**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.

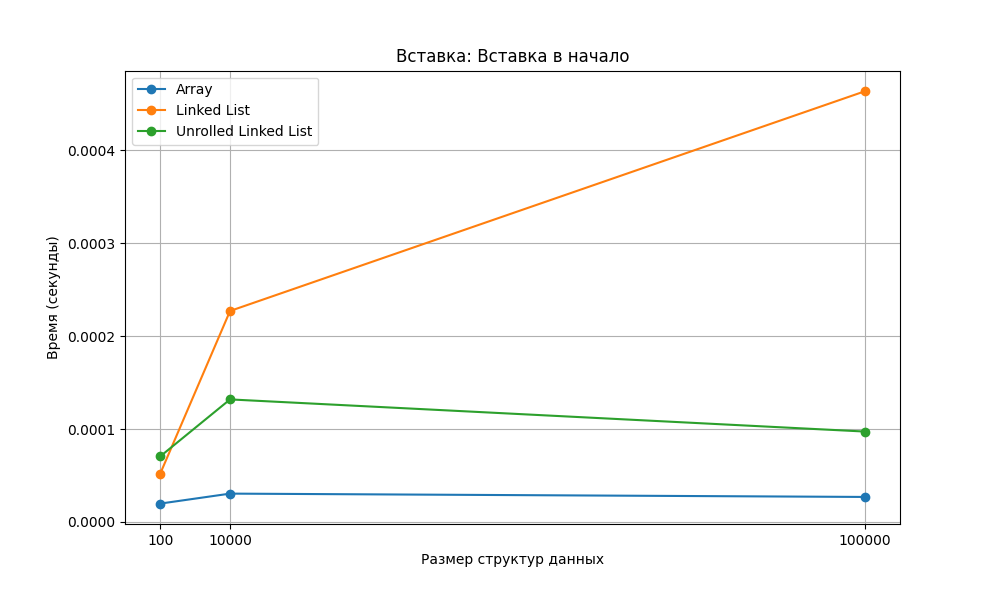
Код программ находится в Приложении А.

**Тестирование**

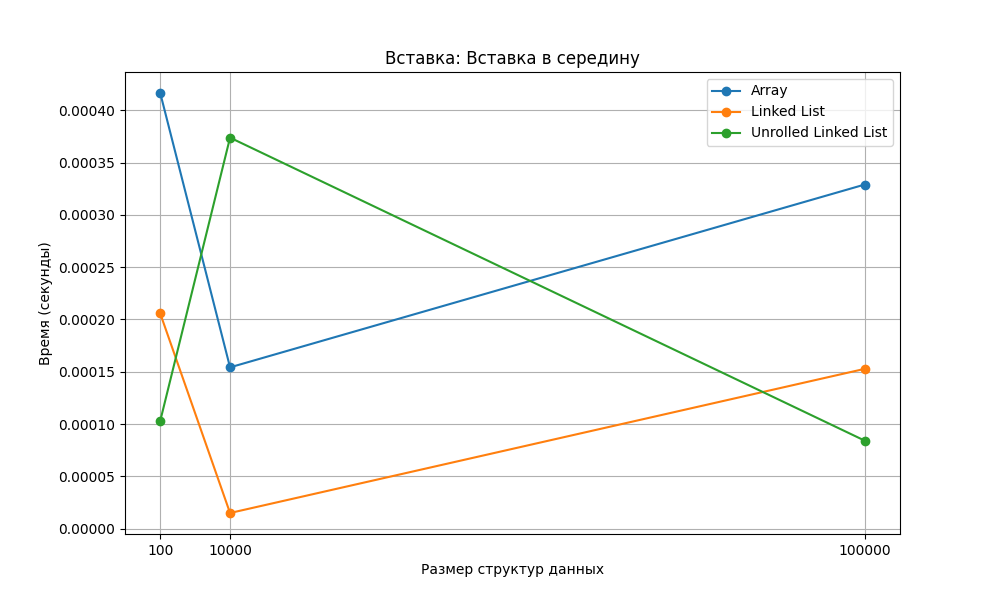
Таблица 1 — Результаты тестирования основного кода программы:

|  |  |  |  |
| --- | --- | --- | --- |
| №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  Node 8: 5 34 5  Node 9: 34 | OK |

