МИНОБРНАУКИ РОССИИ
Санкт-Петербургский государственный
электротехнический университет
«ЛЭТИ» им. В.И. Ульянова (Ленина)
Кафедра МО ЭВМ

Отчет
по лабораторной работе №1
по дисциплине «Алгоритмы и структуры данных»
Тема: Развернутый связный список

Студент гр. 3382 _____ Копасова К. А.

Преподаватель _____ Иванов Д. В.

Санкт-Петербург
2024

Цель работы

Изучить развернутые и линейные списки и научится оценивать алгоритмы по времени их выполнения и по затраченной памяти посредством реализации развернутого списка с определенными методами на языке программирования Python.

Задание

Развёрнутый связный список — список, каждый физический элемент которого содержит несколько логических элементов (обычно в виде массива, что позволяет ускорить доступ к отдельным элементам).

Данная структура позволяет значительно уменьшить расход памяти и увеличить производительность по сравнению с обычным списком. Особенно большая экономия памяти достигается при малом размере логических элементов и большом их количестве.

У данной структуры необходимо реализовать основные операции: поиск, удаление, вставка, а также функцию вывода всего списка в консоль через пробел. В качестве элементов для заполнения используются целые числа. Функция вычисления размера node находится в следующем блоке заданий. Реализацию поиска и удаления делать на свое усмотрение. Данные операции будут проверяться на защите.

Для проверки работоспособности структуры необходимо реализовать функцию (не метод класса) *check*, принимающую на вход два массива: массив *arr_1* для заполнения структуры, массив *arr_2* для поиска и удаления, а также необязательный параметр *n_array* (описан выше). Функция должна сначала заполнять развернутый связный список данным *arr_1*, затем искать элементы *arr_2* и удалять их. После каждой операции по обновлению списка необходимо осуществлять полный его вывод в консоль.

Помимо реализации описанного класса Вам необходимо провести исследование его работы: сравнить время (дополнительные исследуемые параметры, такие как память и на то, что Вам хватит фантазии - будут плюсом) у реализованной структуры, массива (для Python используйте list, для Cpp - стандартный массив) и односвязного списка (*код реализации массива и односвязного списка загружать не нужно!*).

Чтобы провести исследование необходимо проверить основные операции на маленьком (около 10), среднем (10000) и большом (100000) наборах данных для всех трёх случаев операции (в начало, в середину, в конец). По итогам исследования в отчёте необходимо предоставить таблицу с результатами замеров, а так же их графическое представление (на одном графике необходимо изобразить одну операцию в одном случае для трёх структур, т.е. суммарно должно получиться 9 графиков).

Выполнение работы

Для начала разделим нашу задачу на два файла: в первом файле main.py будет реализован развернутый связный список с методами вставки, удаления, поиска и вывода данного списка. Так же там будет находится функция check(), отвечающая за проверку работы развернутого связного списка.

Во втором файле test.py будут находится все тесты и вычисления для анализа производительности различных структур данных (развернутый связный список, односвязный список и массив).

Начнем с файла main.py – опишу все функции и классы данного файла.

Main.py:

1) Выполним импорт библиотеки math, которая понадобятся в работе программы.

2) Calculate_optimal_node_size: напишем функцию, отвечающую за расчет памяти для реализации узла развернутого связного списка и принимающую одну переменную num_elements - количество элементов, которые необходимо хранить в памяти. Каждый элемент занимает 4 байта, что фиксируется в переменной elem_byte (т. к. используем int).

Сначала функция вычисляет общий объем памяти, необходимый для хранения всех элементов, умножая количество элементов на размер одного элемента. Этот объем хранится в переменной total_memory_capa. Далее, учитывая, что минимальный размер кеш-линии составляет 64 байта, функция вычисляет, сколько кеш-линий потребуется для хранения всего объема памяти. Это делается с помощью округления вверх значения, полученного при делении общего объема памяти на размер кеш-линии. Результат сохраняется в переменной num_of_cache_lines.

После этого функция определяет оптимальный размер узла, добавляя единицу к количеству кеш-линий. Это добавление связано с необходимостью учета дополнительного узла или буфера для управления данными. Функция возвращает рассчитанное значение оптимального размера узла, что позволяет понять, сколько кеш-линий будет необходимо для эффективного хранения и обработки данных.

3) Класс Node_for_ULL: класс представляет собой узел для развернутого связного списка (ULL - Unrolled Linked List), который используется для хранения значений и управления связями между узлами. В конструкторе класса инициализируется пустой список values для хранения значений, параметр size, который определяет максимальное количество значений, которые узел может содержать, и ссылка next, указывающая на следующий узел в списке. Этот класс необходим для реализации структуры данных, позволяющей эффективно добавлять, удалять и управлять элементами.

4) Класс UnrolledLinkedList: классреализует развёрнутый связанный список, который позволяет эффективно хранить и управлять коллекцией элементов, используя узлы, каждый из которых содержит массив значений и ссылку на следующий узел. В конструкторе класса инициализируется head, указывающий на первый узел списка, и node_size, определяющий максимальное количество значений, которое может хранить каждый узел.

Метод append добавляет новое значение в список: если список пуст, создаётся новый узел, в который добавляется значение; если узел уже существует, происходит поиск последнего узла, и если в нём достаточно места, значение добавляется туда; в противном случае создаётся новый узел, и значение помещается в него.

Метод delete удаляет указанное значение из списка: он проходит по узлам, ищет значение и, если находит, удаляет его из массива значений, а

если узел становится пустым и есть предшествующий узел, обновляет ссылку, чтобы исключить пустой узел из списка.

Метод search проверяет наличие указанного значения в списке, проходя через все узлы и возвращая True, если значение найдено, и False в противном случае.

Метод print_list выводит содержимое списка на экран, перечисляя значения в каждом узле, что позволяет визуализировать структуру данных и её содержимое.

5) Функция check: функция принимает два массива arr_1 и arr_2, а также необязательный параметр n_array, который по умолчанию равен None, и если он не задан, вычисляет оптимальный размер узла с помощью функции calculate_optimal_node_size, основываясь на длине arr_1. Затем создаётся экземпляр класса UnrolledLinkedList с размером узла n_array, который будет заполняться данными из arr_1. В цикле для каждого элемента в arr_1 вызывается метод append, добавляющий элемент в список, после чего вызывается метод print_list, чтобы вывести текущее состояние списка. Далее, во втором цикле для каждого элемента в arr_2 выполняется поиск этого элемента в списке с помощью метода search; если элемент найден, он удаляется из списка с помощью метода delete, и снова вызывается print_list, чтобы показать обновлённое состояние списка после удаления.

В файле test.py находятся дополнительные функции, помогающие построить графики для анализа производительности различных структур (массив, односвязный список и развернутый связный список). Реализовано 9 функций, которые помогают рассчитать, сколько времени затрачивает определенная структура на добавление в начало/в середину/в конец, на поиск в начале/в середине/в конце и на удаление в начале/в середине/в конце.

Код программ находится в Приложении А.

