**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №4**

**по дисциплине «Информатика»**

Тема: **Парадигмы программирования**

|  |  |  |
| --- | --- | --- |
| Студент гр. 8381 |  | Почаев Н.А. |
| Преподаватель |  | Размочаева Н.В. |

Санкт-Петербург

2018

**Цель работы**

Реализовать собственные классы и на их основе научиться строить такую структуру данных, как двунаправленный связный список. Научиться перегружать методы классов и работать с их атрибутами. Изучить основы ООП на языке Python и его особенности. Освоить работы с исключениями: «бросать» и «ловить» их.

**Задание**

Реализуйте два класса: класс Node и класс LinkedList, представляющие собой элемент и связный двунаправленный список.

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, со значениями по умолчанию для переменных prev и next None.

* getData(self)

метод возвращает значение поля \_\_data

* \_\_str\_\_(self)

перегрузка метода \_\_str\_\_.

Класс LinkedList должен иметь 3 поля:

* \_\_length # длина списка
* first # данные первого элемента списка
* last # данные последнего элемента списка

Вам необходимо реализовать следующие методы в классе LinkedList:

* \_\_init\_\_(self, first, last)

конструктор, со значениями по умолчанию для переменных first и last 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\_\_.

* 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.

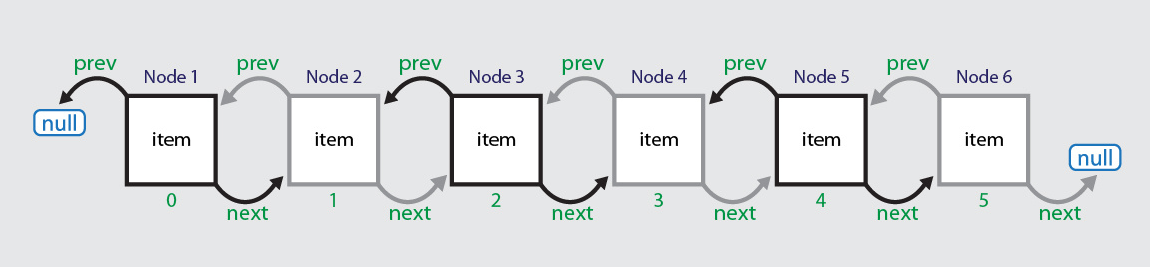
* clear(self)

очищение списка.

**Выполнение работы**

Написание работы производилось на базе операционной системы Linux Arch в интегрированной среде разработки PyCharm.

Ход выполнения работы:

1. В начале был создан класс Node, представляющий собой класс узла – контейнера, хранящего значение элемента и ссылки на следующий/предыдущий элементы списка. По умолчанию поле Data является приватным (в функциональности Python её имя искажено).
2. В данном классе были созданы: конструктор, геттер и переопределение функции str. Первый - специальный блок инструкций, вызываемый при создании объекта. Второй - специальный метод, позволяющий получить данные, доступ к которым напрямую ограничен, в данном случае, поля Data. Переопределение же метода str позволяет преобразовывать объект в строку вида: data: \_\_, prev: \_\_, next: \_\_.
3. Далее был создан класс LinkedList представляющий собой реализацию реализацию двунаправленного списка. Общая схема его работы приведена на рисунке:

Данный класс содержит поля: ссылка на первый/последний элементы списка и его длину (также является приватным полем).

1. В данном классе переопределён метод len, он возвращает текущую длину списка.
2. Конструктор данного класса имеет несколько путей работы: в случае некорректных входных значений (first=None и last != None) он кидает исключение ValueError("invalid value for last"). В других случаях в зависимости от входных значений, он создаёт список из одного/двух элементов.
3. Далее был переопределён метод str для приведения списка к строковому виду. В начале идёт проверка на ненулевую длину, затем в цикле в результирующий список записываются приведённую к строковому представлению элементы двунаправленного списка (т.е. узлы класса Node). Возвращаемое значение генерируется путём конкатенации требуемых элементов строки и содержимого списка res путём применения метода join.
4. Для очистки списка в ходе работы создан отдельный метод clean.
5. Также был создан метод append, позволяющий добавлять в список новое числовое значение. Если длина текущего списка равна 0, то новый элемент становится и последним и первым, иначе он встривается в двусвязный список, где предыдущий элемент начинает указывать на него, а он, в свою очередь, становится последним элементом, указывающим на предыдущий элемент и на Null.
6. В данном классе реализовано два метода для удаления элементов: pop – для удаления элементов из конца (спереди) списка и popitem – для удаления элемента из любой части списка.

Описание первого: в начале идёт на ненулевую длину списка, иначе «бросается» исключение IndexError("LinkedList is empty!"). Далее, если длина равна 1, то ссылки на следующий и предыдущий элементы становятся равными None , а длина уменьшается на единицу (т.е. до нуля). В ином случае переназначаем последний элемент, как последний, его ссылка на следующий становится None и длина также уменьшается.

