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. 3384 _____ Козьмин Н.В.

Преподаватель _____ Шестопалов Р.П.

Санкт-Петербург
2024

Цель работы.

Изучить развернутые связные списки (*ull*) и вычисление оптимального размера узлов, используя оптимизацию за счёт кэш памяти. Для *ull* разобраться в поиске, добавлении и удалении элементов, а также реализовать эти алгоритмы на Python. Измерить скорость работы реализованного класса на выборках разного объемам и разброса значений, произвести анализ полученных данных.

Задание.

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).

Основные теоретические положения.

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.

Выполнение работы.

Описание структуры кода.

Были реализованы:

- функция calculate_optimal_node_size, используя предложенную реализацию от разработчиков курса. На счёт неё есть замечания, так как она, как минимум, не учитывает в действительности размер кэша, но так на проверку ее работоспособности есть отдельный тест от создателей курса, было решено её не изменять.

- Функция filter_none принимает iterable и возвращает итератор, который избавляется от всех отсутствующих элементов (являющимися None)
- Сам UnrolledLinkedList с перечисленными далее объектами.
- Подкласс Node, содержащий текущий список и ссылку на следующий элемент. Определяем вычисление длины (количество элементов не равных None), а также метод приведения в строку, для печати.
- Инициализация ull использует контейнер, который делится на узлы; если не задана длина массива n_array, используется подсчет через функцию node_size_counter, которая сохраняется, как поля, для последующей балансировки при необходимости.
- Вычисление длины также определяем для ull, которое использует количество числовых элементов во всём ull. Строку из ull определяем так, как это требует задание.
- Булевый тип определяет наполнен ли ull элементами или нет.
- Индексация работает следующим образом: если индекс выходит за пределы – ошибка, иначе проходим до нужного узла.
- Удаление элемента по индексу работает аналогично нахождению элемента по индексу за исключением следующего: уменьшаем количество элементов в ull, удаляем элемент, используя удаление из списка, добавляем None. Затем проводим балансировку при необходимости.
- Поиск по значению проходит также, как и поиск значения по индексу, но значение индекса увеличивается, вместо уменьшения, пока это не прекратится.
- Функция container_to_cur_and_new_nodes нужна для того, чтобы передаваемый ей контенер засунуть в передаваемый узел и несколько

новых узлов за ним. Работает рекурсивно, уменьшая контейнер, пока не будет достаточно одного узла.

- Суть вставки состоит в том, чтобы либо поместить в текущую ноду элемент, либо, если нет места, создать с добавлением еще одну ноду. Для этого как раз используется функция container_to_cur_and_new_nodes. После добавления также проводим балансировку при необходимости.
- Описание балансировки: удаляем первый элемент, если он пустой (это гарантирует, что у нас не будут скапливаться пустые узлы в начале). Затем, если меняется размер нод. Проходим по каждой, удаляя следующую, если она пустая. Из каждого узла мы берем список со старым размером и делаем из него новый(-ые), используя ту же функцию container_to_cur_and_new_nodes.
- Выйдя из класса, делаем функцию check, которая требуется по заданию. А также, если код не импортирован, запускаем создание и вывод ull из введенных чисел.

Описание пайплайна.

- Данные считываются из имеющегося контейнера при инициализации, добавляются через функцию вставки (insert), также можно использовать контейнер, чтобы добавить его в уже имеющийся узел, сделав из этого совмещение данных в ull.
- Предобработка происходит при инициализации и также для этой целей можно использовать балансировку (balancing), указав её явно.
- Основная работа происходит при добавлении, удалении и поиске элементов в созданной структуре.
- Постобработка происходит после добавления и удаления элементов, используя для этого балансировку.
- Вывод данных происходит в основной поток вывода при запросе, используя для этого неявный перевод структуры (и ее подструктуры Node) в строку.

Разработанный программный код см. в приложении А.

Результаты тестирования см. в приложении Б.

Анализ полученных значений.

Результаты работы структуры с соответствующими методами представлены на рисунках 1-9.