Тестирование

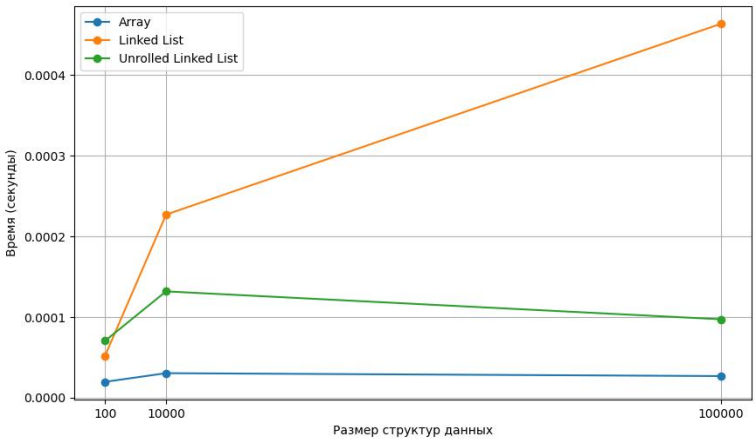
Таблица 1 — Результаты тестирования основного кода программы:

№п/п	Входные данные	Выходные данные	Комментарии
1	1 2 3 4 5 6 7 8 9 10	Node 0: 1 2 Node 1: 3 4 Node 2: 5 6 Node 3: 7 8 Node 4: 9 10	OK
2	14 435 234 62 562 456 24 54 624264	Node 0: 14 435 Node 1: 234 62 Node 2: 562 456 Node 3: 24 54 Node 4: 624264	OK
3	2348 234 32 32 43 234 65756 7567 35673 5675 6735 234 1324 234 34 2342 2 22 2 2 2 24 4 5 5 34 5 34	Node 0: 2348 234 32 Node 1: 32 43 234 Node 2: 65756 7567 35673 Node 3: 5675 6735 234 Node 4: 1324 234 34 Node 5: 2342 2 22 Node 6: 2 2 2 Node 7: 24 4 5	OK

		Node 8: 5 34 5 Node 9: 34	

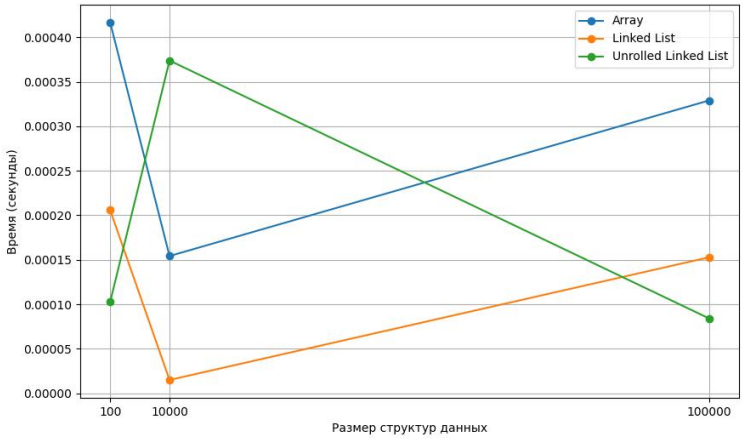
Анализ полученных данных

Вставка элементов в начало различных структур данных:



Проанализировав график, сделаем вывод: массивы работают быстрее для вставки в начало на небольших и средних размерах структуры. Связные списки менее эффективны при больших размерах, когда развёрнутые списки обеспечивают более стабильное время вставки по сравнению с обычными связными списками, особенно на больших данных.

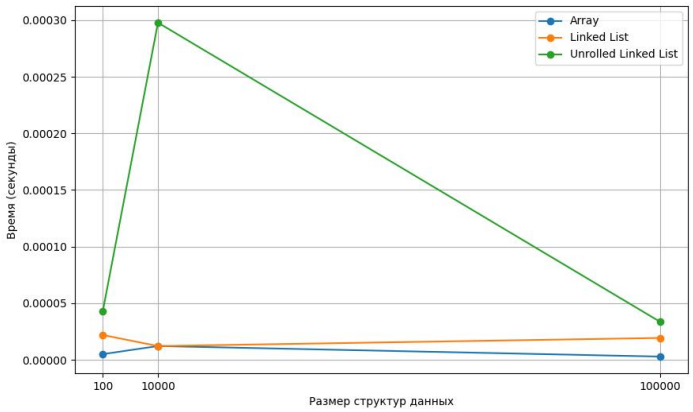
Вставка элементов в середину различных структур данных:



Проанализировав график, сделаем вывод: вставка элемента в середину массива имеет линейное увеличение времени, что соответствует теоретическим ожиданиям (время для массивов увеличивается с ростом размера структуры). Для связного списка время также должно увеличиваться с размером структур, однако график показывает неожиданный спад, что

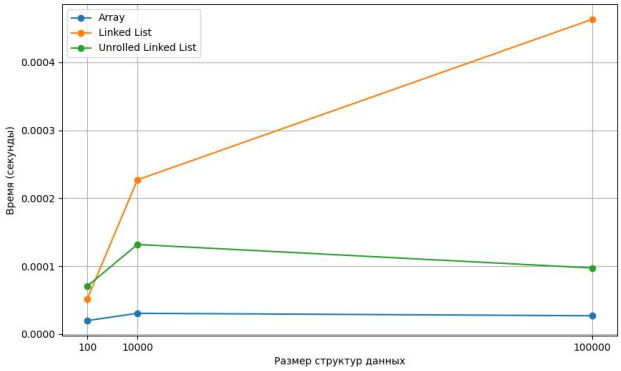
может говорить об особенностях тестирования, но они работают лучше на больших объёмах памяти. Развёрнутый связный список более оптимизирован для работы с большим объёмом данных, поэтому время на больших данных более стабильно.

Вставка элементов в конец различных структур данных:



Проанализировав график, сделаем вывод: вставка в конец малых и больших объёмах данных выполняется за оптимальное время. Односвязный список лучше работает с большими данными. Развёрнутый связный список более оптимизирован для работы с большим объёмом данных, поэтому время на больших данных более стабильно, в отличие от средних данных, что может говорить об особенностях тестирования.

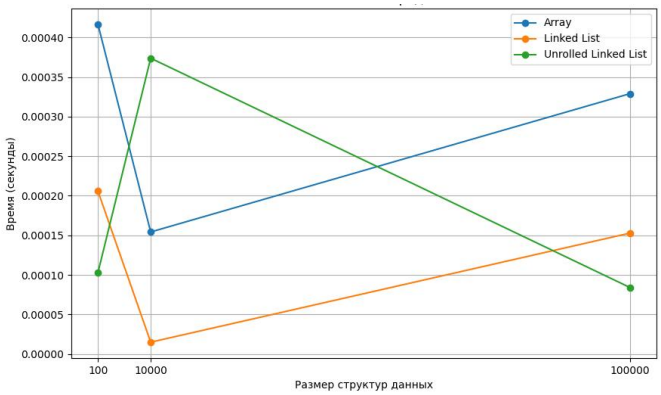
Поиск в начале различных структур данных:



Проанализировав график, сделаем вывод: поиск в начале массива показывает наиболее стабильные и низкие результаты по времени, что связано с тем, что доступ к элементам массива происходит мгновенно

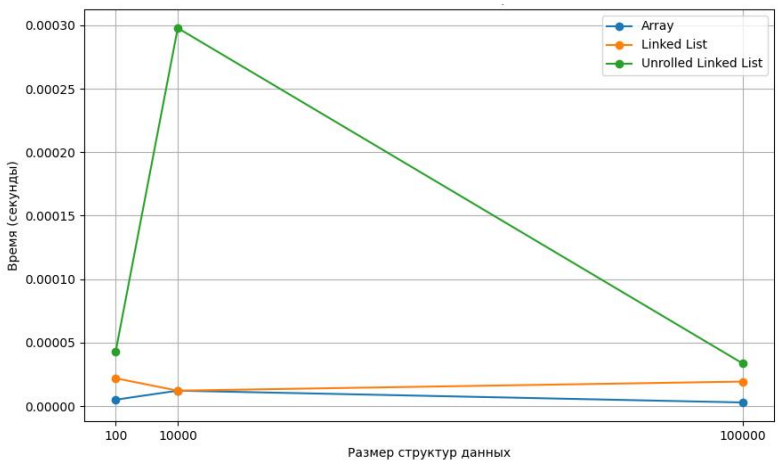
благодаря индексированию. Односвязный список показывает ухудшение производительности при увеличении размера данных. Развёрнутый список наоборот работает лучше на больших объёмах данных.

