**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.

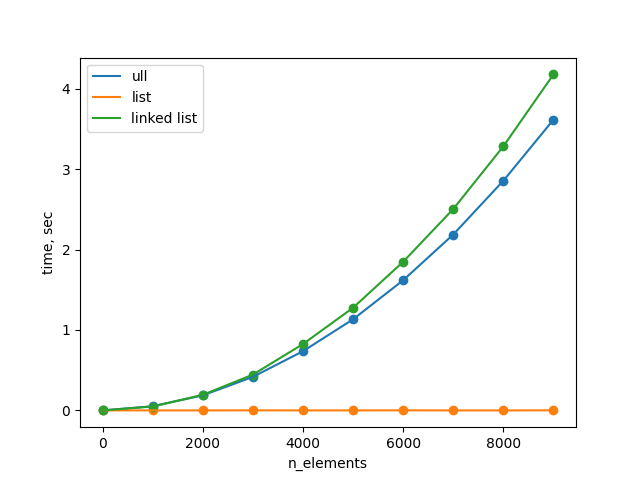


Рисунок 1 – Поиск в конце

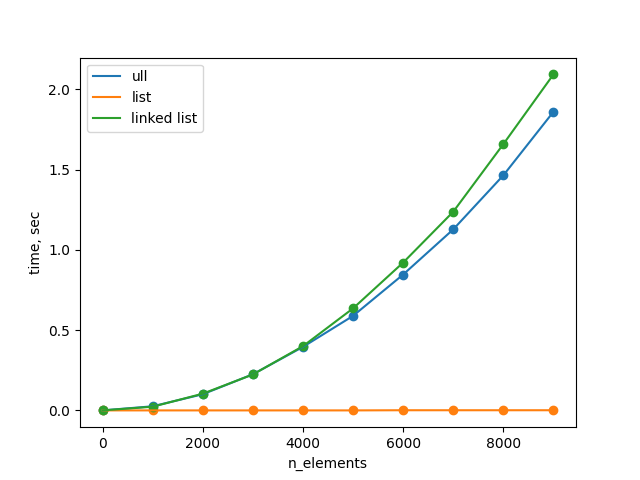


Рисунок 2 – Поиск в середине

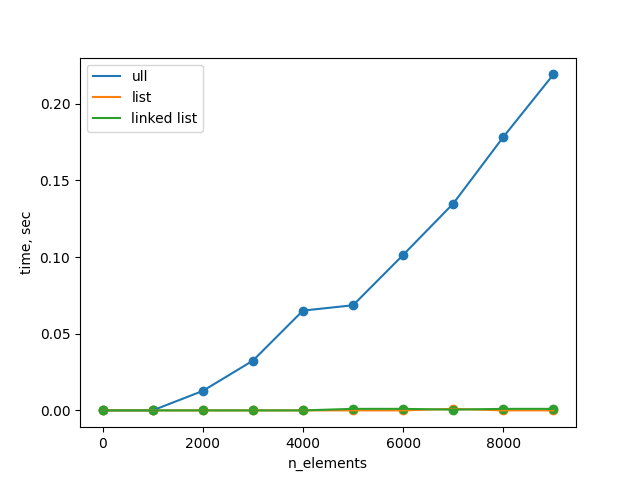


Рисунок 3 – Поиск в начале

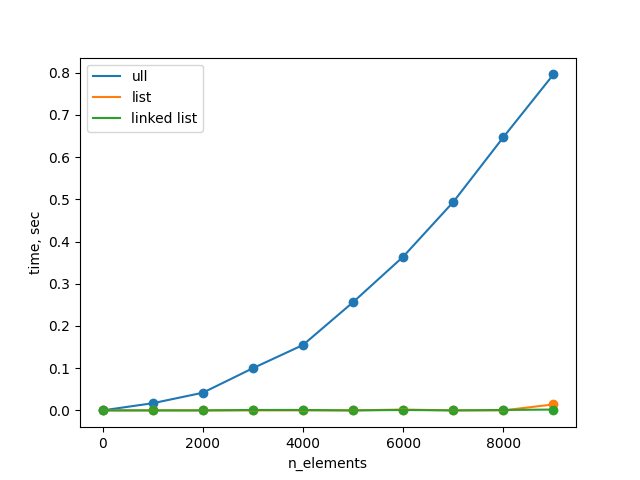


Рисунок 4 – Удаление из начала

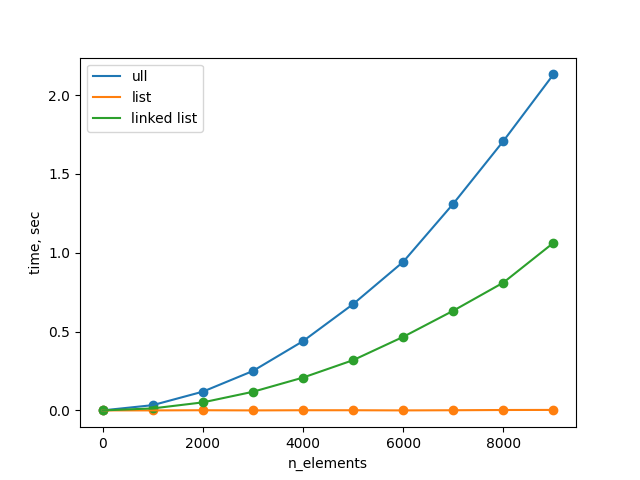


Рисунок 5 – Удаление из середины

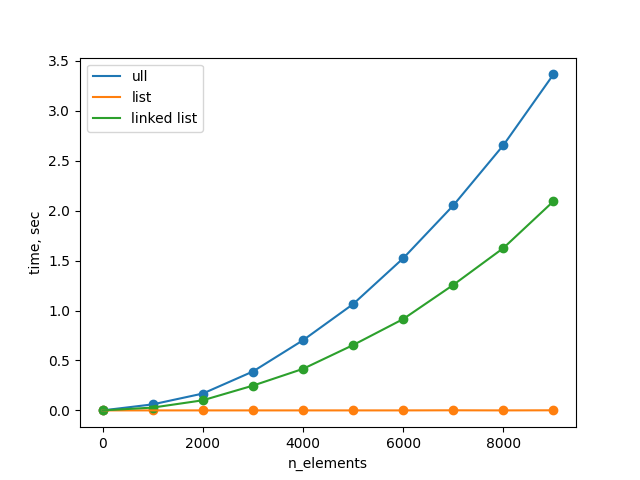


Рисунок 6 – Удаление с конца

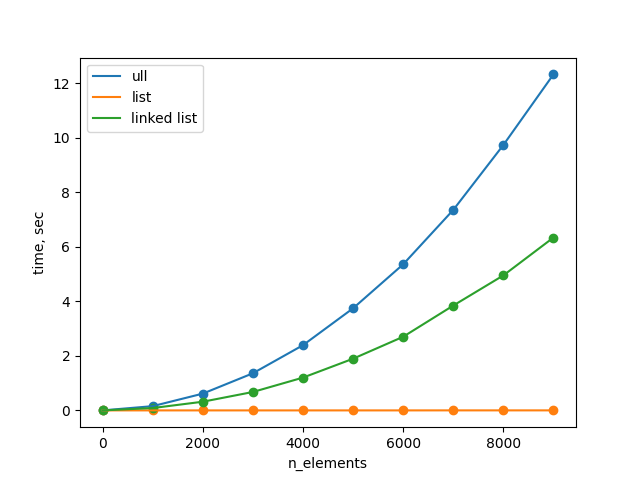