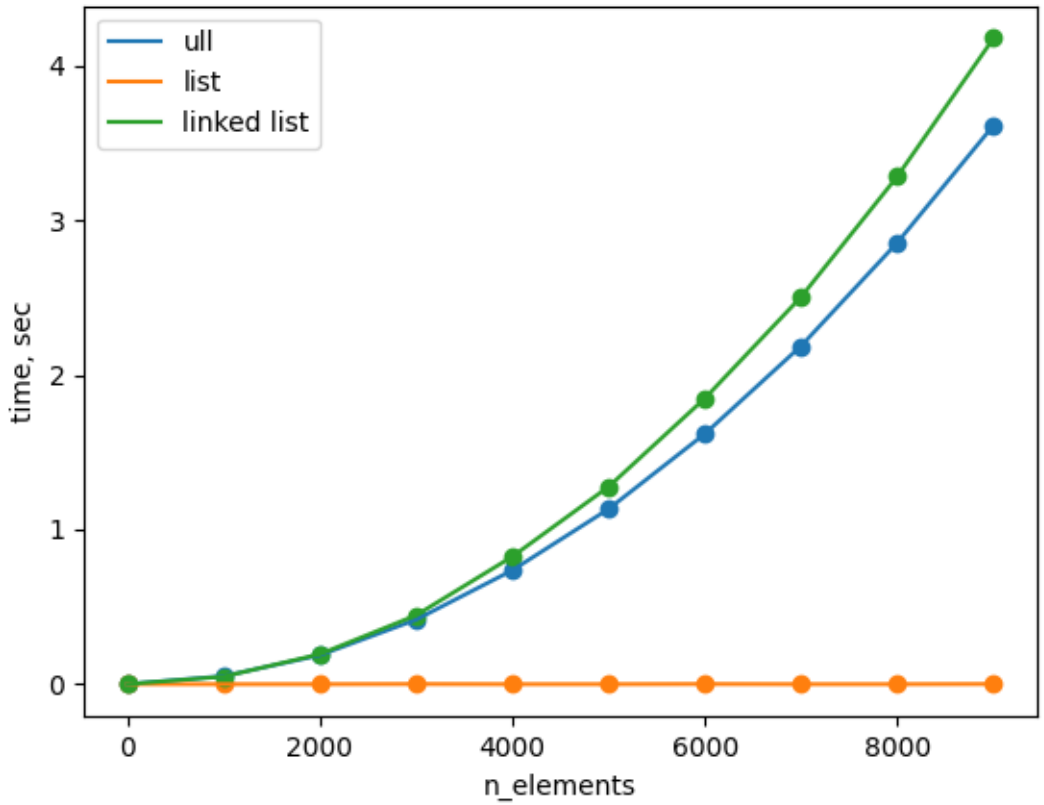

Рисунок 1 – Поиск в конце

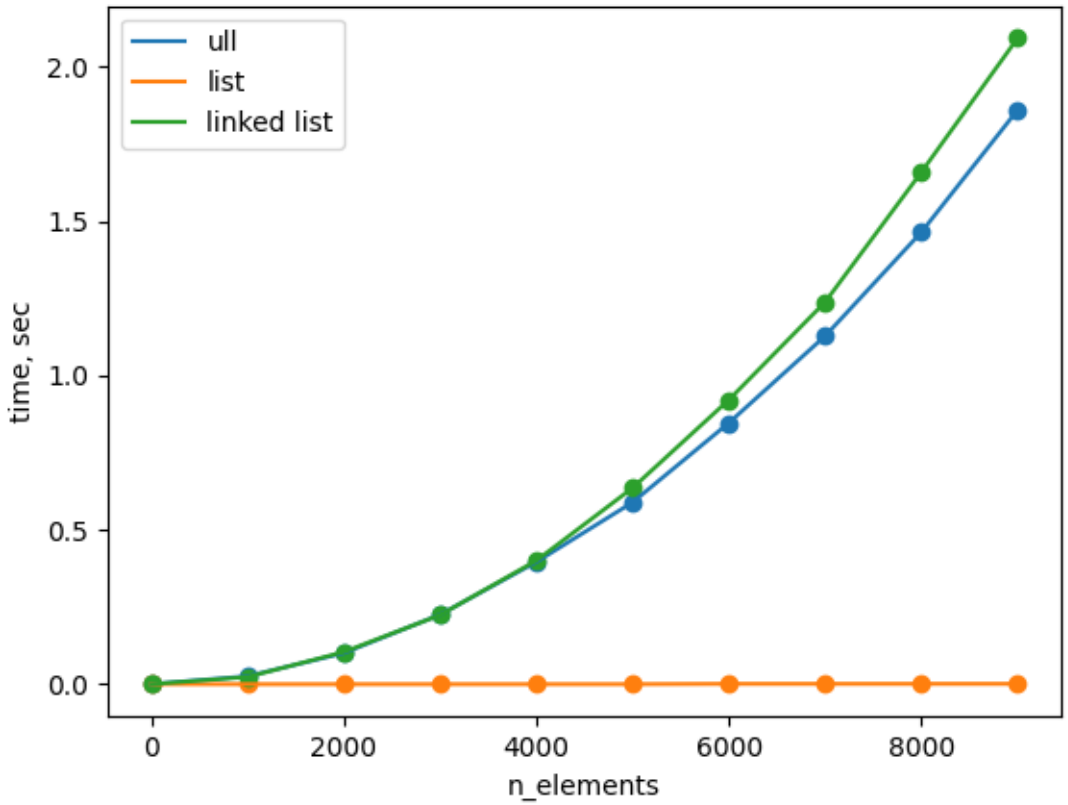

Рисунок 2 – Поиск в середине

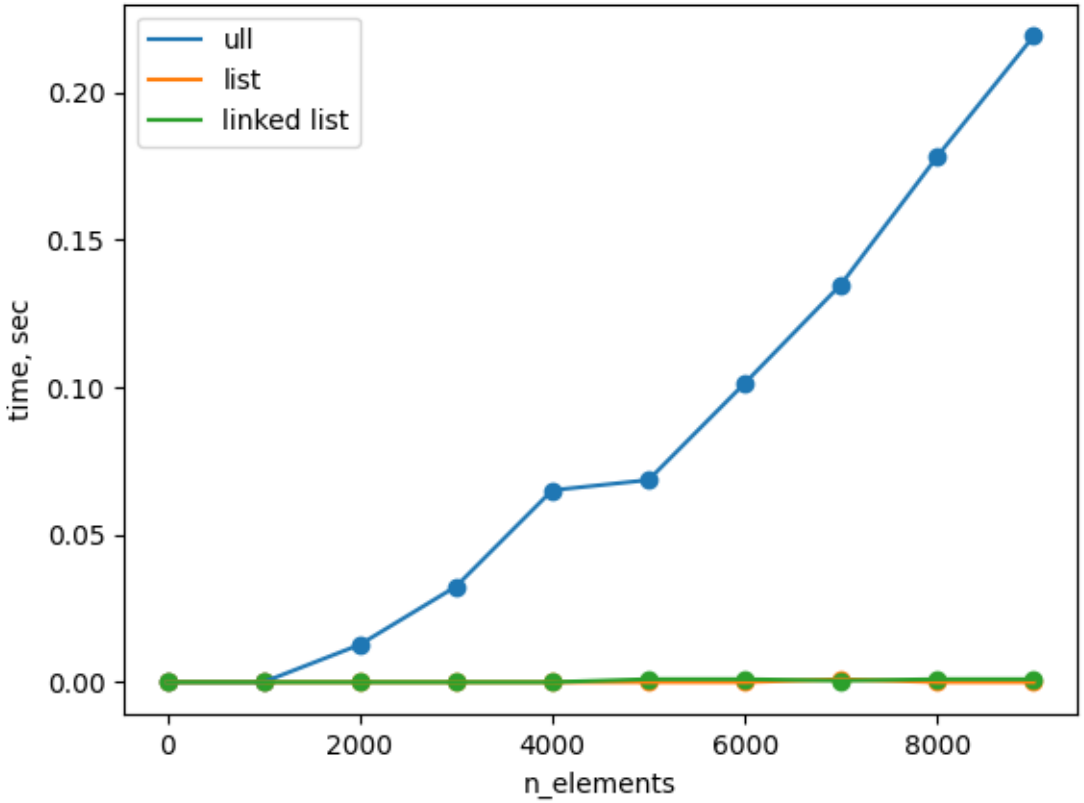