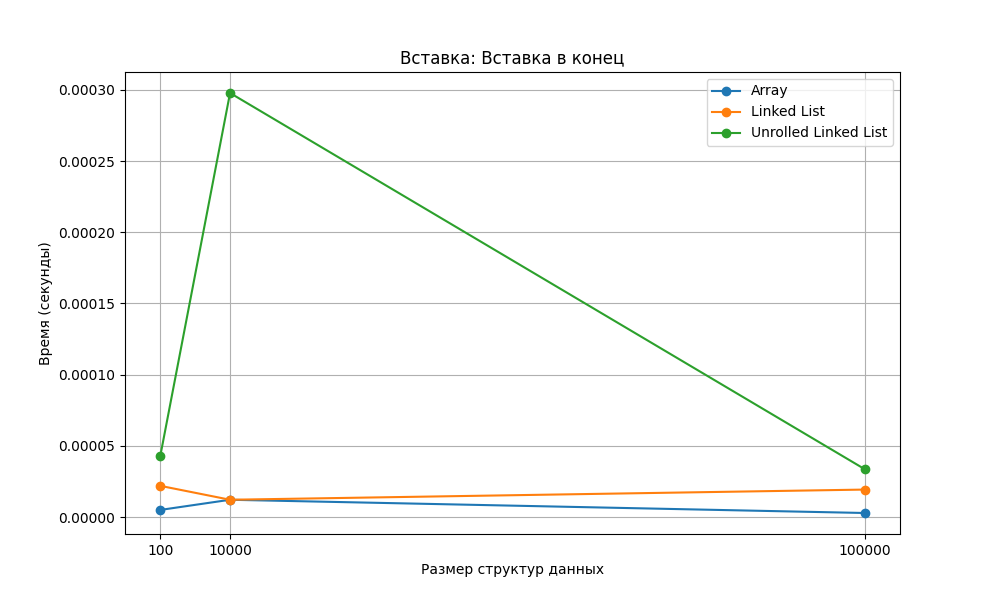
**Анализ полученных данных**

Вставка элементов в начало различных структур данных:

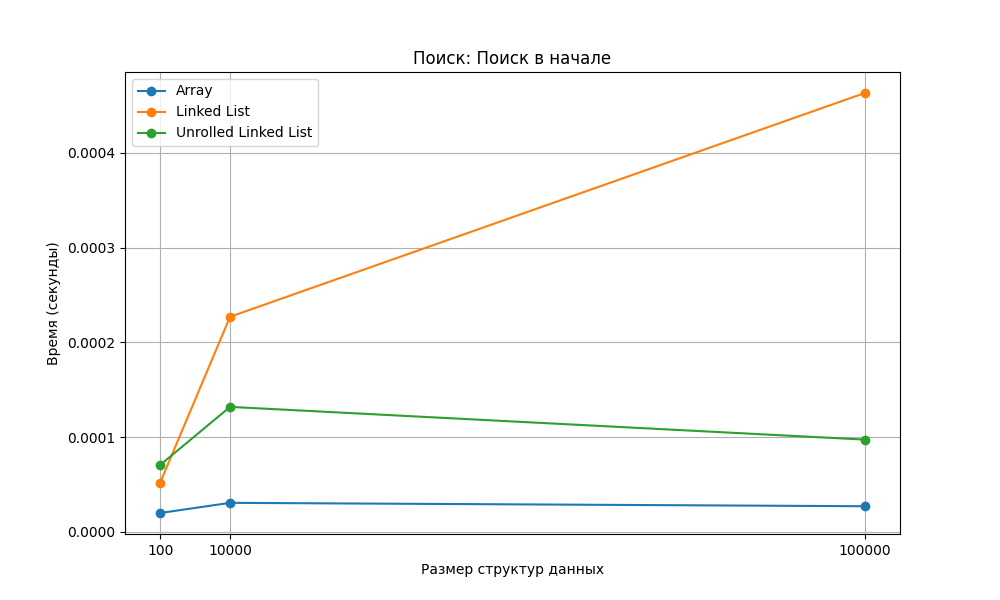
Проанализировав график, сделаем вывод: массивы работают быстрее для вставки в начало на небольших и средних размерах структуры. Связные списки менее эффективны при больших размерах, когда развёрнутые списки обеспечивают более стабильное время вставки по сравнению с обычными связными списками, особенно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