Рисунок 7 – Вставка в конец

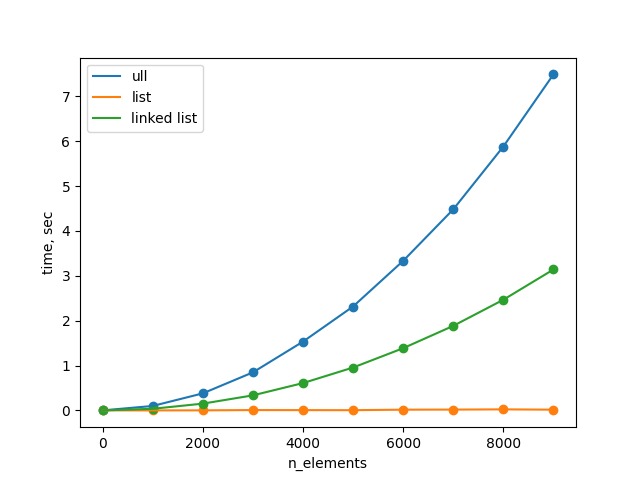


Рисунок 8 – Вставка в середину

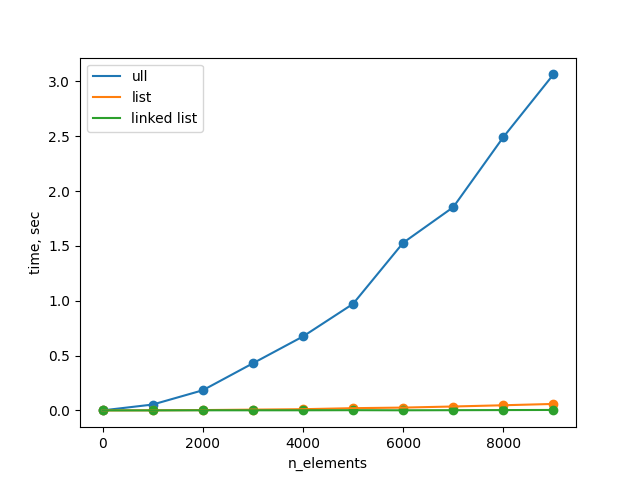


Рисунок 9 – Вставка в начало

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\_array), 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, 2, 3, 4] | [2, 4] | ОК [1, 3] |
|  | [1, 2, 3, 4] | [] | ОК [1, 2, 3, 4] |
|  | [] | [] | ОК [] |
|  | [1, 2, 3, 4] | [1, 2, 3, 4] | ОК [] |
|  | [1, 2, 3, 4] | [5] | ОК ValueError |
|  | [1, 2, 3, 4] | [1, 5] | ОК ValueError |
|  | [] | [2] | ОК ValueError |