Поиск в середине различных структур данных:



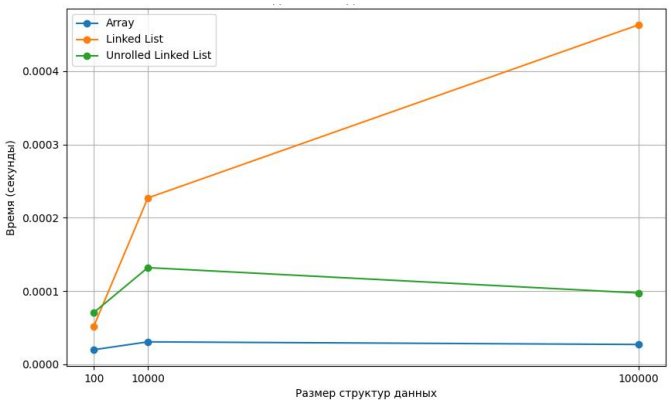
Проанализировав график, сделаем вывод: время поиска в середине массива растёт с увеличением размера структуры данных, т. к. требуется обращение по индексу, но все-равно имеет скачки из-за особенности тестирования. Связный список показывает более быстрые результаты по сравнению со всеми рассматриваемыми структурами при средних размерах данных. Развёрнутый связный список при имеет лучшую эффективность при больших и малых данных.

Поиск в конце различных структур данных:



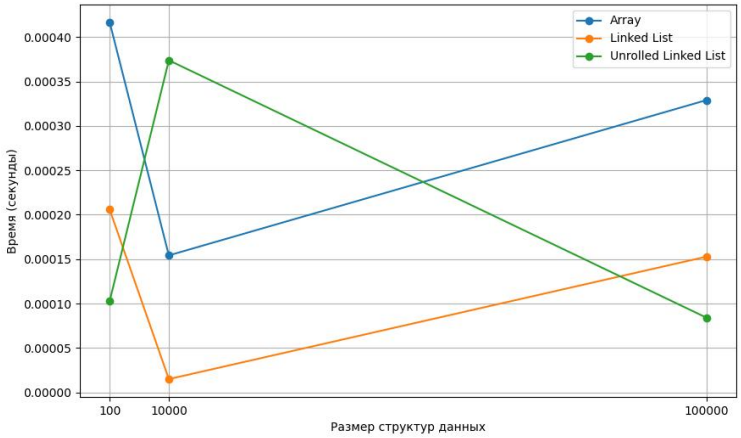
Проанализировав график, сделаем вывод: массив показывает минимальное время для поиска элемента в конце. Связный список показывает средние результаты по сравнению с другими структурами. Развёрнутый список лучше работает с большим количеством данных, чем с меньшим, но в целом уступает массиву из-за особенностей тестирования.

Удаление в начале различных структур данных:



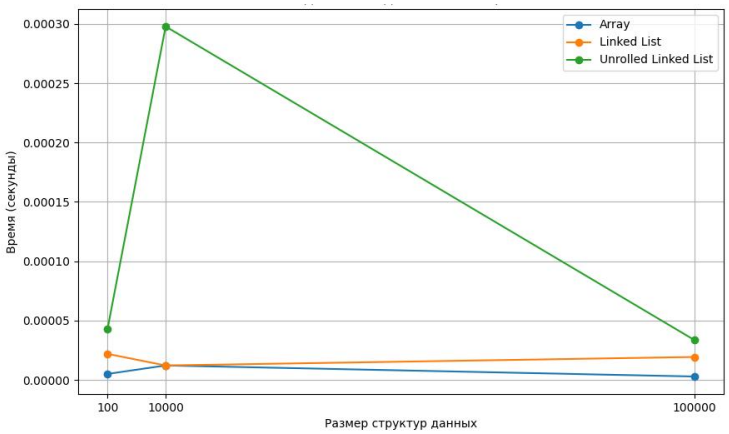
Проанализировав график, сделаем вывод: массив показывает оптимальное время для удаления элемента в начале в зависимости от размера данных. Связный список при меньших данных работает эффективнее развёрнутого списка, но уступает тому при более больших размерах данных. Развёрнутый список, в свою очередь, эффективен при больших данных.

Удаление в середине различных структур данных:



Проанализировав график, сделаем вывод: массив работает оптимально при среднем наборе данных. Односвязный список так же эффективен при среднем объёме данных, но при малых и больших размерах становится не таким эффективным как развёрнутый (оптимальнее справляется при больших размерах данных).

Удаление в конце различных структур данных:



Проанализировав график, сделаем вывод: массив работает оптимальнее остальных структур при различных размерах данных. Односвязный список справляется чуть хуже массива при малых и больших объёмах данных, а развёрнутый список, из-за особенностей тестирования, имеет наихудшую эффективность.

Выводы

В ходе лабораторной работы удалось изучить различные структуры: односвязный список и развёрнутый односвязный список. Так же удалось проанализировать производительность отдельных методов определённых структур. С помощью библиотеки matplotlib удалось построить графики зависимости времени выполнения функции от количества элементов в структурах (100, 10000 и 100000) и проанализировать их. Итог следующий: развёрнутый список эффективнее работает при больших размерах данных, минимизировав время выполнения работы программы, но, к сожалению, он все ещё имеет свои недостатки.

