

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

ОТЧЕТ
по лабораторной работе №4
по дисциплине «Введение в ИТ»
Тема: Алгоритмы и структуры данных в Python

Студент гр. 9383

Ноздрин В.Я.

Преподаватель

Размочаева Н.В.

Санкт-Петербург

2019

1 Цель работы.

Реализовать двунаправленный список на языке **Python**. Реализовать классы «элемент списка» и сам класс двунаправленного списка.

2 Задание.

В данной лабораторной работе Вам предстоит реализовать связный двунаправленный список.

Node

Класс, который описывает элемент списка.

Класс Node должен иметь 3 поля:

`__data` # данные, приватное поле

`__prev__` # ссылка на предыдущий элемент списка

`__next__` # ссылка на следующий элемент списка

Вам необходимо реализовать следующие методы в классе Node:

`__init__(self, data, prev, next)` - конструктор, у которого значения по умолчанию для аргументов `prev` и `next` равны `None`.

`get_data(self)` - метод возвращает значение поля `__data`.

`__str__(self)` - перегрузка метода `__str__`. Описание того, как должен выглядеть результат вызова метода смотрите ниже в примере взаимодействия с Node.

Пример того, как должен выглядеть вывод объекта:

```
node = Node(1)
```

```
print(node) # data: 1, prev: None, next: None
```

```
node.__prev__ = Node(2, None, None)
```

```
print(node) # data: 1, prev: 2, next: None
```

```
node.__next__ = Node(3, None, None)
```

```
print(node) # data: 1, prev: 2, next: 3
```

Linked List

Класс, который описывает связный двунаправленный список.

Класс LinkedList должен иметь 3 поля:

`__length` # длина списка

`__first__` # данные первого элемента списка

`__last__` # данные последнего элемента списка

Вам необходимо реализовать конструктор:

`__init__(self, first, last)` - конструктор, у которого значения по умолчанию для аргументов `first` и `last` равны `None`.

Если значение переменной `first` равно `None`, а переменной `last` не равно `None`, метод должен вызывать исключение `ValueError` с сообщением: "invalid value for last".

Если значение переменной `first` не равно `None`, а переменной `last` равна `None`, метод должен создавать список из одного элемента. В данном случае, `first` равен `last`, ссылки `prev` и `next` равны `None`, значение поля `__data` для элемента списка равно `first`.

Если значения переменных не равны `None`, необходимо создать список из двух элементов. В таком случае, значение поля `__data` для первого элемента списка равно `first`, значение поля `__data` для второго элемента списка равно `last`.

`__len__(self)` - перегрузка метода `__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 метода `__str__`. Описание того, как должен выглядеть результат вызова метода смотрите ниже в примере взаимодействия с `LinkedList`.

`pop(self)` - удаление последнего элемента. Метод должен выбрасывать исключение `IndexError` с сообщением "LinkedList is empty!", если список пустой.

`popitem(self, element)` - удаление элемента, у которого значение поля `__data` равно `element`. Метод должен выбрасывать исключение `KeyError`, с сообщением "<element> doesn't exist!", если элемента в списке нет.

`clear(self)` - очищение списка.

Пример того, как должно выглядеть взаимодействие с Вашим связным списком:

```
linked_list = LinkedList()
print(linked_list) # LinkedList[]
print(len(linked_list)) # 0
linked_list.append(10)
print(linked_list) # LinkedList[length = 1, [data: 10, prev: None, next:
None]]
print(len(linked_list)) # 1
linked_list.append(20)
print(linked_list) # LinkedList[length = 2, [data: 10, prev: None, next: 20;
data: 20, prev: 10, next: None]]
print(len(linked_list)) # 2
linked_list.pop()
print(linked_list) # LinkedList[length = 1, [data: 10, prev: None, next:
None]]
print(len(linked_list)) # 1
```

3 Выполнение работы.

Связный список – это структура данных в информатике, основной чертой которой является ее реализация: список состоит из элементов, которые в свою очередь хранят свои данные и две ссылки – на предыдущий и на следующий элемент.

Основным отличием связного списка от классического массива является невозможность обращения к элементам списка по индексам. Помимо этого, элементы связного списка могут располагаться в памяти свободно и размер элементов может быть любым.

Сложность методов:

init – $O(1)$	len – $O(1)$	append – $O(1)$
str – $O(n)$	pop – $O(1)$	popitem – $O(n)$
clear – $O(1)$		

Бинарный поиск в связном списке реализовать невозможно, так как невозможно обратиться напрямую к элементу списка по индексу, а значит реализация бинарного поиска будет сводиться к переводу связного списка в стандартный массив и далее к бинарному поиску в массиве. Что неэффективно, ведь перевод связного списка в массив по сложности равен поиску перебором.

Разработанный программный код см. в приложении А.

4 Выводы.

Была изучена конструкция связного двунаправленного списка.

Была разработана система классов, имитирующая работу со связным списком.