Вставка элементов в середину различных структур данных:

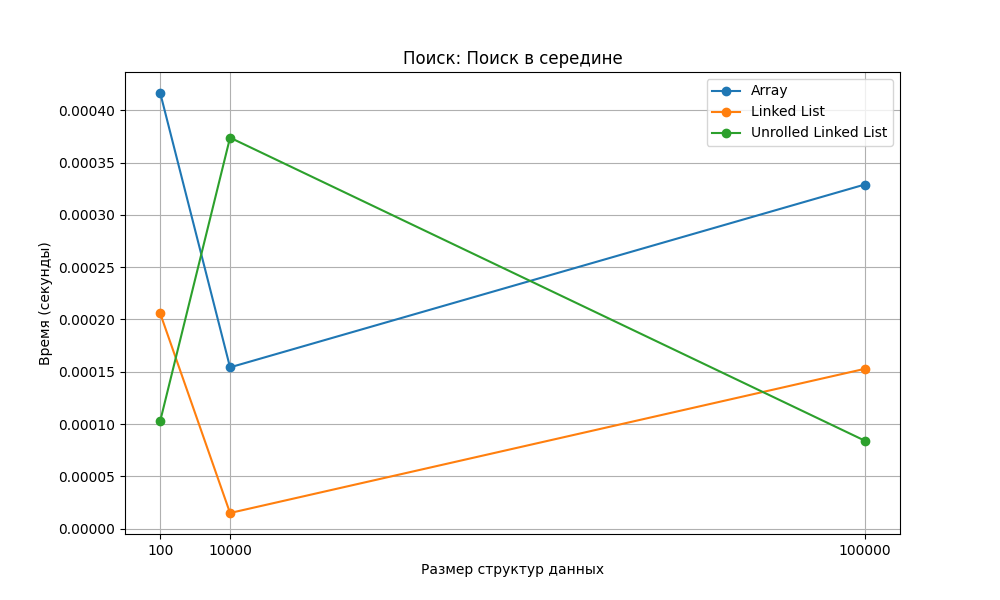
Проанализировав график, сделаем вывод: вставка элемента в середину массива имеет линейное увеличение времени, что соответствует теоретическим ожиданиям (время для массивов увеличивается с ростом размера структуры). Для связного списка время также должно увеличиваться с размером структур, однако график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