Рисунок 3 – Поиск в начале

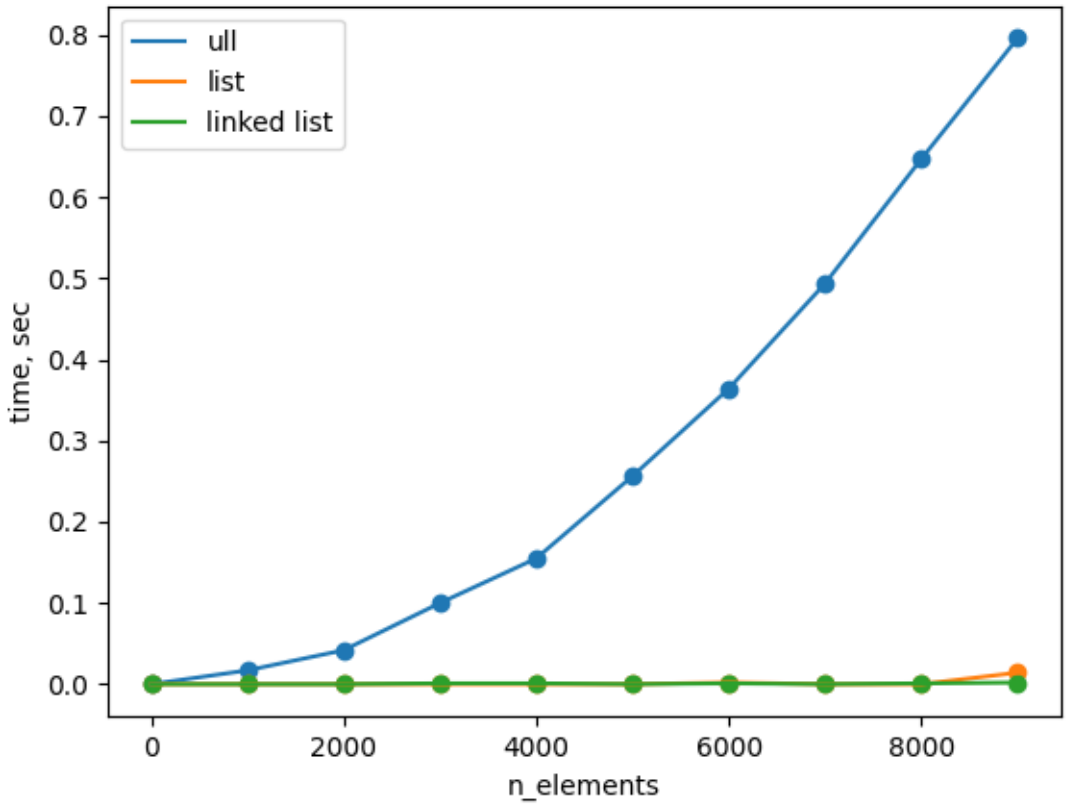

Рисунок 4 – Удаление из начала

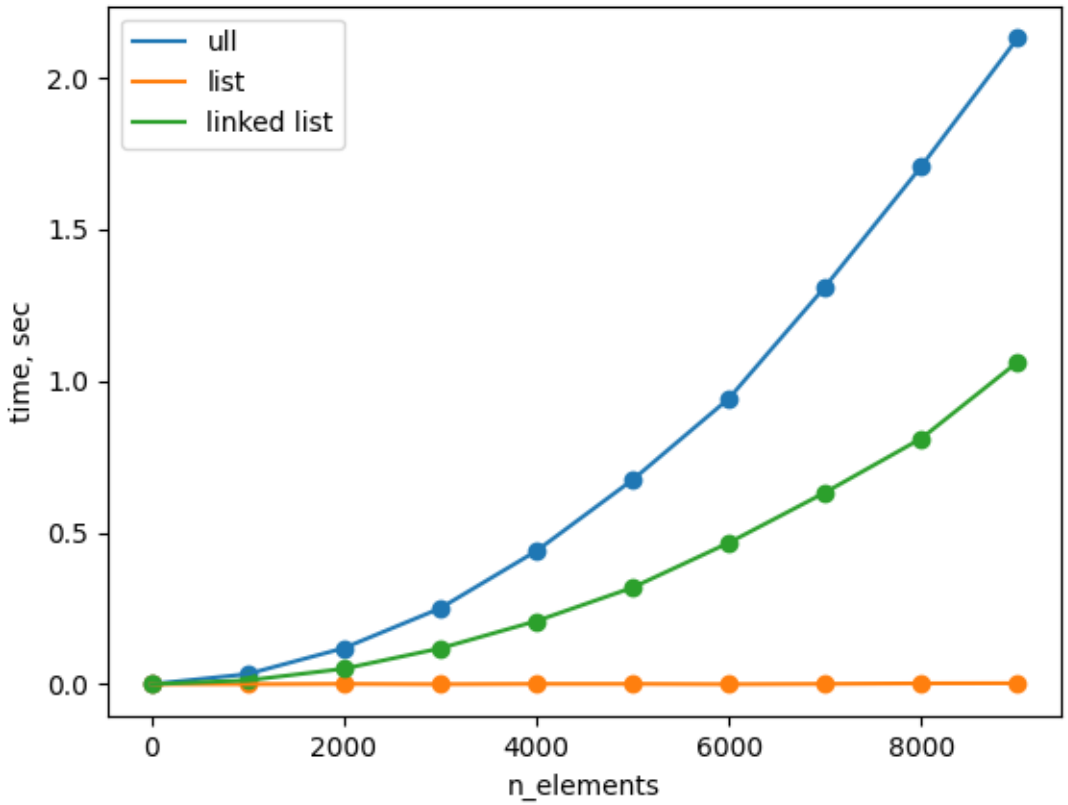

Рисунок 5 – Удаление из середины

