**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

|  |  |  |
| --- | --- | --- |
| Студент гр. 3344 |  | Пачев Д.К. |
| Преподаватель |  | Иванов Д.В. |

Санкт-Петербург

2024

**Цель работы**

Написать программу на языке Python, реализующую связный однонаправленный список.

**Задание**

**Вариант 2.**

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   **\_\_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   **delete\_on\_start(self, n)**- удаление n-того элемента с НАЧАЛА списка. 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:*

linked\_list = LinkedList()

print(linked\_list) # LinkedList[]

print(len(linked\_list)) # 0

linked\_list.append(10)

print(linked\_list) # LinkedList[length = 1, [data: 10, next: None]]

print(len(linked\_list)) # 1

linked\_list.append(20)

print(linked\_list) # LinkedList[length = 2, [data: 10, next:20; data: 20, next: None]]

print(len(linked\_list)) # 2

linked\_list.pop()

print(linked\_list)

print(linked\_list) # LinkedList[length = 1, [data: 10, next: None]]

print(len(linked\_list)) # 1

## **Выполнение работы**

Класс Node – класс для представления элементов связного списка. Внутри реализованы:

* метод \_\_init\_\_() для инициализации полей \_\_data и next
* метод get\_data() для получения поля \_\_data
* метод change\_data() для изменения поля \_\_data
* а также переопределен метод \_\_str\_\_()

Класс LinkedList – класс для реализации связного однонаправленного списка.

Внутри реализованы:

* метод \_\_init\_\_() для инициализации полей head и length
* переопределен метод \_\_len\_\_(), который возвращает поле length
* метод append() – добавляет элемент в список с помощью цикла while
* метод pop() – удаляет последний элемент с помощью цикла for
* метод clear() – очищает список, устанавливая нулевую длину списка и присваивая голове списка значения None
* метод delete\_on\_start() – удаляет n-тый элемент с начала списка с помощью цикла for
* переопределен метод \_\_str\_\_() для представления списка в строковом формате

1. Указать, что такое связный список. Основные отличия связного списка от массива.

Связный список - это линейная структура данных, где элементы, или узлы, связаны с помощью указателей. Основные отличия между связным списком и массивом включают:

1. Выделение памяти: Для связных списков не требуется выделение непрерывной области памяти, в отличие от массивов, которым нужны смежные блоки памяти.
2. Динамический размер: Связные списки могут легко менять размер во время выполнения программы, в то время как у массивов размер фиксирован после выделения.
3. Вставка и удаление: Вставка или удаление элементов в связном списке эффективна, особенно в середине, т.к. она включает обновление указателей, в отличие от массивов, где такие операции могут быть затратными.
4. Время доступа: Массивы обеспечивают доступ к элементам по индексам за постоянное время, в то время как в связных списках для доступа к конкретному элементу нужно пройти от головы, что приводит к медленным временам доступа для произвольного доступа.
5. Указать сложность каждого метода.

O(1):

\_\_init\_\_()

get\_data()

change\_data()

Node.\_\_str\_\_()

\_\_len\_\_

clear()

O(n):

append()

pop()

delete\_on\_start()

LinkedList.\_\_str\_\_()

1. Описать возможную реализацию бинарного поиска в связном списке. Чем отличается реализация алгоритма бинарного поиска для связного списка и для классического списка Python?
   1. Установим указатели на начало и конец интервала поиска (начало - голова списка, конец - последний элемент).
   2. На каждом шаге бинарного поиска вычисляем среднюю точку текущего интервала.
   3. Сравниваем значение средней точки с ключом поиска и соответственно сужаем интервал, двигая указатели.
   4. Повторяем шаги 3-4 до тех пор, пока ключ не будет найден или интервал сужен до одного элемента.

Отличие от реализации бинарного поиска в классическом списке Python заключается в том, что для связного списка нет прямого доступа по индексам, и поэтому каждый шаг поиска требует перехода от головы списка к середине с учетом указателей. Это делает реализацию более сложной и требующей дополнительных операций по сравнению с обычным списком Python, где можно просто обращаться к элементам по индексу для бинарного поиска

**Тестирование**

Результаты тестирования представлены в Таблице 1

Таблица 1 - Результаты тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Входные данные | Выходные данные | Комментарии |
| 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(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 | верно |

**Выводы**

В ходе выполнения лабораторной работы была разработана программа на языке Python, которая реализует связный однонаправленный список.

# 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, data):  
 self.\_\_data = data  
  
 def \_\_str\_\_(self):  
 if self.next is not None:  
 return f'data: {self.\_\_data}, next: {self.next.get\_data()}'  
 return f'data: {self.\_\_data}, next: {self.next}'  
  
  
class LinkedList:  
 def \_\_init\_\_(self, head=None):  
 self.head = head  
 self.length = 0  
  
 def \_\_len\_\_(self):  
 return self.length  
  
 def append(self, element):  
 node = Node(element)  
 if self.head is None:  
 self.head = node  
 self.length += 1  
 else:  
 item = self.head  
 while item:  
 if item.next is None:  
 item.next = node  
 self.length += 1  
 break  
 item = item.next  
  
 def pop(self):  
 if self.length == 0:  
 raise IndexError('LinkedList is empty!')  
 self.length -= 1  
 if self.length == 0:  
 self.head = None  
 else:  
 current = self.head  
 for i in range(self.length - 1):  
 current = current.next  
 current.next = None  
  
 def clear(self):  
 self.head = None  
 self.length = 0  
  
 def delete\_on\_start(self,n):  
 if n > self.length or n < 1:  
 raise KeyError("Element doesn't exist!")  
 if n == 1:  
 self.head = self.head.next  
 else:  
 current = self.head  
 for i in range(n - 2):  
 current = current.next  
 current.next = current.next.next  
 self.length -= 1  
  
 def \_\_str\_\_(self):  
 if self.length == 0:  
 return 'LinkedList[]'  
 items = []  
 item = self.head  
 while item:  
 items.append(f'data: {item.get\_data()}, next: {item.next.get\_data() if item.next else None}')  
 item = item.next  
 return f"LinkedList[length = {self.length}, [{'; '.join(items)}]]"