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. 3341 _____ Шуменков А.П.

Преподаватель _____ Иванов Д.В.

Санкт-Петербург

2024

Цель работы

Целью данной работы является:

- изучение алгоритмов и структур данных
- реализовать связный однонаправленный список на языке Python

Задание

Вариант 4

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:

- 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.

впвпавпвапва

- __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:
  - 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.

- 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

Связный список - это динамическая структура данных, состоящая из узлов, содержащих данные и ссылки на следующий и/или предыдущий узел списка. Принципиальным преимуществом перед массивом является структурная гибкость: порядок элементов связного списка может не совпадать с порядком расположения элементов данных в памяти компьютера, а порядок обхода списка всегда явно задаётся его внутренними связями.

Преимущества связны списков:

- эффективное (за константное время) добавление и удаление элементов
- размер ограничен только объёмом памяти компьютера и разрядностью указателей
- динамическое добавление и удаление элементов

Выполнение работы

1. Класс Node:

Метод init(self, data, next=None):

- Этот метод инициализирует объект класса Node с атрибутами data и next.

- При создании нового узла устанавливается значение data и, по умолчанию, следующий узел next равен None. Метод get_data(self):

- Этот метод возвращает значение атрибута data текущего узла. Метод change_data(self, new_data):

- Этот метод изменяет значение атрибута data текущего узла на new_data. Метод __str(self):

- Этот метод возвращает строковое представление текущего узла, отображая данные текущего узла и данные следующего узла (если он существует).

- Если атрибут next равен None, то возвращается None вместо данных следующего узла.

- Обратите внимание, что неправильно определен магический метод __str, должно быть __str__.

2. Класс LinkedList:

Метод __init__(self, head=None):

- Данный метод предназначен для инициализации объекта класса LinkedList с атрибутами head (голова связанного списка) и length (длина списка).

- Если передается начальный узел head, то длина списка устанавливается в 1, иначе в 0. Метод len(self):

- Этот метод возвращает текущую длину связанного списка. Метод append(self, element):

- Метод добавляет новый узел со значением element в конец связанного списка.

- Если список пустой (головной узел head равен None), то новый узел становится головным.

- В противном случае новый узел присоединяется к концу списка путем прохождения по узлам до последнего и обновления его указателя next.

- После добавления узла увеличивается длина списка. Метод str(self):

- Этот метод возвращает строковое представление связанного списка.

- Если список пустой, возвращается строка "LinkedList".

- В противном случае формируется строка, отображающая текущую длину списка и все его элементы (узлы) последовательно. Метод pop(self):

- Метод удаляет последний узел из связанного списка.

- Если список пустой, генерируется исключение IndexError.

- Если длина списка равна 1, головной узел head просто заменяется на None.

- В случае более длинного списка, происходит перебор узлов до предпоследнего и обрыв связи с последним узлом.

- После удаления узла длина списка уменьшается. Метод change_on_start(self, n, new_data):

- Метод изменяет данные узла связанного списка по его порядковому номеру n (считая с 1) на new_data.

- Если указанный номер n выходит за пределы длины списка или меньше 1, генерируется исключение KeyError.

- Происходит проход до указанного узла и изменение его данных методом change_data узла. Метод clear(self):

- Метод очищает связанный список путем установки головного узла head в None и обнуления длины списка.

Разработанный программный код см. в приложении А.

Тестирование

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	node = Node(7) print(node) node.next = Node(5, None) print(node) print(node.get_data()) node.change_data(10) print(node.get_data()) print(node)	data: 7, next: None data: 7, next: 5 7 10 data: 10, next: 5	Проверка методов класса Node
2.	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.change_on_start(2, 7) print(linked_list) linked_list.pop() print(linked_list) linked_list.clear() print(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 = 2, [data: 10, next: 7; data: 7, next: None]] LinkedList[length = 1, [data: 10, next: None]] LinkedList[]	Проверка методов класса LinkedList
3.	linked_list = LinkedList() linked_list.append(10) linked_list.change_on_start(2, 10)	KeyError: "Element doesn't exist!"	Проверка исключения метода change_on_start

Выводы

В ходе выполнения лабораторной работы были изучены основные алгоритмы и структуры данных. Была написана программа на языке Python, реализующая связный однонаправленный список с помощью двух зависимых классов Node и LinkedList.

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):
        return f"data: {self.get_data()}, next: {None if self.next
== None else self.next.get_data()}"


class LinkedList:
    def __init__(self, head = None):
        self.head = head
        if head is not None:
            self.length = 1
        else:
            self.length = 0

    def __len__(self):
        return self.length

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

    def __str__(self):
        if self.length == 0:
            return "LinkedList[]"
        else:
            result = f"LinkedList[length = {self.length}, ["
            cur = self.head
            while(cur != None):
                result += str(cur)
                cur = cur.next
                if(cur != None):
                    result += "; "
            result += "]]"
        return result
```

```
def pop(self):
    current_node = self.head
    if current_node is None:
        raise IndexError("LinkedList is empty!")
    elif self.length == 1:
            self.head = None
            self.length = 0
    elif self.length >= 2:
            while current_node.next.next:
                current_node =current_node.next
            current_node.next = None
            self.length -= 1
def change_on_start(self, n, new_data):
    if n > self.length or n <= 0:
        raise KeyError("Element doesn't exist!")
    current = self.head
    for i in range(n-1):
        current = current.next
    current.change_data(new_data)

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