В результате лабораторной работы удалось создать код с реализацией развёрнутого односвязного списка и методов добавления, удаления, поиска и вывода на языке программирования Python.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл main.py:

```
"""
Модуль реализации развернутого связанного списка (Unrolled Linked List).
"""

import math

def calculate_optimal_node_size(num_elements: int) -> int:
    """
    Рассчитывает оптимальный размер узла на основе количества элементов.
    """
    elem_byte = 4
    total_memory_capa = elem_byte * num_elements
    min_cache_line_size = 64
    num_of_cache_lines = math.ceil(total_memory_capa / min_cache_line_size)
    optimal_node_size = num_of_cache_lines + 1
    return optimal_node_size

class NodeForULL:
    """
    Класс, представляющий узел для развернутого связанного списка.
    """

    def __init__(self, size: int):
        """
        Инициализирует узел с указанным размером.
        """
        self.values: list[int] = []
        self.size: int = size
        self.next: 'NodeForULL | None' = None

    def __len__(self) -> int:
        """
        Возвращает количество элементов в узле.
        """
        return len(self.values)

    def __str__(self) -> str:
        """
        Возвращает строковое представление узла.
        """
        return f"Node(size={self.size}, values={self.values})"

class UnrolledLinkedList:
    """
    Класс, представляющий развернутый связанный список.
    """

    def __init__(self, node_capacity: int):
        """
        Инициализирует развернутый связанный список с заданной емкостью узла.
        """
        self.head: 'NodeForULL | None' = None
        self.node_capacity: int = node_capacity

    def prepend(self, new_value: int) -> None:
        """
```

```
        Добавляет элемент в начало списка.
        """
        new_node = NodeForULL(self.node_capacity)
        new_node.values.append(new_value)

        if not self.head:
            self.head = new_node
            return

        current = self.head
        if len(current.values) < self.node_capacity:
            new_node.values.extend(current.values)
            new_node.next = current.next
            self.head = new_node
        else:
            new_node.next = current
            self.head = new_node

    def insert_middle(self, new_value: int) -> None:
        """
        Вставляет элемент в середину списка.
        """
        if self.head is None:
            self.head = NodeForULL(self.node_capacity)
            self.head.values.append(new_value)
            return

        slow = self.head
        fast = self.head

        while fast.next and fast.next.next:
            slow = slow.next
            fast = fast.next.next

        if len(slow.values) < self.node_capacity:
            slow.values.append(new_value)
        else:
            new_node = NodeForULL(self.node_capacity)
            mid_index = len(slow.values) // 2
            new_node.values = slow.values[mid_index:]
            slow.values = slow.values[:mid_index]
            new_node.next = slow.next
            slow.next = new_node
            slow.values.append(new_value)

    def append(self, new_value: int) -> None:
        """
        Добавляет элемент в конец списка.
        """
        if self.head is None:
            self.head = NodeForULL(self.node_capacity)
            self.head.values.append(new_value)
        else:
            current = self.head
            while current.next:
                current = current.next
            if len(current.values) < self.node_capacity:
                current.values.append(new_value)
            else:
                new_node = NodeForULL(self.node_capacity)
                new_node.values.append(new_value)
```

```
                current.next = new_node

    def delete(self, delete_value: int) -> None:
        """
        Удаляет указанный элемент из списка.
        """
        current = self.head
        prev = None

        while current:
            if delete_value in current.values:
                current.values.remove(delete_value)
                if not current.values and prev:
                    prev.next = current.next
                return
            prev = current
            current = current.next

    def search(self, search_value: int) -> bool:
        """
        Выполняет поиск элемента в списке.
        """
        current = self.head
        while current:
            if search_value in current.values:
                return True
            current = current.next
        return False

    def clear(self) -> None:
        """
        Очищает весь список.
        """
        self.head = None

    def print_list(self) -> None:
        """
        Выводит содержимое списка.
        """
        current = self.head
        node_index = 0
        while current:
            print(f"Node {node_index}: {' '.join(map(str, current.values))}")
            current = current.next
            node_index += 1

def check(arr_1: list[int], arr_2: list[int], n_array: int | None = None) ->
None:
    """
    Проверяет работу развернутого связанного списка:
    добавляет элементы из arr_1 и удаляет те, что есть в arr_2.
    """
    if n_array is None:
        n_array = calculate_optimal_node_size(len(arr_1))

    final_list = UnrolledLinkedList(n_array)

    for elem in arr_1:
        final_list.append(elem)
        final_list.print_list()
```

```
    for elem in arr_2:
        if final_list.search(elem):
            final_list.delete(elem)
            final_list.print_list()

if __name__ == "__main__":
    values = list(map(int, input("Введите числа через пробел: ").split()))
    calculated_node_size = calculate_optimal_node_size(len(values))
    unrolled_linked_list = UnrolledLinkedList(calculated_node_size)

    for val in values:
        unrolled_linked_list.append(val)

    unrolled_linked_list.print_list()
```

