Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Российский химико-технологический университет имени Д.И. Менделеева»

Кафедра информационных компьютерных технологий

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 3

Выполнил студент группы КС-36 Кошкарев Иван Михайлович

Ссылка на репозиторий: https://github.com/MUCTR-IKT-CPP/KoshkarevIM\_36\_ALG/tree/main/Lab1

Приняли: Крашенинников Роман Сергеевич

Дата сдачи: 10.03

Оглавление

[Описание задачи. 2](#_Toc63548272)

[Описание метода/модели. 2](#_Toc63548273)

[Выполнение задачи. 2](#_Toc63548274)

[Заключение. 2](#_Toc63548275)

# Описание задачи.

Реализовать модель двусвязного списка, а также провести с ней серию тестов использовать для сортировки коллекции чисел.

# Описание метода/модели.

Список – это структура данных, в которой каждый элемент связан с одним из своих соседних элементов, для однонаправленного списка это следующий элемент.

Для двунаправленного списка это следующий и предыдущий элемент.

Структура одного элемента списка:

Двунаправленный – поле указателя на предыдущий, поле значения, поле указателя на последующий

Однонаправленный – поле значения элемента, поле указателя на последующий

Имея такую структуру список позволяет легко проводить операции вставки и удаления, так же в зависимости от реализации списки могут поддерживать обращения по индексу, хотя в базовой модели такого не предполагается.

# Выполнение задачи.

Для реализации модели и тестов я использовал язык Python из-за его простой архитектуры.

import random  
from datetime import datetime, timedelta  
import time  
  
  
# Узел двусвязного списка  
class Node:  
 def \_\_init\_\_(self, data):  
 self.data = data # Данные, которые хранятся в узле  
 self.prev = None # Ссылка на предыдущий узел  
 self.next = None # Ссылка на следующий узел  
  
  
# Двусвязный список  
class DoublyLinkedList:  
 def \_\_init\_\_(self):  
 self.head = None # Ссылка на первый узел  
 self.tail = None # Ссылка на последний узел  
 self.\_length = 0 # Количество элементов в списке  
  
 def is\_empty(self):  
 #Проверяет, пуст ли список.  
 return self.\_length == 0  
  
 def \_\_len\_\_(self):  
 #Возвращает количество элементов в списке.  
 return self.\_length  
  
 def append(self, data):  
 #Добавляет элемент в конец списка.  
 new\_node = Node(data)  
 if self.is\_empty(): # Если список пуст  
 self.head = new\_node  
 self.tail = new\_node  
 else:  
 new\_node.prev = self.tail  
 self.tail.next = new\_node  
 self.tail = new\_node  
 self.\_length += 1  
  
 def prepend(self, data):  
 #Добавляет элемент в начало списка.  
 new\_node = Node(data)  
 if self.is\_empty(): # Если список пуст  
 self.head = new\_node  
 self.tail = new\_node  
 else:  
 new\_node.next = self.head  
 self.head.prev = new\_node  
 self.head = new\_node  
 self.\_length += 1  
  
 def insert\_at(self, data, index):  
 #Добавляет элемент в произвольное место по индексу.  
 if index < 0:  
 raise ValueError("Индекс не может быть отрицательным")  
  
 if index == 0: # Если вставляем в начало  
 self.prepend(data)  
 return  
 elif index >= self.\_length: # Если вставляем в конец  
 self.append(data)  
 return  
  
 new\_node = Node(data)  
 current = self.head  
 current\_index = 0  
  
 # Ищем узел, перед которым нужно вставить новый элемент  
 while current and current\_index < index:  
 current = current.next  
 current\_index += 1  
  
 # Вставляем новый узел между current.prev и current  
 new\_node.prev = current.prev  
 new\_node.next = current  
 current.prev.next = new\_node  
 current.prev = new\_node  
 self.\_length += 1  
  
 def delete\_at(self, index):  
 #Удаляет элемент по индексу.  
 if self.is\_empty():  
 raise IndexError("Список пуст")  
  
 if index < 0 or index >= self.\_length:  
 raise IndexError("Индекс выходит за пределы списка")  
  
 current = self.head  
 current\_index = 0  
  
 # Ищем узел для удаления  
 while current and current\_index < index:  
 current = current.next  
 current\_index += 1  
  
 if current.prev: # Если узел не является головой  
 current.prev.next = current.next  
 else: # Если узел — это голова  
 self.head = current.next  
  
 if current.next: # Если узел не является хвостом  
 current.next.prev = current.prev  
 else: # Если узел — это хвост  
 self.tail = current.prev  
  
 self.\_length -= 1  
  
 def \_\_iter\_\_(self):  
 #Возвращает итератор для списка.  
 current = self.head  
 while current:  
 yield current.data  
 current = current.next  
  
 def \_\_str\_\_(self):  
 #Возвращает строковое представление списка.  
 return " <-> ".join(map(str, self)) + " <-> None"  
  