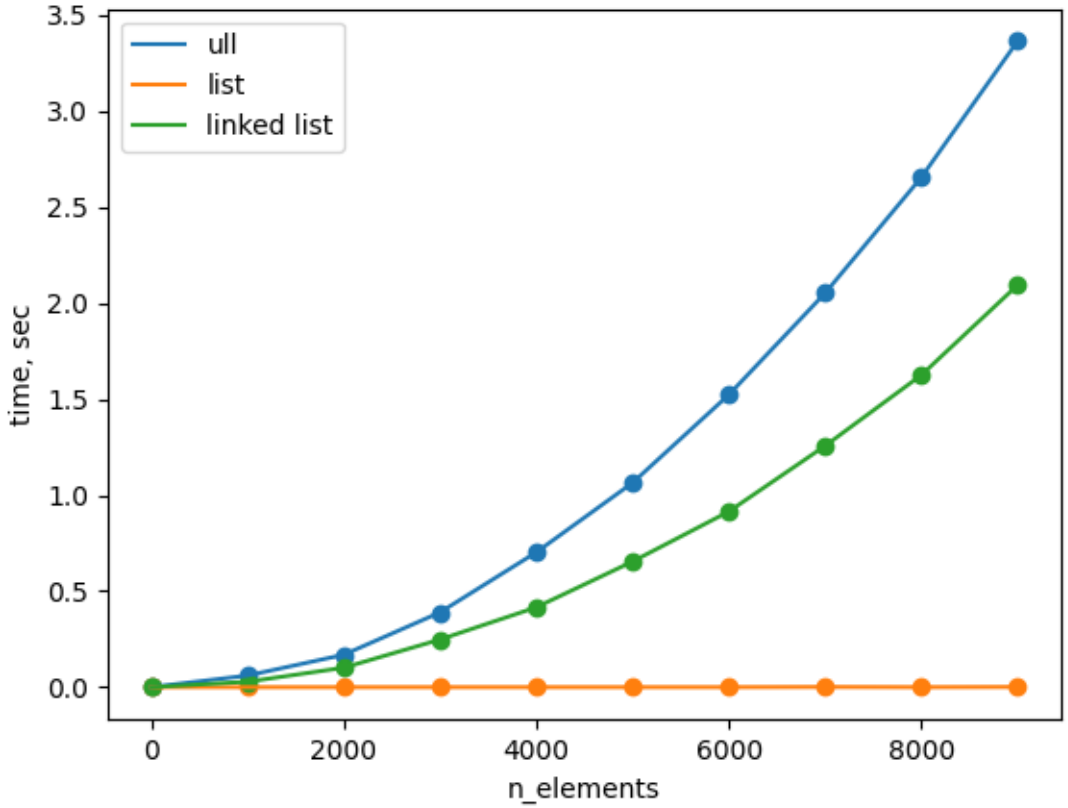

Рисунок 6 – Удаление с конца

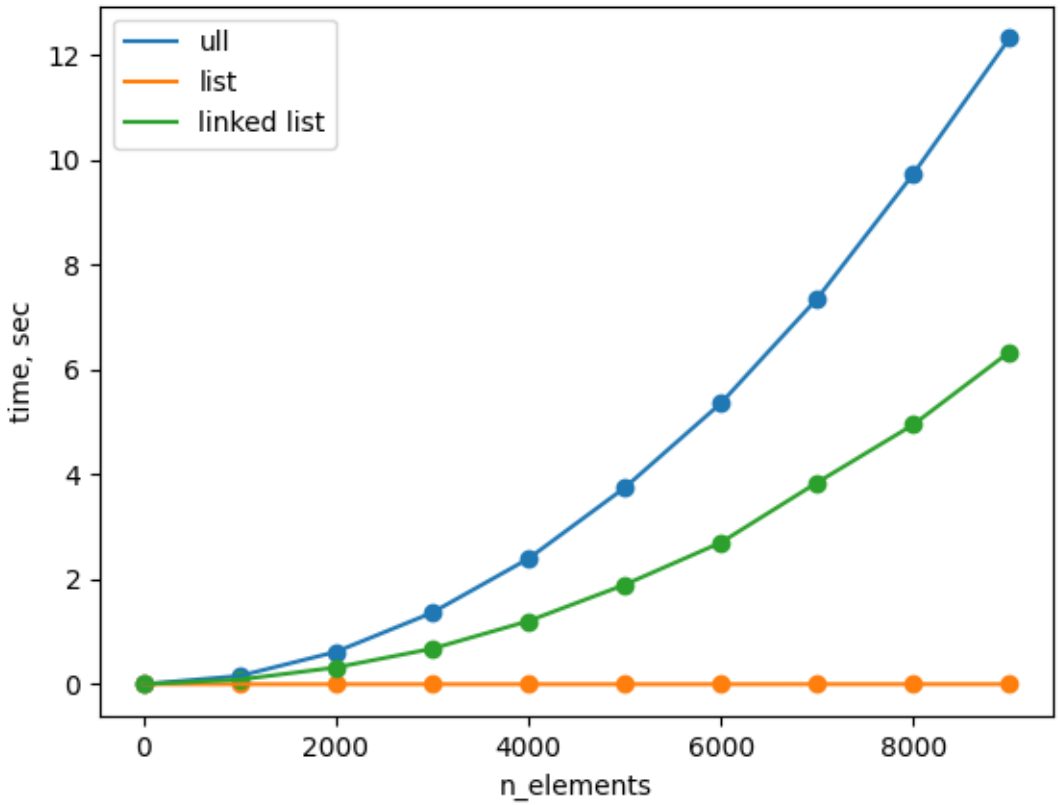

Рисунок 7 – Вставка в конец

