метода str для вывода списка в строковом представлении, удаления последнего элемента, очистки списка и удаления n-того элемента с начала списка.

Задание

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` равно `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.

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

- о `clear(self)` - очищение списка.

- о `delete_on_end(self, n)` - удаление `n`-того элемента с конца списка. Метод должен выбрасывать исключение `KeyError`, с сообщением `"Element doesn't exist!"`, если количество элементов меньше `n`.

Основные теоретические положения

Связанный список (LinkedList) - это структура данных, которая состоит из узлов, каждый из которых содержит данные и ссылку на следующий узел в списке. Основные положения о LinkedList включают:

1. Динамичность: LinkedList динамически растет и сжимается по мере добавления или удаления элементов, так как каждый узел содержит ссылку на следующий узел.

2. Быстрое добавление и удаление: Вставка или удаление элементов в LinkedList имеет константную сложность $O(1)$, в отличие от массива, где эти операции могут быть более сложными (в зависимости от позиции элемента).

3. Медленный доступ к элементам: Для доступа к элементам LinkedList нужно итерироваться с начала списка до нужного элемента, что делает доступ к элементу в LinkedList медленнее, чем в массиве.

4. Не непрерывная память: Узлы в LinkedList распределены в памяти не последовательно, поэтому нет гарантии, что они будут храниться в соседних ячейках памяти.

5. Внедрение: LinkedList обычно используется, когда необходимо часто добавлять и удалять элементы, и доступ по индексу не так важен.

Это основные положения о LinkedList как структуре данных.

Выполнение работы

Ход работы к данному коду можно описать следующим образом:

1. Определение класса Node:

- Создание класса Node с атрибутами data (данные элемента) и next (ссылка на следующий элемент).
- Определение метода `__init__()` для инициализации объекта класса Node с передачей данных и ссылки на следующий элемент.
- Определение метода `get_data()`, который возвращает данные элемента.
- Определение метода `__str__()`, который возвращает строковое представление данных и ссылки на следующий элемент объекта Node.

2. Определение класса LinkedList:

- Создание класса LinkedList с атрибутами head (головной элемент списка) и length (длина списка).
- Определение метода `__init__()` для инициализации объекта класса LinkedList, устанавливающего головной элемент и длину списка.
- Определение метода `__len__()`, который возвращает длину списка.
- Определение метода `__str__()`, который возвращает строковое представление списка.
- Определение метода `append()`, добавляющего новый элемент в конец списка.
- Определение метода `pop()`, удаляющего последний элемент из списка.
- Определение метода `delete_on_end()`, удаляющего n-тый элемент с конца списка.
- Определение метода `clear()`, очищающего список (удаляющего все элементы).

3. Дополнительные пояснения:

- Метод `__str__()` выводит список в красивом виде, обходя все элементы списка.
- Метод `append()` добавляет новый элемент, обходя список до конца.
- Метод `pop()` удаляет последний элемент, обходя список до предпоследнего.
- Метод `delete_on_end()` удаляет n-тый элемент, пересчитывая указатели.
- Метод `clear()` устанавливает `head` в `None` и длину в 0.

Тестирование

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	<pre>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</pre>	<pre>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]] LinkedList[length = 1, [data: 10, next: None]] 1</pre>	Проверка работы основных методов класса

Выводы

В рамках данной задачи была реализован связанный однонаправленный список через два класса: Node и LinkedList.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lb2.py

```
class Node:
    __data = None
    next = None
    def __init__(self, data, next = None):
        self.__data = data
        self.next = next

    def get_data(self):
        return self.__data

    def __str__(self):
        next_data = self.next.__data if self.next else None
        return f"data: {self.get_data()}, next: {next_data}"

node = Node(1)

class LinkedList:
    head = []
    length = 0
    def __init__(self, head = None):
        if head != None:
            self.head = head
            self.length += 1

    def __len__(self):
        return self.length

    def append(self, element):
        if self.head == []:
            self.head = Node(element)
        else:
            current = self.head
            while current.next:
                current = current.next
            current.next = Node(element)
            self.length += 1

    def __str__(self):
        if self.length == 0:
            return "LinkedList[]"
        str = f"LinkedList[length = {self.length},\n[{self.head.__str__()}]"
        current = self.head
        while current.next:
            current = current.next
            str += f"; {current.__str__()}"
        str += "]"
        return str

    def pop(self):
```

```

    if self.length == 0:
        raise IndexError("LinkedList is empty!")
    if self.length == 1:
        self.head = []
        self.length = 0
        return
    counter = 1
    current = self.head
    while counter != self.length - 1:
        current = current.next
        counter += 1
    current.next = None
    self.length -= 1

def delete_on_end(self, n):
    if self.length < n or n <= 0:
        raise KeyError("Element doesn't exist!")
    if self.length - n == 0:
        next_to_head = self.head.next
        self.head = next_to_head
        self.length -= 1
        return
    counter = 1
    current = self.head
    next_to_current = current.next
    while counter != self.length - n:
        current = next_to_current
        next_to_current = next_to_current.next
        counter += 1
    current.next = next_to_current.next
    self.length -= 1

def clear(self):
    self.head = []
    self.length = 0

```