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 в Python.

Студентка гр. 3341 _____ Кузнецова С.Е.

Преподаватель _____ Иванов Д.В.

Санкт-Петербург
2024

Цель работы

Цель – знакомство с алгоритмами и структурами данных.

Задачи:

1. Изучить вопросы оценки сложности алгоритмов, включая рассмотрение разных типов сложностей.

2. Рассмотреть различные структуры данных.

3. Познакомиться с алгоритмами сортировок, алгоритмом бинарного поиска.

4. Написать программу на языке Python с реализацией однонаправленного списка с помощью классов.

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.

- __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.

- 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

Сложность алгоритма – это определенная характеристика алгоритма, которая вычисляется сравнением двух алгоритмов на идеальном (абстрактном, математическом) уровне, игнорируя низкоуровневые детали (язык программирования, физические характеристики компьютера, на котором запущена программа, набор команд в данном CPU и пр.). Целесообразно сравнивать алгоритмы с точки зрения того, чем они являются по своей сути, а именно: идеи, как происходит вычисление, количество операций в зависимости от входных данных и т. д.

Коллекциями называют специальные хранилища объектов. Коллекции предоставляют доступ к своим элементам, а также удобные интерфейсы для их обработки. В языке Python встроенными коллекциями являются:

- Строка.
- Список.
- Словарь.
- Кортеж.
- Множество.

При этом коллекции можно поделить на 3 категории:

– отображения (словарь);

– последовательности (список, кортеж, строка);

– множества (множество).

Ответы на вопросы:

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

Связный список (linked list) — это структура данных, состоящая из узлов, каждый из которых содержит данные и ссылку (указатель) на следующий узел в списке. Связные списки могут быть однонаправленными (когда узлы имеют ссылку только на следующий узел) или двунаправленными (когда узлы имеют ссылки и на следующий, и на предыдущий узлы).

Основные отличия связного списка от массива:

- Вставка и удаление: Вставка и удаление элементов в связном списке происходит быстрее, чем в массиве, так как не требуется смещать остальные элементы.
- Доступ к элементам: Для доступа к элементам массива можно использовать прямой доступ по индексу, в то время как для доступа к элементам связного списка нужно пройти по всем узлам от начала списка до нужного элемента.
- Память: При использовании массива элементы хранятся в памяти непрерывно, то есть рядом друг с другом. При использовании связного списка элементы могут размещаться где угодно в памяти.
- Динамичность: Связные списки позволяют динамически изменять размер структуры данных, добавляя или удаляя элементы, в то время как массив имеет фиксированный размер.

2. Указать сложность каждого метода.

Преобразование объекта в строковое представление __str__(), добавление элемента в конец списка append(), удаление последнего элемента pop() и замена данных в n-ном элементе с начала списка change_on_start() имеют сложность O(n), т.к. требуют итерации по всему списку.

Конструктор списка __init__(), получение длины списка __len__() и очищение списка clear() имеют сложность O(1), т.к. не зависят от количества элементов в списке.

3. Описать возможную реализацию бинарного поиска в связном списке. Чем отличается реализация алгоритма бинарного поиска для связного списка и для классического списка Python?

В случае связного списка бинарный поиск может быть реализован следующим образом:

1. Начнем с указателей на начало и конец списка.

2. Найдем середину списка, перемещая указатель на середину (используя данные о длине списка).

3. Сравним значение искомого элемента с элементом в середине.

4. Если значение равно, вернем индекс элемента; если значение меньше, продолжим поиск в левой половине списка, обновив указатель на конец на элемент перед серединой; если значение больше, продолжим поиск в правой половине списка, обновив указатель на начало на элемент после середины.

5. Повторим шаги до тех пор, пока не найдем искомый элемент или не дойдем до конца списка.

Отличие реализации бинарного поиска для связного списка от классического списка Python заключается в том, что для связного списка нет прямого доступа к элементам по индексу. Поэтому при реализации бинарного поиска в связном списке нужно указатели для перемещения по элементам списка.

Выполнение работы

1. Создаем класс Node, который описывает элемент списка. Имеет два поля – приватное поле data (данные элемента списка) и next (ссылка на следующий элемент). Также имеет методы:

- __init__(self, data, next) - конструктор, у которого значения по умолчанию для аргумента next равно None.
- get_data(self) - метод возвращает значение поля data.
- change_data(self, new_data) - метод меняет значение поля data объекта Node.
- __str__(self) - перегрузка стандартного метода __str__, который преобразует объект в строковое представление.