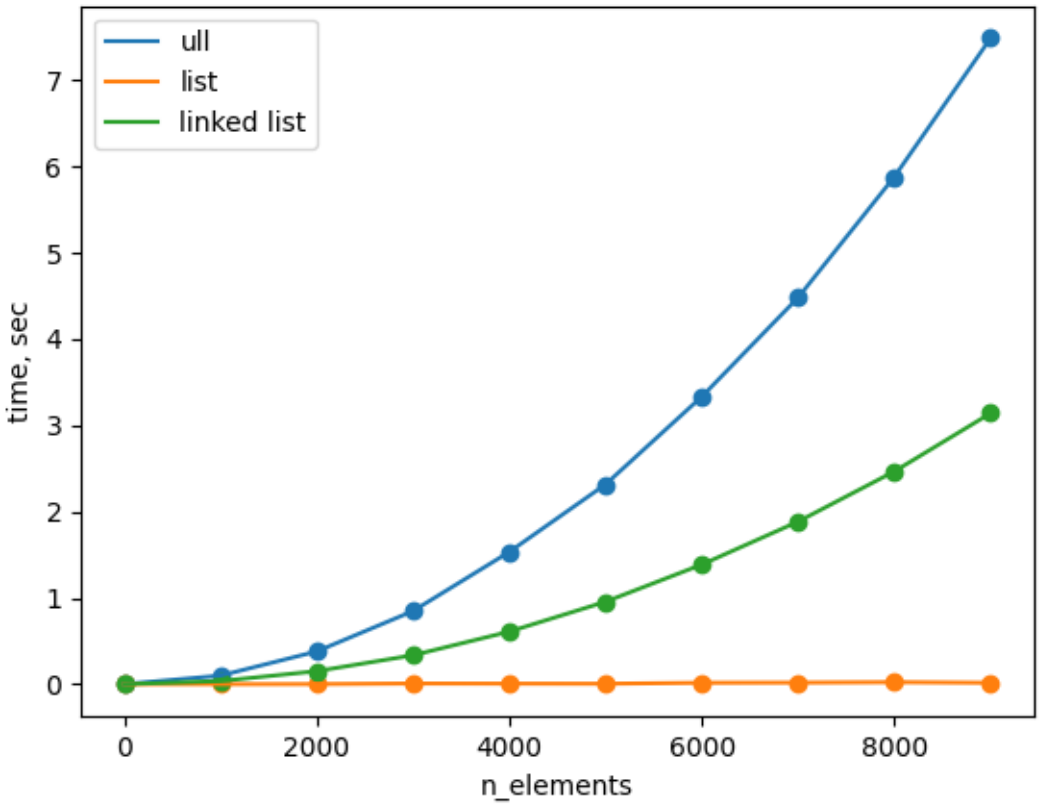

Рисунок 8 – Вставка в середину

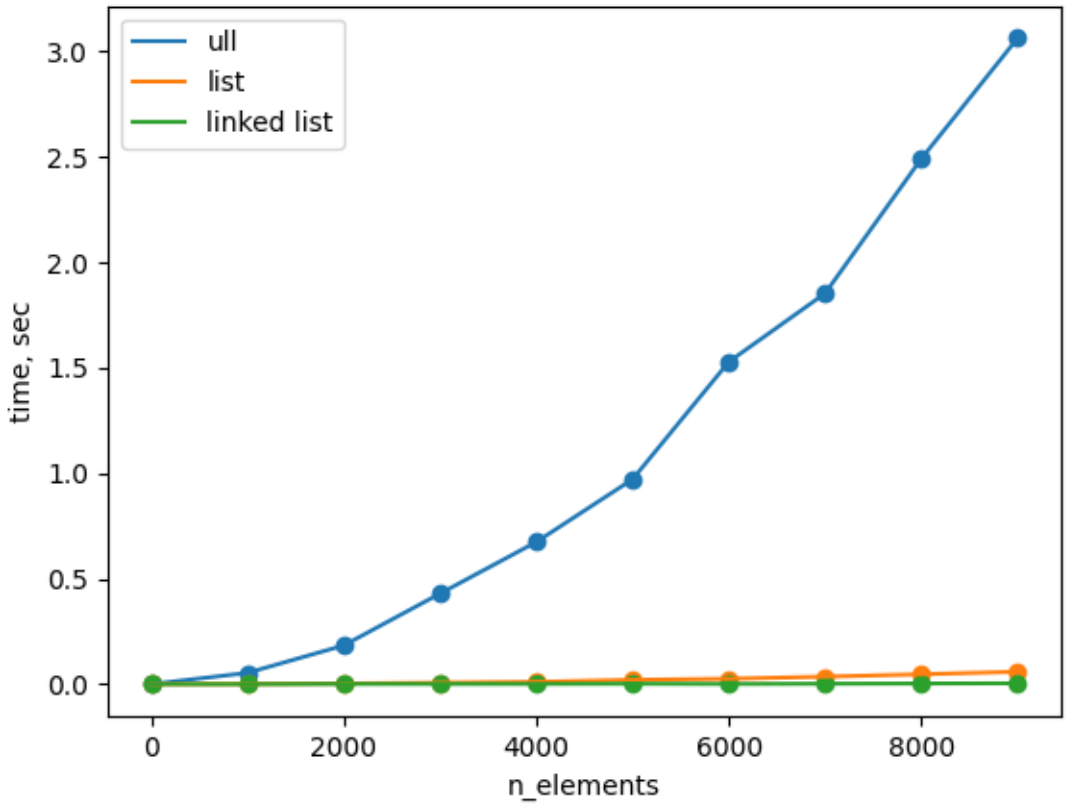

Рисунок 9 – Вставка в начало

Полученные данные можно описать следующим образом: Вставка в ull дольше остальных, по всей вероятности, из-за того, что производится балансировка. Поиск же действительно быстрее в ull, нежели в linked list.

Выводы.