Описание второго: также происходит проверка на ненулевую длину списка, иначе «бросается» исключение KeyError(str(element) + " doesn't exist!"). Оно также вызывается в случае отстутсвия необходимого для удаления элемента в поле Data хотя бы одного элемента (удаляется первый встретившийся). Если список состоит из одного элемента, то он очищается. Далее отдельно обрабатываются случаи, когда удаляемый элемент является первым/последним (происходит соостветсвенное перевешивание ссылок). Иначе просто перевешиваются ссылки на следующий/предыдущий элементы соседних элементов, минуя текущий, тем самым удаляя элемент.

**Выводы**

В ходе лабораторной работы были изучены основыреализации ООП на языке программирования Python. Были получены знания по перегрузке методов, созданию «приватных» полей и т.д. Также была освоена работа с исключениями. В результате был реализован двусвязный список, содержащий основные методы для работы с ним и исключения для корректной работы с ним.

ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД ПРОГРАММЫ

# класс узла - контейнера, хранящего значение и ссылку на след/пред эл.

class Node:

\_\_data = None # искажение имени (приватное поле)

prev = None

next = None

def \_\_init\_\_(self, data, prev=None, next=None):

self.\_\_data = data

self.prev = prev

self.next = next

def getData(self):

return self.\_\_data

# переопределение метода пребразования объекта в строку [data: \_\_, prev: \_\_, next: \_\_]

def \_\_str\_\_(self):

out\_for\_s = "data: " + str(self.getData()) # юзаем геттер

out\_for\_prev = ", prev: " + (str(self.prev.getData()) if self.prev is not None else "None")

out\_for\_next = ", next: " + (str(self.next.getData()) if self.next is not None else "None")

return out\_for\_s + out\_for\_prev + out\_for\_next

'''

Каждый узел содержит элемент, ссылку на предыдущий и следующий узел.

Связанный список состоит из последовательности узлов,

каждый из которых предназначен для хранения объекта определенного при создании типа.

Также хранит свою длину.

'''

class LinkedList:

\_\_length = 0

first = None

last = None

# переопределение метода получения длины

def \_\_len\_\_(self):

return self.\_\_length

def \_\_init\_\_(self, first=None, last=None):

if first is None:

if last is not None:

raise ValueError("invalid value for last")

else:

f\_node = Node(first)

self.first = f\_node

if last is None:

self.last = f\_node # список из одного элемента

self.\_\_length = 1

else:

l\_node = Node(last, self.first) # список двунаправный -> ссылка на пред.

self.first.next = l\_node # ссылка на предыдущий эл.

self.last = l\_node

self.\_\_length = 2

'''преобразование объекта к строковому представлению.

Вызывается, когда объект передается функциям print() и str()'''

def \_\_str\_\_(self):

res = []

if len(self) == 0:

return "LinkedList[]"

else:

now\_el = self.first # берём первый эл.

while now\_el != self.last:

res.append(str(now\_el)) # приводим элемент к строковому представлению

now\_el = now\_el.next

res.append(str(now\_el)) # последний эл.

return "LinkedList[length = " + str(len(self)) + ", [" + "; ".join(res) + "]]"

# очистка списка

def clear(self):

self.first = None

self.last = None

self.\_\_length = 0

# метод добавления в список одного узлового эл. спереди

def append(self, element):

new\_el = Node(element, self.last) # передаём в генератор класса значение Data и ссылку на последний. эл.

if len(self) == 0:

self.first = new\_el

else:

self.last.next = new\_el # ссылка последнего элемента на следующий

self.last = new\_el

self.\_\_length += 1

# метод удаления одного узлового эл. спереди списка

def pop(self):

if len(self) == 0:

raise IndexError("LinkedList is empty!")

elif len(self) == 1:

self.first.prev = None

self.first.next = None

self.\_\_length -= 1

elif len(self) > 1:

self.last = self.last.prev # перевешиваем последий элемент на предпоследний

self.last.next = None # новый последний эл. указывает на None

self.\_\_length -= 1

def popitem(self, element):

now\_el = self.first

if len(self) == 0:

raise KeyError(str(element) + " doesn't exist!")

# находим элемент с need Data

while now\_el.getData() != element:

if now\_el == self.last:

raise KeyError(str(element) + " doesn't exist!")

now\_el = now\_el.next

if len(self) == 1:

self.clear()

self.\_\_length = 0

# если первый элемент содержит Data

elif now\_el == self.first:

self.first = now\_el.next # перенавешиваем указатель на след. эл.

self.first.prev = None # у ного первого эл. указатель на предыдущий None

self.\_\_length -= 1

# если последний элемент содержит Data

elif now\_el == self.last:

self.last = self.last.prev # перенавешиваем указатель на предыдущий эл.

self.last.next = None # у последнего элемента указатель на след. None

self.\_\_length -= 1

else:

now\_el.prev.next = now\_el.next # указатель на следующий эл. предыдущего эл. перенавешиваем, минуя текущий

now\_el.next.prev = now\_el.prev # указатель на предыдущий эл. след. эл перенавешиваем , минуя текущий

self.\_\_length -= 1