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

Студентка гр. 3342 _____ Смирнова Е.С.

Преподаватель _____ Иванов Д.В.

Санкт-Петербург

2024

Цель работы

Научиться работать со структурами данных в Python и реализовать однонаправленный список.

Задание

(Вариант 4)

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).
- **change_data(self, new_data)** - метод меняет значение поля data объекта 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.
- **pop(self)** - удаление последнего элемента. Метод должен выбрасывать исключение IndexError с сообщением "LinkedList is empty!", если список пустой.
- **clear(self)** - очищение списка.
- **change_on_start(self, n, new_data)** - изменение поля data n-того элемента с НАЧАЛА списка на new_data. Метод должен выбрасывать исключение KeyError, с сообщением "Element doesn't exist!", если количество элементов меньше n.

Основные теоретические положения

1. Список – структура данных, в которой каждый элемент (узел), помимо самих данных, содержит указатель на следующий элемент (односвязный список) или на следующий и предыдущий (двусвязный список). Тем самым список отличается от массива (данные идут не подряд, нельзя прямо обратиться к любому элементу списка, нужно до него «дойти», в то время как в массиве достаточно обратиться по адресу, что займёт O(n) времени).

2. Класс Node:

• __init__ — O(1);

• get_data — O(1);

• __str__ — O(1).

Класс LinkedList:

• __init__ — O(n);

• __len__ — O(1);

• append — O(n);

• __str__ — O(n);

• pop — O(n);

• change_on_start — O(n);

• clear — O(1).

3. Бинарный поиск в связном списке может быть реализован следующим образом:

1) Найти длину списка.

2) Найти середину списка.

3) Сравнить искомое значение с значением в середине списка.

4) Если искомое значение меньше значения в середине списка, повторить шаги 2-3 для левой половины списка.

5) Если искомое значение больше значения в середине списка, повторить шаги 2-3 для правой половины списка.

6) Если искомое значение равно значению в середине списка, вернуть индекс этого элемента.

Отличия от реализации бинарного поиска для классического списка Python заключаются в том, что для связного списка нельзя получить доступ к элементам по индексу за константное время, как в случае с классическим списком Python.

Выполнение работы

В ходе выполнения работы были реализованы 2 класса: класс Node, в котором осуществляется описание элемента связанного списка, а также класс LinkedList, описывающий связный однонаправленный список.

В классе Node присутствуют следующие методы: __init__(), который является конструктором. В нем объявлены поля data(приватное поле) и next(ссылка на следующий элемент списка). Также в классе реализован метод get_data() для доступа к полю data, метод change_data() для изменения значения этого поля, а также определен метод __str__().

В классе LinkedList также реализован конструктор с 2-мя приватными полями: head(указатель на первый элемент связного списка) и length(длина списка). Далее определен метод __len__(), который возвращает конечную длину списка. Далее реализован метод append, который добавляет новый элемент в конец списка. Следующий метод __str__(), в нем все элементы списка записываются в строку, после чего, с помощью конкатенации строк получается конечный результат. Далее представлена реализация метода pop() для удаления элемента в конце списка. Здесь используется перебор всех элементов списка, необходимо найти предпоследний элемент, чтобы в указатель на следующий элемент записать None. В классе LinkedList реализованы еще 2 метода: change_on_start() и clear(). Метод clear() полностью очищает список. В поле head записывается None, а длина становится равной 0. С помощью метода change_on_start() осуществляется замена значения элемента списка под определенным номером с начала. В начале происходит проверка на существование данного элемента и вызов соответствующего исключения, если такого элемента нет. Потом находится нужный элемент, у которого вызывается метод change_data().

Разработанный программный код см. в приложении А.

Выводы

В ходе выполнения данной лабораторной работы была освоена работа с связным однонаправленным списком. Также были проанализированы методы списка и определена сложность для каждого из них.

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 change_data(self, new_data):
        self.data = new_data

    def __str__(self):
        if self.next is None:
            return f'data: {self.get_data()}, next: None'
        else:
            return f'data: {self.get_data()}, next:
{self.next.get_data()}'


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

    def __len__(self):
        return self.length

    def append(self, element):
        self.length += 1
        new_element = Node(element, None)
        head = self.head
        if head:
            while head.next:
                head = head.next
            head.next = new_element
        else:
            self.head = new_element

    def __str__(self):
        if self.head is not None:
            tmp = self.head
            element = f'{tmp}'
            while tmp.next is not None:
                tmp = tmp.next
                element += "; " + tmp.__str__()
            return f'LinkedList[length = {self.length}, [{element}]]'
        else:
            return f'LinkedList[]'
```

```
def pop(self):
    if self.length == 1:
        self.length -= 1
        self.head = None
    elif self.length != 0:
        tmp = self.head
        while tmp.next.next is not None:
            tmp = tmp.next
        tmp.next = None
        self.length -= 1
    else:
        raise IndexError("LinkedList is empty!")

def change_on_start(self, n, new_data):
    if self.length >= n and n > 0:
        tmp = self.head
        for i in range (n - 1):
            tmp = tmp.next
        tmp.change_data(new_data)
    else:
        raise KeyError("Element doesn't exist!")

def clear(self):
    self.head = None
    self.lenght = 0
```