Файл test.py:

```
import time
import random
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
from main import UnrolledLinkedList, calculate_optimal_node_size
class Node:
    def __init__(self, value):
        self.value = value
        self.next = None


# Односвязный список
class LinkedList:
    def __init__(self):
        self.head = None
    # Поиск
    def search(self, search_value):
        current = self.head

        while current:
            if current.value == search_value:
                return True
            current = current.next
        return False


    # Удаление
    def delete(self, delete_value):
        current = self.head

        if current and current.value == delete_value:
            self.head = current.next
            return


        prev = None
        while current and current.value != delete_value:
            prev = current
            current = current.next


        if current is None:
            return


        prev.next = current.next
```

```
    # Добавление элемента в начало
    def prepend(self, value):
        new_value = Node(value)
        new_value.next = self.head
        self.head = new_value


    # Вставка в середину
    def insert_middle(self, value):
        new_value = Node(value)

        if not self.head:  # Если список пуст, добавляем элемент в начало
            self.head = new_value
            return

        slow = self.head
        fast = self.head
        # Находим средний узел
        while fast and fast.next:
            slow = slow.next
            fast = fast.next.next

        # Вставляем новый узел после среднего узла
        new_value.next = slow.next
        slow.next = new_value
    # Вставка в конец
    def append(self, value):
        new_value = Node(value)
        if self.head is None:
            self.head = new_value
        else:
            current = self.head
            while current.next:
                current = current.next
            current.next = new_value

    def clear(self):
        self.head = None

    # Вывод списка
    def print_list(self):
        current = self.head
        result = []
        while current:
            result.append(current.value)
            current = current.next
        print(" ".join(map(str, result)))

# Функция проверки затраченного времени у массива на вставку/поиск/удаление в
начало
def array_elapsed_time_ins_ser_del_begin():
    result_time_array_ins_ser_del_begin = [[], [], []]
    #100 элементов
    array_1 = np.random.randint(1, 100, 100)
    working_array = []
    time_start_array_1 = time.time()
    #вставка в начало массива
    for i in array_1:
```

```
        working_array.insert(0,int(i))
    time_end_array_1 = time.time()
    result_time_array_ins_ser_del_begin[0].append(time_end_array_1 -
time_start_array_1)
    #поиск в начале массива
    time_start_array_1 = time.time()
    for i in array_1[::-1]:
        for j in working_array:
            if j == i:
                continue
    time_end_array_1 = time.time()
    result_time_array_ins_ser_del_begin[1].append(time_end_array_1 -
time_start_array_1)
    #удаление в начале массива
    time_start_array_1 = time.time()
    while working_array:
        working_array.pop(0)
    time_end_array_1 = time.time()
    result_time_array_ins_ser_del_begin[2].append(time_end_array_1 -
time_start_array_1)


    #10к элементов
    array_2 = np.random.randint(1, 1000, 1000)
    time_start_array_2 = time.time()
    #вставка в начало массива
    for i in array_2:
        working_array.insert(0, int(i))
    time_end_array_2 = time.time()
    result_time_array_ins_ser_del_begin[0].append(time_end_array_2 -
time_start_array_2)
    #поиск в начале массива
    time_start_array_2 = time.time()
    for i in array_2[::-1]:
        for j in working_array:
            if j == i:
                continue
    time_end_array_2 = time.time()
    result_time_array_ins_ser_del_begin[1].append(time_end_array_2 -
time_start_array_2)
    #удаление в начале массива
    time_start_array_2 = time.time()
    while working_array:
        working_array.pop(0)
    time_end_array_2 = time.time()
    result_time_array_ins_ser_del_begin[2].append(time_end_array_2 -
time_start_array_2)



    #100к элементов
    array_3 = np.random.randint(1, 10000, 10000)
    time_start_array_3 = time.time()
    #вставка в начало массива
    for i in array_3:
        working_array.insert(0, int(i))
    time_end_array_3 = time.time()
    result_time_array_ins_ser_del_begin[0].append(time_end_array_3 -
time_start_array_3)
    #поиск в начале массива
    time_start_array_3 = time.time() #тоже ну ооооочень долго :(
    for i in array_3[::-1]:
```

```
        for j in working_array:
            if j == i:
                continue
    time_end_array_3 = time.time()
    result_time_array_ins_ser_del_begin[1].append(time_end_array_3 -
time_start_array_3)
    #удаление в начале массива
    time_start_array_3 = time.time()
    while working_array:
        working_array.pop(0)
    time_end_array_3 = time.time()
    result_time_array_ins_ser_del_begin[2].append(time_end_array_3 -
time_start_array_3)

    return result_time_array_ins_ser_del_begin

# Функция проверки затраченного времени у массива на вставку/поиск/удаление в
середину
def array_elapsed_time_ins_ser_del_middle():
    result_time_array_ins_ser_del_middle = [[], [], []]
    #100 элементов
    array_1 = np.random.randint(1, 100, 100)
    working_array = []
    time_start_array_1 = time.time()
    #вставка в середину массива
    for i in array_1:
        working_array.insert(int(len(working_array)/2),int(i))
    time_end_array_1 = time.time()
    result_time_array_ins_ser_del_middle[0].append(time_end_array_1 -
time_start_array_1)
    #поиск в середине массива
    time_start_array_1 = time.time()
    for i in range(int(len(array_1)/2), len(array_1) + len(array_1)%2):
        for j in range(0, int(len(working_array)/2 + len(working_array)%2)):
            if j == i:
                continue
    for i in range(0, int(len(array_1)+len(array_1)%2)):
        for j in range(int(len(working_array)/2),
len(working_array)+len(working_array)%2):
            if j == i:
                continue
    time_end_array_1 = time.time()
    result_time_array_ins_ser_del_middle[1].append(time_end_array_1 -
time_start_array_1)

    #удаление в середине массива
    time_start_array_1 = time.time()
    while working_array:
        working_array.pop(int(len(working_array)/2))
    time_end_array_1 = time.time()
    result_time_array_ins_ser_del_middle[2].append(time_end_array_1 -
time_start_array_1)



    #10к элементов
    array_2 = np.random.randint(1, 1000, 1000)
    time_start_array_2 = time.time()
    #вставка в середину массива
    for i in array_2:
```

```
        working_array.insert(int(len(working_array)/2), int(i))
    time_end_array_2 = time.time()
    result_time_array_ins_ser_del_middle[0].append(time_end_array_2 -
time_start_array_2)

    #поиск в середине массива
    time_start_array_2 = time.time()
    for i in range(int(len(array_2)/2), len(array_2) + len(array_2)%2):
        for j in range(0, int(len(working_array)/2 + len(working_array)%2)):
            if j == i:
                continue
    for i in range(0, int(len(array_2)+len(array_2)%2)):
        for j in range(int(len(working_array)/2),
len(working_array)+len(working_array)%2):
            if j == i:
                continue
    time_end_array_2 = time.time()
    result_time_array_ins_ser_del_middle[1].append(time_end_array_2 -
time_start_array_2)

    #удаление в середине массива
    time_start_array_2 = time.time()
    while working_array:
        working_array.pop(int(len(working_array)/2))
    time_end_array_2 = time.time()
    result_time_array_ins_ser_del_middle[2].append(time_end_array_2 -
time_start_array_2)

    #100к элементов
    array_3 = np.random.randint(1, 10000, 10000)
    time_start_array_3 = time.time()
    #вставка в середину массива
    for i in array_3:
        working_array.insert(int(len(working_array)/2), int(i))
    time_end_array_3 = time.time()
    result_time_array_ins_ser_del_middle[0].append(time_end_array_3 -