  
# Структура для хранения данных о человеке  
class Person:  
 def \_\_init\_\_(self, last\_name, first\_name, patronymic, birth\_date):  
 self.last\_name = last\_name  
 self.first\_name = first\_name  
 self.patronymic = patronymic  
 self.birth\_date = birth\_date  
  
 def \_\_str\_\_(self):  
 return f"{self.last\_name} {self.first\_name} {self.patronymic} ({self.birth\_date.strftime('%d.%m.%Y')})"  
  
  
# Генерация случайной даты  
def generate\_random\_date(start, end):  
 #Генерирует случайную дату между start и end.  
 delta = end - start  
 random\_days = random.randint(0, delta.days)  
 return start + timedelta(days=random\_days)  
  
  
# Тест 1: Заполнение контейнера 1000 целыми числами и подсчет суммы, среднего, минимального и максимального  
def test\_integer\_operations():  
 dll = DoublyLinkedList() # Создаем двусвязный список  
 numbers = [random.randint(-1000, 1000) for \_ in range(1000)] # Генерируем 1000 чисел  
  
 # Заполняем список  
 for number in numbers:  
 dll.append(number)  
  
 # Подсчет суммы, среднего, минимального и максимального  
 total\_sum = sum(dll)  
 average = total\_sum / len(dll)  
 min\_value = min(dll)  
 max\_value = max(dll)  
  
 print(f"Сумма: {total\_sum}")  
 print(f"Среднее: {average}")  
 print(f"Минимальное: {min\_value}")  
 print(f"Максимальное: {max\_value}")  
  
  
# Тест 2: Проверка операций вставки и удаления на коллекции из 10 строковых элементов  
def test\_string\_operations():  
 dll = DoublyLinkedList() # Создаем двусвязный список  
 strings = ["apple", "banana", "cherry", "date", "elderberry", "fig", "grape", "honeydew", "kiwi", "lemon"]  
  
 # Заполняем список  
 for s in strings:  
 dll.append(s)  
  
 # Вставляем элемент в середину  
 dll.insert\_at("mango", 5) # Вставляем "mango" на позицию 5  
  
 # Удаляем элемент по индексу  
 dll.delete\_at(2) # Удаляем "cherry"  
  
 # Выводим результат  
 print("Список после операций:")  
 print(dll)  
  
  
# Тест 3: Заполнение контейнера 100 структур (ФИО и дата рождения) и фильтрация  
def test\_person\_filter():  
 dll = DoublyLinkedList() # Создаем двусвязный список  
 last\_names = ["Иванов", "Петров", "Сидоров", "Кузнецов", "Смирнов"]  
 first\_names = ["Алексей", "Дмитрий", "Ольга", "Екатерина", "Иван"]  
 patronymics = ["Иванович", "Дмитриевич", "Олеговна", "Егоровна", "Алексеевич"]  
  
 # Генерируем 100 случайных людей  
 start\_date = datetime(1980, 1, 1)  
 end\_date = datetime(2020, 1, 1)  
 for \_ in range(100):  
 last\_name = random.choice(last\_names)  
 first\_name = random.choice(first\_names)  
 patronymic = random.choice(patronymics)  
 birth\_date = generate\_random\_date(start\_date, end\_date)  
 person = Person(last\_name, first\_name, patronymic, birth\_date)  
 dll.append(person)  
  
 # Фильтрация: люди младше 20 и старше 30 лет  
 young\_people = DoublyLinkedList()  
 old\_people = DoublyLinkedList()  
 current\_date = datetime.now()  
  
 for person in dll:  
 age = (current\_date - person.birth\_date).days // 365  
 if age < 20:  
 young\_people.append(person)  
 elif age > 30:  
 old\_people.append(person)  
  
 # Проверка правильности фильтрации  
 print(f"Людей младше 20 лет: {len(young\_people)}")  
 print(f"Людей старше 30 лет: {len(old\_people)}")  
  