В проделанной работе были изучены развернутые связные списки (*ull*) и способы нахождения оптимального размера для узлов. Также для *ull* были реализованы поиск, добавление и удалении элементов на Python вместе с сопутствующими функциями. Дополнительно была измерена скорость работы реализованного класса на выборках разного объемам и разброса значений, произведён анализ полученных данных.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: src/main.py

```
def filter_none(iterable):
    return filter(lambda data: data is not None, iterable)

class UnrolledLinkedList:
    class Node(list):
        def __init__(self, iterable) -> None:
            self.data = list(iterable)
            self.next = None

        def __len__(self) -> int:
            return len(list(filter_none(self.data)))

        def __str__(self) -> str:
            return " ".join(map(str, filter_none(self.data)))

     def calculate_optimal_node_size(num_elements, element_bytes=4,
min_cache_line_size=64) -> int:
        sum_bytes = num_elements * element_bytes
        cache_lines_needed = -(-sum_bytes // min_cache_line_size)
        optimal_node_size = cache_lines_needed+1
        return optimal_node_size

            def     __init__(self,     container=[],     n_array=0,
node_size_counter=calculate_optimal_node_size) -> None:
        self.node_size_counter = node_size_counter
        self.__len = len(container)
        if n_array <= 0:
            n_array = node_size_counter(len(self))
        self.n_array = n_array
        if len(self) == 0:
            self.num_nodes = 1
        else:
            self.num_nodes = -(-len(self) // n_array)
        i = 0
       self.head = self.Node([container[i*n_array + j] if i*n_array+j
< len(self) else None for j in range(n_array)])
        cur = self.head
        for i in range(1, self.num_nodes):
                cur.next = self.Node([container[i*n_array + j] if
i*n_array+j < len(self) else None for j in range(n_array)])
            cur = cur.next

    def __len__(self) -> int:
        return self.__len

    def __str__(self) -> str:
        rslt_str = ""
        cur = self.head
        i = 0
        while cur != None:
```

```
                rslt_str = f"{rslt_str}Node {i}: {str(cur)}\n"
                cur = cur.next
                i += 1
            return rslt_str

        def __bool__(self):
             return False if len(self.head)==0 and self.head.next==None
else True

        def __getitem__(self, index):
            if index < 0 or index >= len(self):
                raise IndexError("ull index out of range")
            cur = self.head
            for _ in range(self.num_nodes):
                dif = index - len(cur)
                if dif >= 0:
                    index = dif
                    cur = cur.next
                else:
                    return cur.data[index]

        def __delitem__(self, index):
            if index < 0 or index >= len(self):
                raise IndexError("ull index out of range")
            cur = self.head
            for _ in range(self.num_nodes):
                dif = index - len(cur)
                if dif >= 0:
                    index = dif
                    cur = cur.next
                else:
                    del cur.data[index]
                    cur.data.append(None)
                    self.__len -= 1
                    new_n_array = self.node_size_counter(len(self))
                    if new_n_array != self.n_array:
                        self.balancing(new_n_array)
                    break

        def index(self, value):
            index = 0
            cur = self.head
            for _ in range(self.num_nodes):
                if value in cur.data:
                    local_index = cur.data.index(value)
                    cur = self.head
                    return index+local_index
                index += len(cur)
                cur = cur.next
            raise ValueError(f"{value} is not in ull")

        def container_to_cur_and_new_nodes(self, container, cur: Node):
            container = list(container)
            if len(container) <= self.n_array:
               cur.data = container + [None]*(self.n_array-len(container))
                self.__len += len(cur)
                return None
```

```
                    len_container_for_two_nodes   =   min(self.n_array*2,
len(container))
                new = self.Node([None]*self.n_array)
                new.next = cur.next
                cur.next = new
                self.num_nodes += 1
                cur.data = container[:len_container_for_two_nodes//2]
                self.__len += len(cur)
                                cur.data     +=     [None]*(self.n_array-
len_container_for_two_nodes//2)
                 self.container_to_cur_and_new_nodes(container[len_container
_for_two_nodes//2:], new)

            def insert(self, index, value):
                if index < 0 or index > len(self):
                    raise IndexError("ull index out of range")
                cur = self.head
                for _ in range(self.num_nodes):
                    dif = index - len(cur)
                    if dif > 0:
                        index = dif
                        cur = cur.next
                    else:
                        if len(cur) < self.n_array:
                            del cur.data[-1]
                            cur.data.insert(index, value)
                            self.__len += 1
                        else:
                            whole_array = cur.data.copy()
                            whole_array.insert(index, value)
                            self.__len -= len(cur)
                             self.container_to_cur_and_new_nodes(filter_none
(whole_array), cur)
                        new_n_array = self.node_size_counter(len(self))
                        if new_n_array != self.n_array:
                            self.balancing(new_n_array)
                        return None
                return None

            def balancing(self, new_n_array):
                if len(self.head) == 0:
                    del_cur = self.head
                    self.head = del_cur.next
                    del del_cur
                if new_n_array == self.n_array:
                    return None
                self.n_array = new_n_array
                cur = self.head
                while cur != None:
                    if cur.next != None and len(cur.next) == 0:
                        del_cur = cur.next
                        cur.next = del_cur.next
                        del del_cur
                        self.num_nodes -= 1
                    whole_array = cur.data.copy()
                    self.__len -= len(cur)
                     self.container_to_cur_and_new_nodes(filter_none(whole_a
rray), cur)
```

```
            cur = cur.next
        return None


def check(arr_1, arr_2, n_array=0):
    ull = UnrolledLinkedList(arr_1, n_array)
    print(ull)
    for e in arr_2:
        del ull[ull.index(e)]
        print(ull)
    return [ull[i] for i in range(len(ull))]

if __name__ == "__main__":
    ull = UnrolledLinkedList(list(map(int, input().split())))
    print(ull)
```

