**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.

|  |  |  |
| --- | --- | --- |
| Студент гр. 3344 |  | Клюкин А.В. |
| Преподаватель |  | Иванов Д.В. |

Санкт-Петербург

2024

## Цель работы

Научиться реализовывать односвязный список с использованием классов на языке программирования Python.

## Задание.

В данной лабораторной работе Вам предстоит реализовать связный однонаправленный список. Для этого необходимо реализовать 2 зависимых класса:

Node

Класс, который описывает элемент списка.

Он должен иметь 2 поля:

o data # Данные элемента списка, приватное поле.

o next # Ссылка на следующий элемент списка.

И следующие методы:

o \_\_init\_\_(self, data, next) - конструктор, у которого значения по умолчанию для аргумента next равно None.

o get\_data(self) - метод возвращает значение поля data (это необходимо, потому что в идеале пользователь класса не должен трогать поля класса Node).

o change\_data(self, new\_data) - метод меняет значение поля data объекта Node.

o \_\_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:

o head # Данные первого элемента списка.

o length # Количество элементов в списке.

И следующие методы:

o \_\_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.

o \_\_len\_\_(self) - перегрузка метода \_\_len\_\_, он должен возвращать длину списка (этот стандартный метод, например, используется в функции len).

o 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.

o \_\_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.

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

o clear(self) - очищение списка.

o 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.

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

## Выполнение работы

1. Связный список – структура, в которой каждый элемент содержит ссылку на следующий или даже предыдущий, если список двунаправленный. Отличие в том, что массив располагает свои данные последовательно в памяти, а список – нет. Так же массив хранит один тип данных, а список не обязательно.
2. get\_data – O(1), change\_data – O(1), append() – O(n) (“в худшем случае”), pop() – O(n) (“в худшем случае”), change\_on\_start() – O(n) (“в худшем случае”), clear() – O(1)
3. Берется серединный элемент списка и сравнивается с искомым. Если подходит, то конец, иначе и исходя из сравнения берется соседний справа или слева элемент и снова сравнивается.

Отличие реализации для классического списка Python в том, что можно использовать индексы для доступа к элементам списка, а в связном списке нет.

## Выводы

Получен навык реализации односвязного списка с использованием классов.

# Приложение А Исходный код программы

Название файла: Klyukin\_Aleksandr\_lb2.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 != None:

return f'data: {self.data}, next: {self.next.data}'

else:

return f'data: {self.data}, next: None'

class LinkedList:

def \_\_init\_\_(self, head=None):

self.head = head

self.length = 0

if head:

self.length += 1

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

return self.length

def append(self, element):

if self.head:

last = self.head

while last.next != None:

last = last.next

last.next = Node(element)

else:

self.head = Node(element)

self.length += 1

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

if self.length == 0:

return "LinkedList[]"

else:

out\_data = ''

node = self.head

while node != None:

out\_data += node.\_\_str\_\_() + '; '

node = node.next

return f"LinkedList[length = {self.length}, [{out\_data[:-2]}]]"

def pop(self):

if not self.head:

raise IndexError("LinkedList is empty!")

if self.head:

last = self.head

if self.head.next:

while last.next.next != None:

last = last.next

last.next = None

self.length-=1

else:

self.head = None

self.length = 0

def change\_on\_start(self, n, new\_data):

if self.length < n or n <= 0:

raise KeyError("Element doesn't exist!")

node = self.head

i = 1

while i < n:

node = node.next

i+=1

node.data = new\_data

def clear(self):

self.head = None

self.length = 0