2. Создаем класс Linked list, который описывает связный однонаправленный список. Имеет два поля – head (данные первого элемента списка), length (количество элементов в списке). Также имеет методы:

- __init__(self, head) - конструктор, у которого значения по умолчанию для аргумента head равно None. Если значение переменной head равна None, метод создает пустой список, иначе создает список из одного элемента.
- __len__(self) - возвращает длину списка.
- append(self, element) - добавление элемента в конец списка. Метод создает объект класса Node, у которого значение поля data будет равно element и добавляет этот объект в конец списка.
- __str__(self) - преобразует объект в строковое представление в определенном формате, заданном в задании работы.
- pop(self) - удаляет последний элемент. Метод выбрасывает исключение IndexError с сообщением "LinkedList is empty!", если список пустой.
- clear(self) - очищает список.
- change_on_start(self, n, new_data) - изменяет поля data n-того элемента с начала списка на new_data. Метод выбрасывает исключение

KeyError с сообщением "Element doesn't exist!", если количество элементов меньше n.

Тестирование

Результаты тестирования представлены в табл. 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	Пример взаимодействия со связным списком и методов append и pop
2.	linked_list = LinkedList() linked_list.append(10) print(linked_list) print(len(linked_list)) linked_list.pop() print(linked_list) print(len(linked_list)) linked_list.pop() print(linked_list) print(len(linked_list))	LinkedList[length = 1, [data: 10, next: None]] 1 LinkedList[] 0 IndexError: LinkedList is empty!	Проверка выбрасывания исключения IndexError
3.	linked_list = LinkedList() linked_list.append(10) linked_list.append(20) print(linked_list) print(len(linked_list)) linked_list.change_on_start(	LinkedList[length = 2, [data: 10, next: 20; data: 20, next: None]] 2 LinkedList[length = 2, [data: 200, next: 20; data: 20, next:	Проверка метода change_on_start

	1, 200) print(linked_list) print(len(linked_list))	None]] 2	
4.	linked_list = LinkedList() linked_list.append(10) linked_list.append(20) print(linked_list) print(len(linked_list)) linked_list.change_on_start(3, 200) print(linked_list) print(len(linked_list))	LinkedList[length = 2, [data: 10, next: 20; data: 20, next: None]] 2 KeyError: "Element doesn't exist!"	Проверка выбрасывания исключения KeyError
5.	linked_list = LinkedList() linked_list.append(10) linked_list.append(20) print(linked_list) print(len(linked_list)) linked_list.clear() print(linked_list) print(len(linked_list))	LinkedList[length = 2, [data: 10, next: 20; data: 20, next: None]] 2 LinkedList[] 0	Проверка метода clear

Выводы

Познакомились с алгоритмами и структурами данных.

Были достигнуты поставленные задачи:

1. Изучили вопросы оценки сложности алгоритмов, включая рассмотрение разных типов сложностей.

2. Рассмотрели различные структуры данных.

3. Познакомились с алгоритмами сортировок, алгоритмом бинарного поиска.

4. Написали программу на языке Python с реализацией однонаправленного списка с помощью классов.

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 "data: " + str(self.get_data()) + ", next: " +
str(self.next.get_data() if self.next != None else None)


class LinkedList:
    def __init__(self, head=None):
        self.head = head
        self.length = 1 if head != None else 0

    def __len__(self):
        return self.length

    def append(self, element):
        elem = Node(element)
        if (self.length > 0):
            curr = self.head
            while (curr.next != None):
                curr = curr.next
            curr.next = elem
        else:
            self.head = elem
        self.length += 1

    def __str__(self):
        if (self.length == 0):
            return "LinkedList[]"
        else:
            curr = self.head
            result = "LinkedList[length = " + str(self.length) + ",
["
            while (curr != None):
                result += str(curr) + "; "
                curr = curr.next
            result = result[:-2]
            result += "]]"
            return result

    def pop(self):
```

```
    if (self.length == 0) :
        raise IndexError("LinkedList is empty!")
    elif (self.length == 1):
        self.head = None
        self.length = 0
    else:
        curr = self.head
        while (curr.next.next != None):
            curr = curr.next
        curr.next = None
        self.length -= 1

def change_on_start(self, n, new_data):
    if (self.length < n or n < 1):
        raise KeyError("Element doesn't exist!")
    else:
        curr = self.head
        for i in range(n-1):
            curr = curr.next
        curr.change_data(new_data)

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