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 _____ Львов А. В.

Преподаватель _____ Иванов Д. В.

Санкт-Петербург
2024

Цель работы

Ознакомление с алгоритмами и структурами данных и их реализация на языке Python, создание односвязного списка и функций для работы с ним.

Задание

Вариант 2

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.

Выполнение работы

Связный список - динамическая структура данных, состоящая из узлов, содержащих данные и ссылки на следующий и/или предыдущий узел списка.

Принципиальным преимуществом перед массивом является гибкость: порядок элементов связного списка может не совпадать с порядком расположения элементов данных в памяти компьютера.

Описание методов классов:

Node:

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.
3) __str__(self) - перегрузка стандартного метода __str__, который преобразует объект в строковое представление (в данной работе строка вида: «data: <node_data>, next: <node_next>».

LinkedList:

1) __init__(self, head) - конструктор, у которого значения по умолчанию для аргумента head равно None. Если значение переменной head равна None, метод создает пустой список.Если значение head не равно None, метод создаёт список из одного элемента. Сложность метода – O(1).
2) __len__(self) - перегрузка метода __len__, он возвращает длину списка. Сложность метода – O(n).
3) append(self, element) - добавление элемента в конец списка. Метод создаёт объект класса Node, у которого значение поля data равно element и добавляет этот объект в конец списка. Сложность метода – O(n).
4) __str__(self) - перегрузка стандартного метода __str__, который преобразует объект в строковое представление (в данной работе строка вида: «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]». Сложность метода - O(n).

5) pop(self) - удаление последнего элемента. Сложность метода – O(n).
6) clear(self) - очищение списка. Сложность метода – O(n).
7) delete_on_start(self, n) - удаление n-того элемента с начала списка. Сложность метода – O(n).

Исходный код программы см. в приложении А.

Тестирование

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные
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(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

Выводы

Было проведено ознакомление с такой структурой данных, как односвязный список, его реализация на языке Python.

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 is not None:
            return       f'data:       {self.get_data()},       next:
{self.next.get_data()}'
        else:
            return f'data: {self.get_data()}, next: {None}'


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

    def __len__(self):
        curr = self.head
        c = 0
        while curr is not None:
            c += 1
            curr = curr.next
        self.len = c
        return self.len

    def append(self, data):
        new_node = Node(data)
        curr = self.head
        if curr is not None:
            while curr.next is not None:
                curr = curr.next
            curr.next = new_node
        else:
            self.head = new_node
        self.len += 1

    def __str__(self):
        if self.head is not None:
            curr = self.head
            elements = []
            while curr is not None:
                elements.append(str(curr))
                curr = curr.next
            return    f'LinkedList[length    =    {len(self)},    [{";
".join(elements)}]]'
```

```
        else:
            return f'LinkedList[]'

    def pop(self):
        if self.len == 1:
            self.head = None
        elif self.head is not None:
            curr = self.head
            while curr.next.next is not None:
                curr = curr.next
            curr.next = None
        else:
            raise IndexError('LinkedList is empty!')
        self.len -= 1

    def clear(self):
        curr = self.head
        tmp = self.head
        while curr is not None:
            tmp = curr.next
            curr.next = None
            curr = tmp
        self.head = None
        self.len = 0

    def delete_on_start(self, n):
        if len(self) < n or n <= 0:
            raise KeyError("Element doesn't exist!")
        else:
            curr = self.head
            if n == 1:
                self.head = curr.next
            else:
                for _ in range(n - 2):
                    curr = curr.next
                curr.next = curr.next.next if curr.next is not None else
None
```