time_start_array_3)
    #поиск в середине массива
    time_start_array_3 = time.time()
    for i in range(int(len(array_3)/2), len(array_3) + len(array_3)%2):
        for j in range(0, int(len(working_array)/2 + len(working_array)%2)):
            if j == i:
                continue
    for i in range(0, int(len(array_3)+len(array_3)%2)):
        for j in range(int(len(working_array)/2),
len(working_array)+len(working_array)%2):
            if j == i:
                continue
    time_end_array_3 = time.time()
    result_time_array_ins_ser_del_middle[1].append(time_end_array_3 -
time_start_array_3)

    #удаление в середине массива
    time_start_array_3 = time.time()
    while working_array:
        working_array.pop(int(len(working_array)/2))
    time_end_array_3 = time.time()
    result_time_array_ins_ser_del_middle[2].append(time_end_array_3 -
time_start_array_3)
```

```
    return result_time_array_ins_ser_del_middle

# Функция проверки затраченного времени у массива на вставку/поиск/удаление в
конец
def array_elapsed_time_ins_ser_del_end():
    result_time_array_ins_ser_del_end = [[], [], []]
    #100 элементов
    array_1 = np.random.randint(1, 100, 100)
    working_array = []
    time_start_array_1 = time.time()
    #вставка в конец массива
    for i in array_1:
        working_array.append(int(i))
    time_end_array_1 = time.time()
    result_time_array_ins_ser_del_end[0].append(time_end_array_1 -
time_start_array_1)
    #поиск в конце массива
    time_start_array_1 = time.time()
    for i in array_1:
        for j in working_array:
            if j == i:
                continue
    time_end_array_1 = time.time()
    result_time_array_ins_ser_del_end[1].append(time_end_array_1 -
time_start_array_1)
    #удаление в конце массива
    time_start_array_1 = time.time()
    while working_array:
        working_array.pop()
    time_end_array_1 = time.time()
    result_time_array_ins_ser_del_end[2].append(time_end_array_1 -
time_start_array_1)


    #10к элементов
    array_2 = np.random.randint(1, 1000, 1000)
    time_start_array_2 = time.time()
    #вставка в конец массива
    for i in array_2:
        working_array.append(int(i))
    time_end_array_2 = time.time()
    result_time_array_ins_ser_del_end[0].append(time_end_array_2 -
time_start_array_2)
     #поиск в конце массива
    time_start_array_2 = time.time()
    for i in array_2:
        for j in working_array:
            if j == i:
                continue
    time_end_array_2 = time.time()
    result_time_array_ins_ser_del_end[1].append(time_end_array_2 -
time_start_array_2)
    #удаление в конце массива
    time_start_array_2 = time.time()
    while working_array:
        working_array.pop()
    time_end_array_2 = time.time()
    result_time_array_ins_ser_del_end[2].append(time_end_array_2 -
time_start_array_2)
```

```
    #100к элементов
    array_3 = np.random.randint(1, 10000, 10000)
    time_start_array_3 = time.time()
    #вставка в конец массива
    for i in array_3:
        working_array.append(int(i))
    time_end_array_3 = time.time()
    result_time_array_ins_ser_del_end[0].append(time_end_array_3 -
time_start_array_3)
     #поиск в конце массива
    time_start_array_3 = time.time()
    for i in array_3:
        for j in working_array:
            if j == i:
                continue
    time_end_array_3 = time.time()
    result_time_array_ins_ser_del_end[1].append(time_end_array_3 -
time_start_array_3)
    #удаление в конце массива
    time_start_array_3 = time.time()
    while working_array:
        working_array.pop()
    time_end_array_3 = time.time()
    result_time_array_ins_ser_del_end[2].append(time_end_array_3 -
time_start_array_3)

    return result_time_array_ins_ser_del_end

# Функция проверки затраченного времени у односвязанного списка на
вставку/поиск/удаление в начало
def linkedlist_elapsed_time_ins_ser_del_begin():
    result_time_LL_ins_ser_del_begin = [[], [], []]
    worked_LinkedList = LinkedList()
    #100 элементов
    array_1 = np.random.randint(1, 100, 100)
    time_start_LL_1 = time.time()
    #вставка в начало односвязного списка
    for i in array_1:
        worked_LinkedList.prepend(i)
    time_end_LL_1 = time.time()
    result_time_LL_ins_ser_del_begin[0].append(time_end_LL_1 -
time_start_LL_1)
   #поиск в начале односвязного списка
    time_start_array_1 = time.time()
    for i in array_1[::-1]:
        worked_LinkedList.search(i)
    time_end_array_1 = time.time()
    result_time_LL_ins_ser_del_begin[1].append(time_end_array_1 -
time_start_array_1)
    #удаление в начале односвязного списка
    time_start_array_1 = time.time()
    for i in array_1[::-1]:
        worked_LinkedList.delete(i)
    time_end_array_1 = time.time()
    result_time_LL_ins_ser_del_begin[2].append(time_end_array_1 -
time_start_array_1)

   #10к элементов
    array_2 = np.random.randint(1, 1000, 1000)
    time_start_LL_2 = time.time()
```

```
    #вставка в начало односвязного списка
    for i in array_2:
        worked_LinkedList.prepend(i)
    time_end_LL_2 = time.time()
    worked_LinkedList.clear()
    result_time_LL_ins_ser_del_begin[0].append(time_end_LL_2 -
time_start_LL_2)
    #поиск в начале односвязного списка
    time_start_array_2 = time.time()
    for i in array_2[::-1]:
        worked_LinkedList.search(i)
    time_end_array_2 = time.time()
    result_time_LL_ins_ser_del_begin[1].append(time_end_array_2 -
time_start_array_2)
    #поиск в начале односвязного списка
    time_start_array_2 = time.time()
    for i in array_2[::-1]:
        worked_LinkedList.delete(i)
    time_end_array_2 = time.time()
    result_time_LL_ins_ser_del_begin[2].append(time_end_array_2 -
time_start_array_2)


    #100к элементов
    array_3 = np.random.randint(1, 10000, 10000)
    time_start_LL_3 = time.time()
    #вставка в начало односвязного списка
    for i in array_3:
        worked_LinkedList.prepend(i)
    time_end_LL_3 = time.time()
    worked_LinkedList.clear()
    result_time_LL_ins_ser_del_begin[0].append(time_end_LL_3 -
time_start_LL_3)
    #поиск в начале односвязного списка
    time_start_array_3 = time.time()
    for i in array_3[::-1]:
        worked_LinkedList.search(i)
    time_end_array_3 = time.time()
    result_time_LL_ins_ser_del_begin[1].append(time_end_array_3 -
time_start_array_3)
    #поиск в начале односвязного списка
    time_start_array_3 = time.time()
    for i in array_3[::-1]:
        worked_LinkedList.delete(i)
    time_end_array_3 = time.time()
    result_time_LL_ins_ser_del_begin[2].append(time_end_array_3 -
time_start_array_3)


    return result_time_LL_ins_ser_del_begin


# Функция проверки затраченного времени у односвязанного списка на
вставку/поиск/удаление в середину
def linkedlist_elapsed_time_ins_ser_del_middle():
    result_time_LL_ins_ser_del_middle = [[], [], []]
    worked_LinkedList = LinkedList()
    #100 элементов
    array_1 = np.random.randint(1, 100, 100)
    time_start_LL_1 = time.time()
    #вставка в середину односвязного списка
    for i in array_1:
        worked_LinkedList.insert_middle(i)
```

```
    time_end_LL_1 = time.time()
    worked_LinkedList.clear()
    result_time_LL_ins_ser_del_middle[0].append(time_end_LL_1 -
time_start_LL_1)
    #поиск в середине односвязного списка
    time_start_array_1 = time.time()
    for i in range(0, int(len(array_1)+len(array_1)%2)):
        worked_LinkedList.search(i)
    for i in range(int(len(array_1)/2), len(array_1) + len(array_1)%2):
        worked_LinkedList.search(i)
    time_end_array_1 = time.time()
    result_time_LL_ins_ser_del_middle[1].append(time_end_array_1 -
time_start_array_1)
    #удаление в середине односвязного списка
    time_start_array_1 = time.time()
    for i in range(0, int(len(array_1)+len(array_1)%2)):
        worked_LinkedList.delete(i)
    for i in range(int(len(array_1)/2), len(array_1) + len(array_1)%2):
        worked_LinkedList.delete(i)
    time_end_array_1 = time.time()
    result_time_LL_ins_ser_del_middle[2].append(time_end_array_1 -
time_start_array_1)


   #10к элементов
    array_2 = np.random.randint(1, 1000, 1000)
    time_start_LL_2 = time.time()
