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 _____ Романов А. К.

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, перегрузки метода str для вывода списка в строковом представлении, добавления элемента в конец списка, очистки списка и удаления n-того элемента с конца списка.

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:

- __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_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, состоящая 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 данные 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.

Выполнение работы

Ход работы к данному коду можно описать следующим образом:

1. Описание класса Node:

- Создание класса Node с приватным полем __data (данные элемента) и next (ссылка на следующий элемент).

- Определение метода __init__(). Присваиваются значения __data и next. (Параметр next при создании объекта класса опционален, и, в случае если он не подан, соответствующее поле инициализируется как None).

- Определение метода get_data. Возвращает значение приватного поля __data.

- Переопределение стандартного метода __str__(). Изменение вывода информации об объекте в соответствии с условием.

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. В случае если список инициализируется без указания головного элемента, полю head присваивается значение None.

- Определение метода len(), который возвращает длину списка. Сложность: 1

- Определение метода __str__(), который возвращает строковое представление списка в соответствии с условием.

- Определение метода append(), добавляющего новый элемент в конец списка. Сложность: n

- Определение метода pop(), удаляющего последний элемент из списка. Сложность: n.

- Определение метода delete_on_end(), удаляющего n-тый элемент с начала списка. Сложность: n.

- Определение метода clear(), очищающего список (полю head вновь присваивается значение None, длина приравнивается к нулю). Сложность: 1

Касательно реализации бинарного поиска в линейном списке: во-первых для этого список изначально должен создаваться так, чтобы элементы в нем были отсортированы, иначе бинарный поиск не имеет смсыла. Во-вторых, в линейном списке нельзя напрямую обратиться к элементу по индексу, как это можно сделать в обычном списке. Отсюда следует что для обращения к очередному элементу придется итерироваться по элементам между ним и предыдущим рассмотренным элементом. Однако в целом реализация алгоритма не будет отличаться, разве что необходимо добавить переменную, в которой будет храниться информация о том, какой элемент списка рассматриввается.

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) # 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	[] 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	Проверка работы основных методов класса

Выводы

Была изучены структура данных однонаправленный связный список, ее устройство применение, а также различные алгоритмы и методы работы, связанные с ней. Были освоены важные навыки работы с вышеописанной структурой данных.

В рамках данной задачи была реализован связанный однонаправленный список. Для этой цели был использован язык программирования «Python». Реализация связанного однонаправленного списка была осуществлена при помощи двух классов: Node и Linked List.

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):
        return f"data: {self.__data}, next: {self.next.__data if
(self.next is not None) else None}"


class LinkedList:

    def __init__(self, head=None):
        self.head = head
        self.length = 1

        if self.head is None:
            self.length = 0

    def __len__(self):
        return self.length

    def append(self, element):

        if self.head is None:
            self.head = Node(element)

        else:
            pointer = self.head

            while pointer.next is not None:
                pointer = pointer.next
            else:
                pointer.next = Node(element)

        self.length += 1

    def pop(self):

        if self.length == 0:
            raise IndexError("LinkedList is empty!")

        if self.length == 1:
            self.head = None

        else:
```

```
            pointer = self.head

            while pointer.next.next is not None:
                pointer = pointer.next
            else:
                pointer.next = None
        self.length -= 1

    def delete_on_end(self, n):

        if self.length == 0 or n > self.length or n <= 0:
            raise KeyError("Element doesn't exist!")

        pointer = self.head
        n = self.length - n
        number = 0

        if n == 0:
            self.head = self.head.next

        else:
            while number != n-1:
                pointer = pointer.next
                number += 1
            pointer.next = pointer.next.next

        self.length -= 1



    def clear(self):
        self.head = None
        self.length = 0

    def __str__(self):
        if self.length == 0:
            return "LinkedList[]"

        pointer = self.head
        info = ""
        while True:
            info += f"{pointer}; "
            if pointer.next is None:
                break
            pointer = pointer.next
        info = info[0:-2]

        return f"LinkedList[length = {self.length}, [{info}]]"
```