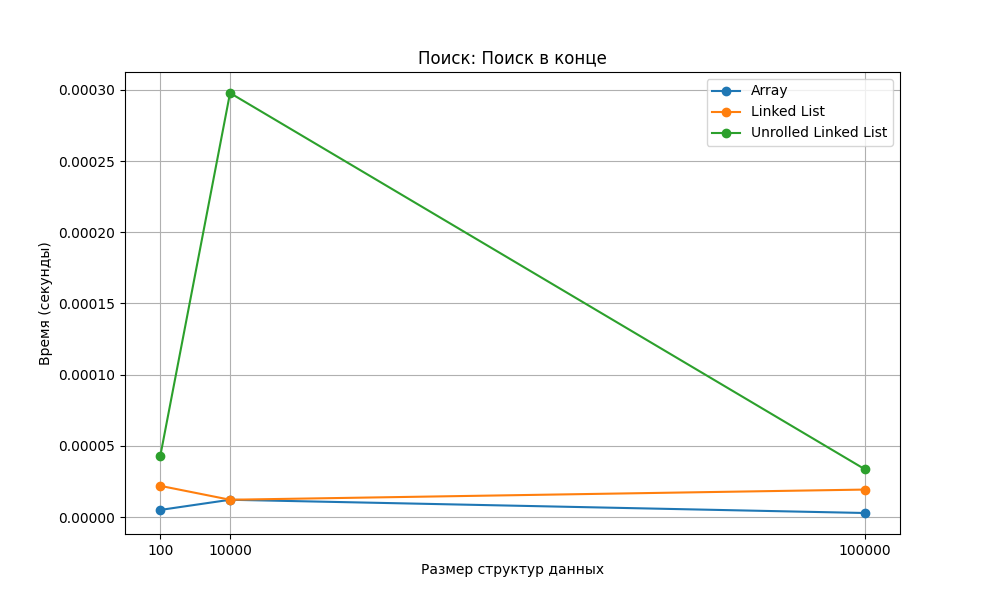
Проанализировав график, сделаем вывод: вставка в конец малых и больших объёмах данных выполняется за оптимальное время. Односвязный список лучше работает с большими данными. Развёрнутый связный список более оптимизирован для работы с большим объёмом данных, поэтому время на больших данных более стабильно,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

Поиск в начале различных структур данных:

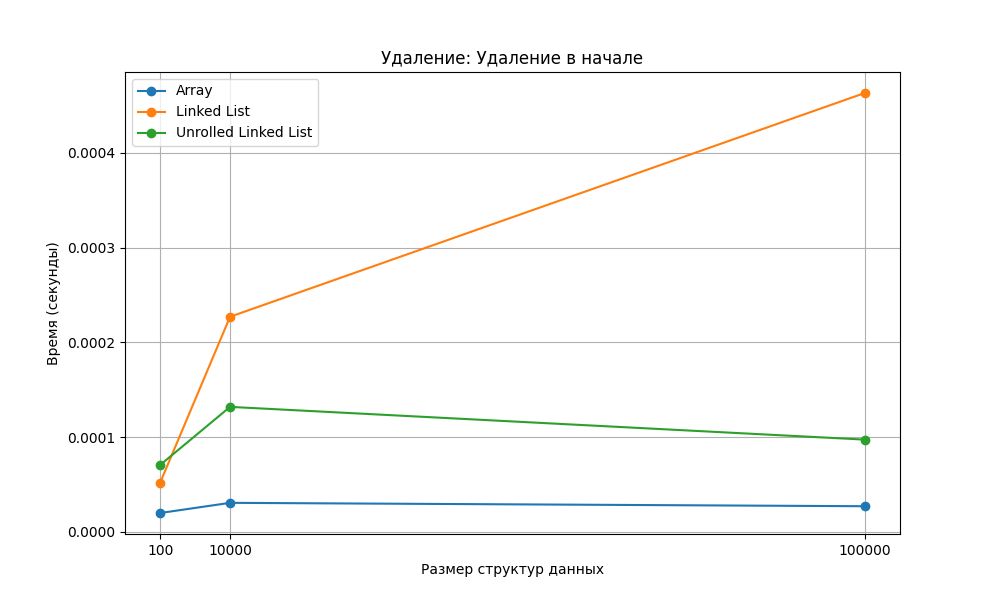
Проанализировав график, сделаем вывод: поиск в начале массива показывает наиболее стабильные и низкие результаты по времени, что связано с тем, что доступ к элементам массива происходит мгновенно благодаря индексированию. Односвязный список показывает ухудшение производительности при увеличении размера 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.