    #вставка в середину односвязного списка
    for i in array_2:
         worked_LinkedList.insert_middle(i)
    time_end_LL_2 = time.time()
    result_time_LL_ins_ser_del_middle[0].append(time_end_LL_2 -
time_start_LL_2)
    #поиск в середине односвязного списка
    time_start_array_2 = time.time()
    for i in range(0, int(len(array_2)+len(array_2)%2)):
        worked_LinkedList.search(i)
    for i in range(int(len(array_2)/2), len(array_2) + len(array_2)%2):
        worked_LinkedList.search(i)
    time_end_array_2 = time.time()
    result_time_LL_ins_ser_del_middle[1].append(time_end_array_2 -
time_start_array_2)


    #удаление в середине односвязного списка
    time_start_array_2 = time.time()
    for i in range(0, int(len(array_2)+len(array_2)%2)):
        worked_LinkedList.delete(i)
    for i in range(int(len(array_2)/2), len(array_2) + len(array_2)%2):
        worked_LinkedList.delete(i)
    time_end_array_2 = time.time()
    result_time_LL_ins_ser_del_middle[2].append(time_end_array_2 -
time_start_array_2)


    #100к элементов
    array_3 = np.random.randint(1, 10000, 10000)#оооочень долго :(
    time_start_LL_3 = time.time()
    #вставка в середину односвязного списка
    for i in array_3:
        worked_LinkedList.insert_middle(i)
    time_end_LL_3 = time.time()
```

```
    result_time_LL_ins_ser_del_middle[0].append(time_end_LL_3 -
time_start_LL_3)
     #поиск в середине односвязного списка
    time_start_array_3 = time.time()
    for i in range(0, int(len(array_3)+len(array_3)%2)):
        worked_LinkedList.search(i)
    for i in range(int(len(array_3)/2), len(array_3) + len(array_3)%2):
        worked_LinkedList.search(i)
    time_end_array_3 = time.time()
    result_time_LL_ins_ser_del_middle[1].append(time_end_array_3 -
time_start_array_3)
    #удаление в середине односвязного списка
    time_start_array_3 = time.time()
    for i in range(0, int(len(array_3)+len(array_3)%2)):
        worked_LinkedList.delete(i)
    for i in range(int(len(array_3)/2), len(array_3) + len(array_3)%2):
        worked_LinkedList.delete(i)
    time_end_array_3 = time.time()
    result_time_LL_ins_ser_del_middle[2].append(time_end_array_3 -
time_start_array_3)

    return result_time_LL_ins_ser_del_middle


# Функция проверки затраченного времени у односвязанного списка на
вставку/поиск/удаление в конец
def linkedlist_elapsed_time_ins_ser_del_end():
    result_time_LL_ins_ser_del_end = [[], [], []]
    worked_LinkedList = LinkedList()
    #100 элементов
    array_1 = np.random.randint(1, 100, 100)
    time_start_LL_1 = time.time()
    #вставка в конец односвязного списка
    for i in array_1:
        worked_LinkedList.append(i)
    time_end_LL_1 = time.time()
    result_time_LL_ins_ser_del_end[0].append(time_end_LL_1 - time_start_LL_1)
    #поиск в конце односвязного списка
    time_start_array_1 = time.time()
    for i in array_1:
        worked_LinkedList.search(i)
    time_end_array_1 = time.time()
    result_time_LL_ins_ser_del_end[1].append(time_end_array_1 -
time_start_array_1)
    #удаление в конце односвязного списка
    time_start_array_1 = time.time()
    for i in array_1:
        worked_LinkedList.delete(i)
    time_end_array_1 = time.time()
    result_time_LL_ins_ser_del_end[2].append(time_end_array_1 -
time_start_array_1)


   #10к элементов
    array_2 = np.random.randint(1, 1000, 1000)
    time_start_LL_2 = time.time()
    #вставка в конец односвязного списка
    for i in array_2:
         worked_LinkedList.append(i)
    time_end_LL_2 = time.time()
    result_time_LL_ins_ser_del_end[0].append(time_end_LL_2 - time_start_LL_2)
    #поиск в конце односвязного списка
```

```
    time_start_array_2 = time.time()
    for i in array_2:
        worked_LinkedList.search(i)
    time_end_array_2 = time.time()
    result_time_LL_ins_ser_del_end[1].append(time_end_array_2 -
time_start_array_2)
    #удаление в конце односвязного списка
    time_start_array_2 = time.time()
    for i in array_2:
        worked_LinkedList.delete(i)
    time_end_array_2 = time.time()
    result_time_LL_ins_ser_del_end[2].append(time_end_array_2 -
time_start_array_2)
    #100к элементов
    array_3 = np.random.randint(1, 10000, 10000)
    time_start_LL_3 = time.time()
    #вставка в конец односвязного списка
    for i in array_3:
        worked_LinkedList.append(i)
    time_end_LL_3 = time.time()
    result_time_LL_ins_ser_del_end[0].append(time_end_LL_3 - time_start_LL_3)
    #поиск в конце односвязного списка
    time_start_array_3 = time.time()
    for i in array_3:
        worked_LinkedList.search(i)
    time_end_array_3 = time.time()
    result_time_LL_ins_ser_del_end[1].append(time_end_array_3 -
time_start_array_3)
    #удаление в конце односвязного списка
    time_start_array_3 = time.time()
    for i in array_3:
        worked_LinkedList.delete(i)
    time_end_array_3 = time.time()
    result_time_LL_ins_ser_del_end[2].append(time_end_array_3 -
time_start_array_3)

    return result_time_LL_ins_ser_del_end


# Функция проверки затраченного времени у развернутого списка на
вставку/поиск/удаление в начало
def ULL_elapsed_time_ins_ser_del_begin():
    result_time_ULL_ins_ser_del_begin = [[],[],[]]
    worked_ULL = UnrolledLinkedList(calculate_optimal_node_size(100))
    #100 элементов
    array_1 = np.random.randint(1, 100, 100)
    time_start_ULL_1 = time.time()
    #вставка в начало развернутого списка
    for i in array_1:
        worked_ULL.prepend(i)
    time_end_ULL_1 = time.time()
    result_time_ULL_ins_ser_del_begin[0].append(time_end_ULL_1 -
time_start_ULL_1)
    #поиск в начале развернутого списка
    time_start_array_1 = time.time()
    for i in array_1[::-1]:
        worked_ULL.search(i)
    time_end_array_1 = time.time()
    result_time_ULL_ins_ser_del_begin[1].append(time_end_array_1 -
time_start_array_1)
    #удаление в начале развернутого списка
```

```
    time_start_array_1 = time.time()
    for i in array_1[::-1]:
        worked_ULL.delete(i)
    time_end_array_1 = time.time()
    result_time_ULL_ins_ser_del_begin[2].append(time_end_array_1 -
time_start_array_1)

    worked_ULL = UnrolledLinkedList(calculate_optimal_node_size(10000))
    #10к элементов
    array_2 = np.random.randint(1, 1000, 1000)
    time_start_ULL_2 = time.time()
    #вставка в начало развернутого списка
    for i in array_2:
        worked_ULL.prepend(i)
    time_end_ULL_2 = time.time()
    result_time_ULL_ins_ser_del_begin[0].append(time_end_ULL_2 -
time_start_ULL_2)
    #поиск в начале развернутого  списка
    time_start_array_2 = time.time()
    for i in array_2[::-1]:
        worked_ULL.search(i)
    time_end_array_2 = time.time()
    result_time_ULL_ins_ser_del_begin[1].append(time_end_array_2 -
time_start_array_2)
    #удаление в начале развернутого  списка
    time_start_array_2 = time.time()
    for i in array_2[::-1]:
        worked_ULL.delete(i)
    time_end_array_2 = time.time()
    result_time_ULL_ins_ser_del_begin[2].append(time_end_array_2 -
time_start_array_2)

    worked_ULL = UnrolledLinkedList(calculate_optimal_node_size(100000))
    #100к элементов
    array_3 = np.random.randint(1, 10000, 10000)
    time_start_ULL_3 = time.time()
    #вставка в начало развернутого  списка
    for i in array_3:
        worked_ULL.prepend(i)
    time_end_ULL_3 = time.time()
    result_time_ULL_ins_ser_del_begin[0].append(time_end_ULL_3 -
time_start_ULL_3)
    #поиск в начале развернутого  списка
    time_start_array_3 = time.time()
    for i in array_3[::-1]:
        worked_ULL.search(i)
    time_end_array_3 = time.time()
    result_time_ULL_ins_ser_del_begin[1].append(time_end_array_3 -
time_start_array_3)
    #удаление в начале односвязного списка
    time_start_array_3 = time.time()
    for i in array_3[::-1]:
        worked_ULL.delete(i)
    time_end_array_3 = time.time()
    result_time_ULL_ins_ser_del_begin[2].append(time_end_array_3 -
time_start_array_3)

    return result_time_ULL_ins_ser_del_begin
```

```
# Функция проверки затраченного времени у развернутого списка на
вставку/поиск/удаление в середину
def ULL_elapsed_time_ins_ser_del_middle():
    result_time_ULL_ins_ser_del_middle = [[], [], []]
