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

ОТЧЕТ

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

Студент гр. 3343 _____ Пименов П.В.

Преподаватель _____ Иванов Д.В.

Санкт-Петербург
2024

Цель работы

Изучить общие понятия об алгоритмах и структурах данных в Python, реализовать однонаправленный список на Python.

Задание

Вариант 1. Необходимо реализовать два 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.
- 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.
  - 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.

Выполнение работы

Требуемые в задании классы, а также их методы, были успешно реализованы. Ответы на вопросы:

- Что такое связный список? Основные отличия связного списка от массива.

  Связный список – структура данных, элементы которой – узлы – помимо конкретных данных содержат также и ссылки (или указатели) на следующий элемент списка. Основные отличия связного списка в том, что его элементы не располагаются в памяти последовательно (как в массиве), можно добавлять или удалять новые элементы. Кроме того, методы взаимодействия со связным списком отличаются по сложности от аналогичных методов взаимодействия с массивом.
- Указать сложность каждого метода

1. class Node
   1. __init__ – O(1)
   2. get_data – O(1)
   3. __str__ – O(1)
2. class LinkedList
   1. __init__ – O(1)
   2. __len__ – O(1)
   3. append – O(n)
   4. __str__ – O(n)
   5. pop – O(n)
   6. delete_on_end – O(n)
   7. clear – O(1)

8. __iter__ – O(1)
9. __next__ – O(1)

Примечание: сложность большинства из реализованных методов O(1), поскольку они, в большинстве своем, просто возвращают конкретные значения полей класса. Стоит обратить внимание на то, что сложность метода __str__ в классе LinkedList составляет O(n), что вызвано спецификой формата вывода, определенного в задании (необходимо составить строку из данных всех элементов списка, что нельзя сделать без его полного обхода).

- Описать возможную реализацию бинарного поиска в связном списке. Чем отличается реализация алгоритма бинарного поиска для связного списка и для классического списка Python?

  Реализация бинарного поиска в связном списке будет практически такая же, как и реализация в массиве. Отличие в том, что надо будет реализовать метод нахождения N-ного элемента списка, который, в худшем случае, будет работать за O(n). Это связано с тем, что работая с массивом, можно гораздо проще получить элемент в середине массива (просто обратившись по соответствующему индексу). В связном списке такой возможности нет.

  Разработанный программный код см. в приложении А.

Тестирование

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	l = LinkedList() l.append(1) l.append(2) l.append(3) print(l)	LinkedList[length = 3, [data: 1, next: 2; data: 2, next: 3; data: 3, next: None]]	Программа работает корректно
2.	l = LinkedList() l.append(1) l.append(2) l.append(3) l.pop() print(l)	LinkedList[length = 2, [data: 1, next: 2; data: 2, next: None]]	Программа работает корректно
3.	l = LinkedList() l.append(1) l.append(2) l.append(3) l.delete_on_end(2) print(l)	LinkedList[length = 2, [data: 1, next: 3; data: 3, next: None]]	Программа работает корректно
4.	l = LinkedList() l.append(1) l.append(2) l.append(3) l.clear() print(l)	LinkedList[]	Программа работает корректно

Выводы

Были изучены общие понятия об алгоритмах и структурах данных в Python, реализован однонаправленный список на Python.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.py

```
from itertools import islice


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.get_data()}, next: {self.next}'
                    return   f'data:   {self.get_data()},   next:
{self.next.get_data()}'


class LinkedList:
    def __init__(self, head=None):
        self.head = head
        self.length = int(head is not None)

    def __len__(self):
        return self.length

    def append(self, element):
        if len(self) == 0:
            self.head = Node(element)
        else:
            *rest, last = self  # PEP 448
            last.next = Node(element)
        self.length += 1

    def __str__(self):
        if len(self) == 0:
            return 'LinkedList[]'
                return  f"LinkedList[length  =  {len(self)},  [{';
'.join(map(str, self))}]]"

    def pop(self):
        if len(self) == 0:
            raise IndexError('LinkedList is empty!')
        elif len(self) == 1:
            self.head = None
        else:
            *rest, previous, last = self
            previous.next = None
```

```
            self.length -= 1

        def delete_on_end(self, n):
            if len(self) < n or n <= 0:
                raise KeyError("Element doesn't exist!")
            if n == 1:
                self.pop()
            elif n == len(self):
                second = self.head.next
                self.head = second
                self.length -= 1
            else:
                target_index = len(self) - n
                   first, mid, last = islice(self, target_index - 1,
target_index + 2)
                first.next = last
                self.length -= 1

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

        def __iter__(self):
            self.__current = self.head
            return self

        def __next__(self):
            if self.__current is None:
                raise StopIteration
            last = self.__current
            self.__current = self.__current.next
            return last
```