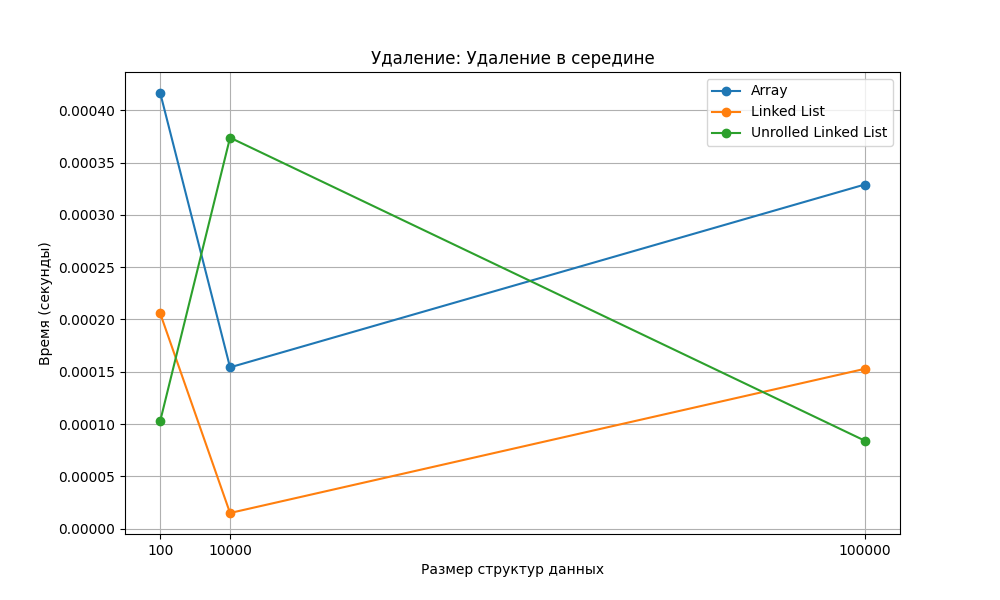
Поиск в конце различных структур данных:

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:

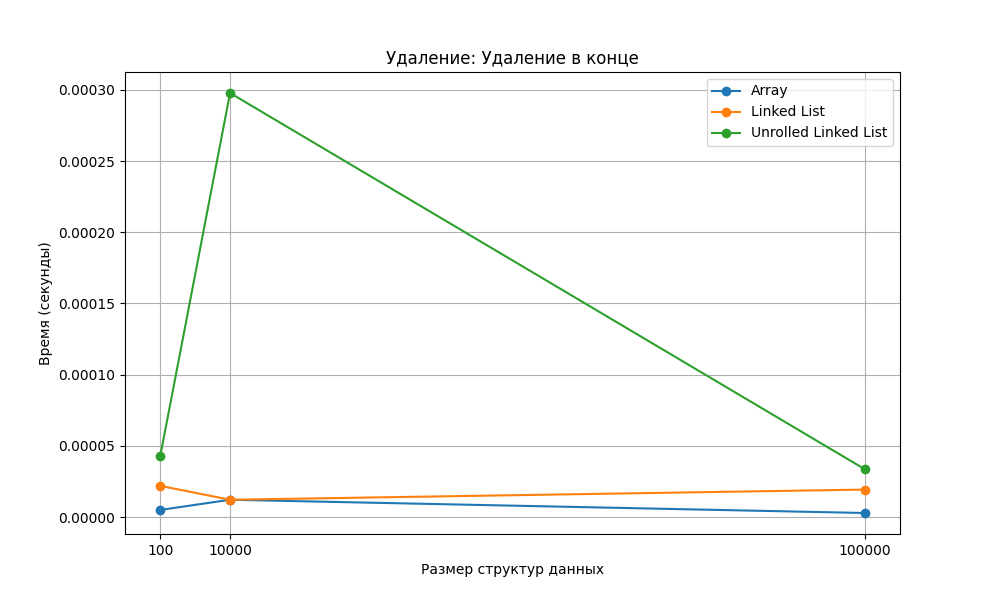
Проанализировав график, сделаем вывод: массив показывает оптимальное время для удаления элемента в начале в зависимости от размера данных. Связный список при меньших данных работает эффективнее развёрнутого списка, но уступает тому при более больших размерах данных. Развёрнутый список, в свою очередь,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