    worked_ULL = UnrolledLinkedList(calculate_optimal_node_size(100))
    #100 элементов
    array_1 = np.random.randint(1, 100, 100)
    time_start_ULL_1 = time.time()
    #вставка в середину развернутого списка
    for i in array_1:
        worked_ULL.insert_middle(i)
    time_end_ULL_1 = time.time()
    result_time_ULL_ins_ser_del_middle[0].append(time_end_ULL_1 -
time_start_ULL_1)
    #поиск в середине односвязного списка
    time_start_array_1 = time.time()
    for i in range(0, int(len(array_1)+len(array_1)%2)):
        worked_ULL.search(i)
    for i in range(int(len(array_1)/2), len(array_1) + len(array_1)%2):
        worked_ULL.search(i)
    time_end_array_1 = time.time()
    result_time_ULL_ins_ser_del_middle[1].append(time_end_array_1 -
time_start_array_1)
    #удаление в середине односвязного списка
    time_start_array_1 = time.time()
    for i in range(0, int(len(array_1)+len(array_1)%2)):
        worked_ULL.delete(i)
    for i in range(int(len(array_1)/2), len(array_1) + len(array_1)%2):
        worked_ULL.delete(i)
    time_end_array_1 = time.time()
    result_time_ULL_ins_ser_del_middle[2].append(time_end_array_1 -
time_start_array_1)


    worked_ULL = UnrolledLinkedList(calculate_optimal_node_size(10000))
    #10к элементов
    array_2 = np.random.randint(1, 1000, 1000)
    time_start_ULL_2 = time.time()
    #вставка в середину развернутого списка
    for i in array_2:
        worked_ULL.insert_middle(i)
    time_end_ULL_2 = time.time()
    result_time_ULL_ins_ser_del_middle[0].append(time_end_ULL_2 -
time_start_ULL_2)
#поиск в середине односвязного списка
    time_start_array_2 = time.time()
    for i in range(0, int(len(array_2)+len(array_2)%2)):
        worked_ULL.search(i)
    for i in range(int(len(array_2)/2), len(array_2) + len(array_2)%2):
        worked_ULL.search(i)
    time_end_array_2 = time.time()
    result_time_ULL_ins_ser_del_middle[1].append(time_end_array_2 -
time_start_array_2)
    #удаление в середине односвязного списка
    time_start_array_2 = time.time()
    for i in range(0, int(len(array_2)+len(array_2)%2)):
        worked_ULL.delete(i)
    for i in range(int(len(array_2)/2), len(array_2) + len(array_2)%2):
        worked_ULL.delete(i)
    time_end_array_2 = time.time()
    result_time_ULL_ins_ser_del_middle[2].append(time_end_array_2 -
time_start_array_2)
```

```
    worked_ULL = UnrolledLinkedList(calculate_optimal_node_size(100000))
    #100к элементов
    array_3 = np.random.randint(1, 10000, 10000)
    time_start_ULL_3 = time.time()
    #вставка в середину развернутого списка
    for i in array_3:
        worked_ULL.insert_middle(i)
    time_end_ULL_3 = time.time()
    result_time_ULL_ins_ser_del_middle[0].append(time_end_ULL_3 -
time_start_ULL_3)
    #поиск в середине односвязного списка
    time_start_array_3 = time.time()
    for i in range(0, int(len(array_3)+len(array_3)%2)):
        worked_ULL.search(i)
    for i in range(int(len(array_3)/2), len(array_3) + len(array_3)%2):
        worked_ULL.search(i)
    time_end_array_3 = time.time()
    result_time_ULL_ins_ser_del_middle[1].append(time_end_array_3 -
time_start_array_3)
    #удаление в середине односвязного списка
    time_start_array_3 = time.time()
    for i in range(0, int(len(array_3)+len(array_3)%2)):
        worked_ULL.delete(i)
    for i in range(int(len(array_3)/2), len(array_3) + len(array_3)%2):
        worked_ULL.delete(i)
    time_end_array_3 = time.time()
    result_time_ULL_ins_ser_del_middle[2].append(time_end_array_3 -
time_start_array_3)


    return result_time_ULL_ins_ser_del_middle


# Функция проверки затраченного времени у развернутого списка на
вставку/поиск/удаление в конец
def ULL_elapsed_time_ins_ser_del_end():
    result_time_ULL_ins_ser_del_end = [[],[],[]]
    worked_ULL = UnrolledLinkedList(calculate_optimal_node_size(100))
    #100 элементов
    array_1 = np.random.randint(1, 100, 100)
    time_start_ULL_1 = time.time()
    #вставка в конец развернутого списка
    for i in array_1:
        worked_ULL.append(i)
    time_end_ULL_1 = time.time()
    result_time_ULL_ins_ser_del_end[0].append(time_end_ULL_1 -
time_start_ULL_1)
     #поиск в конце односвязного списка
    time_start_array_1 = time.time()
    for i in array_1:
        worked_ULL.search(i)
    time_end_array_1 = time.time()
    result_time_ULL_ins_ser_del_end[1].append(time_end_array_1 -
time_start_array_1)
    #удаление в конце односвязного списка
    time_start_array_1 = time.time()
    for i in array_1:
        worked_ULL.delete(i)
    time_end_array_1 = time.time()
    result_time_ULL_ins_ser_del_end[2].append(time_end_array_1 -
time_start_array_1)
```

```
    worked_ULL = UnrolledLinkedList(calculate_optimal_node_size(10000))
    #10к элементов
    array_2 = np.random.randint(1, 1000, 1000)
    time_start_ULL_2 = time.time()
    #вставка в конец развернутого списка
    for i in array_2:
        worked_ULL.append(i)
    time_end_ULL_2 = time.time()
    result_time_ULL_ins_ser_del_end[0].append(time_end_ULL_2 -
time_start_ULL_2)
     #поиск в конце разв списка
    time_start_array_2 = time.time()
    for i in array_2:
        worked_ULL.search(i)
    time_end_array_2 = time.time()
    result_time_ULL_ins_ser_del_end[1].append(time_end_array_2 -
time_start_array_2)
    #удаление в конце разв списка
    time_start_array_2 = time.time()
    for i in array_2:
        worked_ULL.delete(i)
    time_end_array_2 = time.time()
    result_time_ULL_ins_ser_del_end[2].append(time_end_array_2 -
time_start_array_2)


    worked_ULL = UnrolledLinkedList(calculate_optimal_node_size(100000))
    #100к элементов
    array_3 = np.random.randint(1, 10000, 10000)
    time_start_ULL_3 = time.time()
    #вставка в конец развернутого списка
    for i in array_3:
        worked_ULL.append(i)
    time_end_ULL_3 = time.time()
    result_time_ULL_ins_ser_del_end[0].append(time_end_ULL_3 -
time_start_ULL_3)
    #поиск в конце односвязного списка
    time_start_array_3 = time.time()
    for i in array_3:
        worked_ULL.search(i)
    time_end_array_3 = time.time()
    result_time_ULL_ins_ser_del_end[1].append(time_end_array_3 -
time_start_array_3)
    #удаление в конце односвязного списка
    time_start_array_3 = time.time()
    for i in array_3:
        worked_ULL.delete(i)
    time_end_array_3 = time.time()
    result_time_ULL_ins_ser_del_end[2].append(time_end_array_3 -
time_start_array_3)


    return result_time_ULL_ins_ser_del_end



# Размеры данных
sizes = [100, 10000, 100000]
```

```
# Время для массивов
array_times = [
    array_elapsed_time_ins_ser_del_begin(),
    array_elapsed_time_ins_ser_del_middle(),
    array_elapsed_time_ins_ser_del_end(),
]

# Время для односвязных списков
linkedlist_times = [
    linkedlist_elapsed_time_ins_ser_del_begin(),
    linkedlist_elapsed_time_ins_ser_del_middle(),
    linkedlist_elapsed_time_ins_ser_del_end(),
]

# Время для развернутых списков
ULL_times = [
    ULL_elapsed_time_ins_ser_del_begin(),
    ULL_elapsed_time_ins_ser_del_middle(),
    ULL_elapsed_time_ins_ser_del_end(),
]

# Операции
operations = [
    ('Вставка', ['Вставка в начало', 'Вставка в середину', 'Вставка в
конец']),
    ('Поиск', ['Поиск в начале', 'Поиск в середине', 'Поиск в конце']),
    ('Удаление', ['Удаление в начале', 'Удаление в середине', 'Удаление в
конце'])
]

# Создание графиков
for operation_name, operation_list in operations:
    for i, operation in enumerate(operation_list):
        plt.figure(figsize=(10, 6))

        # Время для массивов
        plt.plot(sizes, [array_times[j][i][0] for j in range(len(sizes))],
label='Array', marker='o')

        # Время для односвязных списков
        plt.plot(sizes, [linkedlist_times[j][i][0] for j in
range(len(sizes))], label='Linked List', marker='o')

        # Время для развернутых списков
        plt.plot(sizes, [ULL_times[j][i][0] for j in range(len(sizes))],
label='Unrolled Linked List', marker='o')

        plt.title(f'{operation_name}: {operation}')
        plt.xlabel('Размер структур данных')
        plt.ylabel('Время (секунды)')
        plt.xticks(sizes)
        plt.legend()
        plt.grid()
        plt.show()
```