1 ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.py

-*- coding: utf-8 -*-

```
class Node:
```

```
    """
```

```
    Поля:
```

```
    __data    # данные, приватное поле
```

```
    __prev__  # ссылка на предыдущий элемент списка
```

```
    __next__  # ссылка на следующий элемент списка
```

```
    """
```

```
    def __init__(self, data, prev=None, next_=None):
```

```
        self.__data = data # данные, приватное поле
```

```
        self.__prev__ = prev # ссылка на предыдущий элемент списка
```

```
        self.__next__ = next_ # ссылка на следующий элемент списка
```

```
    def get_data(self): # метод возвращает значение поля __data.
```

```
        return self.__data
```

```
    def __str__(self):
```

```
        prev = self.__prev__.get_data() if self.__prev__ is not None else "None"
```

```
        next_ = self.__next__.get_data() if self.__next__ is not None else "None"
```

```
        return 'data: {}, prev: {}, next: {}'.format(self.__data, prev, next_)
```

```
    def set_next(self, next_):
```

```
        if self is not None:
```

```
            self.__next__ = next_
```

```
    def get_next(self):
```

```
        if self is not None:
```

```
            return self.__next__
```

```
    def set_prev(self, prev):
```

```
        if self is not None:
```

```
            self.__prev__ = prev
```

```
def get_prev(self):
    if self is not None:
        return self.__prev__
```

```
class LinkedList:
```

```
    """
```

Класс, который описывает связный двунаправленный список.

Класс LinkedList должен иметь 3 поля:

```
    __length    # длина списка
    __first__   # данные первого элемента списка
    __last__    # данные последнего элемента списка
    """
```

```
def __init__(self, first=None, last=None):
```

```
    if first is None:
```

```
        if last is not None:
```

```
            raise ValueError("invalid value for last")
```

```
        # LinkedList[len = 0, [prev: None, next: None, data: first]]
```

```
        else:
```

```
            self.__length = 0
```

```
            self.__first__ = None
```

```
            self.__last__ = None
```

```
        else:
```

```
            # LinkedList[len = 1, [prev: None, next: None, data: first]]
```

```
            if last is None:
```

```
                self.__length = 1
```

```
                self.__first__ = Node(first)
```

```
                self.__last__ = self.__first__
```

```
            # LinkedList[len = 2, [prev: None, next: last, data: first; prev: first, next: None, data:
```

```
last]]
```

```
            else:
```

```
                self.__length = 2
```

```
                self.__first__ = Node(first)
```

```
                self.__last__ = Node(last, prev=self.__first__)
```

```
                self.__first__.set_next(self.__last__)
```

```
def __len__(self):
```

```

return self.__length

def append(self, element):
    if element is not None:
        new_node = Node(element, prev=self.__last__)
        if self.__length != 0:
            self.__last__.set_next(new_node)
            self.__last__ = new_node
        else:
            self.__first__ = new_node
            self.__last__ = new_node
        self.__length += 1

def __str__(self):
    if self.__length == 0:
        return 'LinkedList[]'

    length = 'length = ' + str(self.__length)
    elements = []
    current_node = self.__first__
    while current_node is not None:
        elements.append(current_node)
        current_node = current_node.get_next()
    elements = map(str, elements)
    return 'LinkedList[' + length + ', [' + '; '.join(elements) + ']' + ']'

def pop(self):
    if self.__length == 0:
        raise IndexError("LinkedList is empty!")
    self.__length -= 1
    self.__last__ = self.__last__.get_prev()
    self.__last__.set_next(None)

def popitem(self, element):
    if self.__length == 0:
        raise KeyError(f'{element} doesn\'t exist!')
    current_node = self.__first__
    new_len = self.__length

```



```

while current_node is not None:
    if current_node.get_data() == element: # prev -- element -- next
        new_len -= 1
        prev = current_node.get_prev()
        if prev is None:
            self.__first__ = current_node.get_next()
        else:
            prev.set_next(current_node.get_next())
        next_ = current_node.get_next()
        if next_ is None:
            self.__last__ = current_node.get_prev()
        else:
            next_.set_prev(current_node.get_prev())
        break
    current_node = current_node.get_next()
if new_len == self.__length:
    raise KeyError(f'{element} doesn\'t exist!')
self.__length = new_len

```

```

def clear(self):
    self.__length = 0
    self.__first__ = None
    self.__last__ = None

```

```

linked_list = LinkedList()
print(linked_list) # LinkedList[]
print(len(linked_list)) # 0

```

```

linked_list.append(10)
print(linked_list) # LinkedList[length = 1, [data: 10, prev: None, next: None]]
print(len(linked_list)) # 1

```

```

linked_list.append(20)
print(linked_list) # LinkedList[length = 2, [data: 10, prev: None, next: 20; data: 20, prev: 10,
next: None]]
print(len(linked_list)) # 2

```

```
linked_list.popitem(5)
print(linked_list) # LinkedList[length = 1, [data: 10, prev: None, next: None]]
print(len(linked_list)) # 1
```