Название файла: tests/efficiency_test.py

```
"""This file should be placed along with the executable file."""

import matplotlib.pyplot as plt
from random import randint
from main import *
import time

# Node for LinkedList
class Node:
    def __init__(self, value = None, next = None):
        self.value = value
        self.next = next

class LinkedList:
    def __init__(self):
        self.first = None
        self.last = None
        self.length = 0

    def __str__(self):
        if self.first != None:
            current = self.first
            out = 'LinkedList [\n' +str(current.value) +'\n'
            while current.next != None:
                current = current.next
                out += str(current.value) + '\n'
            return out + ']'
        return 'LinkedList []'

    def clear(self):
        self.__init__()

    def __getitem__(self, index):
        cur = self.first
        while cur != None:
            if index == 0:
                return cur.value
            index -= 1
```

```
                cur = cur.next
            raise IndexError("linked list index out of range")


    # add to end of LinkedList
        def add(self, x):
            self.length+=1
            if self.first == None:
                # self.first and self.last will point to the same memory
location
                self.last = self.first = Node(x, None)
            else:
               # here, already to different ones, because the assignment
occurred
                self.last.next = self.last = Node(x, None)

        def InsertNth(self,i,x):
            if self.first == None:
                self.last = self.first = Node(x, None)
                self.length += 1
                return
            if i == 0:
              self.first = Node(x,self.first)
              self.length += 1
              return
            curr=self.first
            count = 0
            while curr != None:
                count+=1
                if count == i:
                  curr.next = Node(x,curr.next)
                  self.length += 1
                  if curr.next.next == None:
                    self.last = curr.next
                  break
                curr = curr.next

        def Del(self,i):
            if (self.first == None):
              return
            curr = self.first
            count = 0
            if i == 0:
              self.first = self.first.next
              self.length -= 1
              return
            while curr != None:
                if count == i:
                  if curr.next == None:
                    self.last = curr
                  old.next = curr.next
                  self.length -= 1
                  break
                old = curr
                curr = curr.next
                count += 1

    if __name__ == "__main__":
```

```
set_n_elements = []
ull_times = []
list_times = []
ll_times = []
for n_elements in range(0, 10000, 1000):
    print(f"Check {n_elements} elements...")
    set_n_elements.append(n_elements)
    a = [randint(-1000, 1000) for _ in range(n_elements)]
    count = 1
    ull_local_times = []
    list_local_times = []
    ll_local_times = []
    for _ in range(count):
        ull = UnrolledLinkedList(a)
        start = time.time()
        for i in range(n_elements):
            ull.insert(0, a[i]) #len(ull)
        del ull
        end = time.time() - start
        ull_local_times.append(end)

        lst = list(a)
        start = time.time()
        for i in range(n_elements):
            lst.insert(0, a[i]) #len(lst)
        del lst
        end = time.time() - start
        list_local_times.append(end)

        ll = LinkedList()
        for i in range(n_elements):
            ll.InsertNth(i-1, a[i])
        start = time.time()
        for i in range(n_elements):
            ll.InsertNth(0, a[i]) #ll.length
        del ll
        end = time.time() - start
        ll_local_times.append(end)
    end = sum(ull_local_times) / count
    ull_times.append(end)
    end = sum(list_local_times) / count
    list_times.append(end)
    end = sum(ll_local_times) / count
    ll_times.append(end)
plt.scatter(set_n_elements, ull_times)
plt.plot(set_n_elements, ull_times, label='ull')
plt.scatter(set_n_elements, list_times)
plt.plot(set_n_elements, list_times, label='list')
plt.scatter(set_n_elements, ll_times)
plt.plot(set_n_elements, ll_times, label='linked list')
plt.xlabel("n_elements")
plt.ylabel("time, sec")
plt.legend()
plt.show()
```

Название файла: tests/ functionality_test.py

```
"""This file should be placed along with the executable file."""

import pytest
from main import *

@pytest.mark.parametrize('arr1, arr2, expected',
                    [
                        ([1, 2, 3, 4], [2, 4], [1, 3]),
                        ([1, 2, 3, 4], [], [1, 2, 3, 4]),
                        ([], [], []),
                        ([1, 2, 3, 4], [1, 2, 3, 4], []),
                    ])
def test_check_values(arr1, arr2, expected):
    assert check(arr1, arr2) == expected

@pytest.mark.parametrize('arr1, arr2, expected',
                    [
                        ([1, 2, 3, 4], [5], ValueError),
                        ([1, 2, 3, 4], [1, 5], ValueError),
                        ([], [2], ValueError),
                    ])
def test_check_raises(arr1, arr2, expected):
    try:
        check(arr1, arr2)
    except expected:
        assert 1
    else:
        assert 0
```

ПРИЛОЖЕНИЕ Б

ТЕСТИРОВАНИЕ

Таблица 1 - Примеры тестовых случаев

№ п/п	Входные данные	Выходные данные	Комментарии
1.	[1, 2, 3, 4]	[2, 4]	OK [1, 3]
2.	[1, 2, 3, 4]	[]	OK [1, 2, 3, 4]
3.	[]	[]	OK []
4.	[1, 2, 3, 4]	[1, 2, 3, 4]	OK []
5.	[1, 2, 3, 4]	[5]	OK ValueError
6.	[1, 2, 3, 4]	[1, 5]	OK ValueError
7.	[]	[2]	OK ValueError