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. Вариант 2

Студент гр. 3343 _____ Атоян М. А.

Преподаватель _____ Иванов Д. В.

Санкт-Петербург
2024

Цель работы

Изучить основные особенности структур данных и методов работы с ними. Написать собственную практическую реализацию линейного односвязного списка на Python, используя ООП. Сравнить асимптотическую сложность операций над списком и над массивом.

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_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, которая состоит из узлов, где каждый узел содержит данные и ссылку на следующий узел в списке. Основное отличие связного списка от массива заключается в том, что связный список не требует непрерывной памяти для хранения элементов, в отличие от массива. Кроме того, связный список позволяет эффективно добавлять и удалять элементы из середины списка, так как не требует переноса всех элементов при изменении размера. Также связный список не обладает прямым доступом к произвольному элементу по индексу, в отличие от массива, где это возможно.

Сложности методов:

Класс Node:

1. init – O(1);
2. get_data – O(1);
3. str – O(1).

Класс LinkedList:

1. init – O(1);
2. len – O(1);
3. append – O(n);
4. str – O(n);
5. pop – O(n);
6. delete_on_start - O(n);
7. clear – O(1).

Анализ:

В случае классического списка Python, бинарный поиск выполняется путем разделения списка на две части и последующего сравнения искомого элемента с элементом в середине списка. Затем выполняется поиск в соответствующей половине списка в зависимости от результата сравнения.

В связном списке бинарный поиск немного сложнее из-за того, что к элементам списка можно получить доступ только последовательно, начиная с головы списка. Поэтому в бинарном поиске для связного списка сначала необходимо определить размер списка и затем выполнить поиск с учетом разделения списка наполовину. Однако такой подход не эффективен, так как он требует прохода по всему списку для нахождения размера.

Тестирование

Результаты тестирования содержатся в таблице 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]] LinkedList[length = 1, [data: 10, next: None]] 1	Программа сработала корректно.

Выводы

В ходе выполнения лабораторной работы были изучены и применены на практике алгоритмы и структуры данных в Python. Разработан односвязный линейный список с применением полученных знаний, реализованы методы для работы с ним.

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 else None}"


class LinkedList:
    def __init__(self, head = None):
        self.head = head
        self.length = 1 if head else 0

    def __len__(self):
        return self.length

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

    def __str__(self):
        current = self.head
        if (current == None):
            return "LinkedList[]"
```

```
        list_of_nodes = []
        while current:
            list_of_nodes.append(str(current))
            current = current.next
        return    f"LinkedList[length    =    {len(self)},    [{';
'.join(list_of_nodes)}]]"

    def pop(self):
        if (len(self) == 0):
            raise IndexError("LinkedList is empty!")
        current = self.head
        if (len(self) == 1):
            self.head = None
        else:
            while(current.next.next):
                current = current.next
            current.next = None
        self.length -= 1

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

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