  
# Тест 4: Сортировка выбором на двусвязном списке  
def selection\_sort(dll):  
 current = dll.head  
 while current:  
 min\_node = current  
 next\_node = current.next  
 while next\_node:  
 if next\_node.data < min\_node.data:  
 min\_node = next\_node  
 next\_node = next\_node.next  
 if current != min\_node:  
 current.data, min\_node.data = min\_node.data, current.data  
 current = current.next  
  
  
def test\_selection\_sort():  
 #Создает и сортирует 20 списков разного размера, измеряя время выполнения.  
 sizes = [1000, 2000, 4000, 8000, 16000]  
 num\_tests = 20  
 for size in sizes:  
 total\_time = 0  
 for \_ in range(num\_tests):  
 dll = DoublyLinkedList()  
 for \_ in range(size):  
 dll.append(random.randint(-10000, 10000))  
 start\_time = time.time()  
 selection\_sort(dll)  
 end\_time = time.time()  
 total\_time += (end\_time - start\_time)  
 avg\_time = total\_time / num\_tests  
 print(f"Среднее время сортировки списка размером {size}: {avg\_time:.5f} секунд")  
  
  
# Тест 5: Перемешивание элементов списка  
def shuffle\_list(dll):  
 nodes = []  
 current = dll.head  
 while current:  
 nodes.append(current)  
 current = current.next  
  
 random.shuffle(nodes)  
  
 # Восстанавливаем связи  
 dll.head = nodes[0]  
 dll.tail = nodes[-1]  
 dll.head.prev = None  
 dll.tail.next = None  
  
 for i in range(len(nodes)):  
 if i > 0:  
 nodes[i].prev = nodes[i - 1]  
 if i < len(nodes) - 1:  
 nodes[i].next = nodes[i + 1]  
  
  
def test\_shuffle():  
 dll = DoublyLinkedList() # Создаем двусвязный список  
 numbers = [random.randint(-1000, 1000) for \_ in range(10)] # Генерируем 10 чисел  
  
 # Заполняем список  
 for number in numbers:  
 dll.append(number)  
  
 print("До перемешивания:")  
 print(dll)  
  
 # Перемешиваем список  
 shuffle\_list(dll)  
  
 print("После перемешивания:")  
 print(dll)  
  
  
# Запуск всех тестов  
if \_\_name\_\_ == "\_\_main\_\_":  
 print("Тест 1: Список с числами")  
 test\_integer\_operations()  
  
 print("\nТест 2: Список со строками")  
 test\_string\_operations()  
  
 print("\nТест 3: Фильтрация людей")  
 test\_person\_filter()  
  
 print("\nТест 4: Сортировка выбором")  
 test\_selection\_sort()  
  
 print("\nТест 5: Перемешивание элементов")  
 test\_shuffle()

***Результаты:***

Тест 1: Список с числами

Сумма: -10373

Среднее: -10.373

Минимальное: -999

Максимальное: 1000

Тест 2: Список со строками

Список после операций:

apple <-> banana <-> date <-> elderberry <-> mango <-> fig <-> grape <-> honeydew <-> kiwi <-> lemon <-> None

Тест 3: Фильтрация людей

Людей младше 20 лет: 38

Людей старше 30 лет: 35

Тест 4: Сортировка выбором списка:

Среднее время сортировки списка размером 1000: 0.03392 секунд

Среднее время сортировки списка размером 2000: 0.13388 секунд

Среднее время сортировки списка размером 4000: 0.53494 секунд

Среднее время сортировки списка размером 8000: 2.13904 секунд

Среднее время сортировки списка размером 16000: 8.44633 секунд

Сортировка выбором массива:

Среднее время сортировки массива размером 1000: 0.029275 секунд

Среднее время сортировки массива размером 2000: 0.118932 секунд

Среднее время сортировки массива размером 4000: 0.487384 секунд

Среднее время сортировки массива размером 8000: 1.903711 секунд

Среднее время сортировки массива размером 16000: 7.076226 секунд

Тест 5: Перемешивание элементов

До перемешивания:

277 <-> -521 <-> 843 <-> 170 <-> 515 <-> -248 <-> -540 <-> -173 <-> -917 <-> -863 <-> None

После перемешивания:

515 <-> -521 <-> -540 <-> -863 <-> 277 <-> 843 <-> -173 <-> 170 <-> -917 <-> -248 <-> None

# Заключение.

Списки это важный инструмент для работы с памятью, который лучше, чем массивы подходит для больших данных, в силу более экономного использования памяти, возможности хранить список в "разных местах" памяти(распределение), благодаря ссылкам в узлах, и меньшему времени, которое тратится на удаление и добавление элементов.