# 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: Реализация и исследование развернутого связного списка

|  |  |
| --- | --- |
| Студент гр. 2384 | Кузьминых Е.М |
| Преподаватель | Иванов Д.В. |

Санкт-Петербург 2023

# Цель работы.

Написать реализацию развернутого односвязного списка на языке программирования Python.

# Задачи.

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. Так же при инициализации списка существует необязательный параметр n\_array (значение по умолчанию = 4), отвечающий за размер физического элемента (т.е. за размер массива).

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 (лучший, средний, худший). 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).

Автоматическая проверка вашего кода пока не предусмотрена, поэтому он будет проверяться на защите - загружайте в репозитории исходный код программы (весь), а также тесты для тестирования реализованного вами класса и функций.

# Выполнение работы.

Код состоит из двух классов: *Node* и *Unrolled\_linked\_list*.

Класс *Node* представляет узел, используемый для хранения элементов внутри *Unrolled Linked List*.

Узел содержит следующие поля:

*length*: Поле, которое отслеживает количество элементов в массиве *array* данного узла. При создании узла оно инициализируется нулем.

*array*: Массив, который хранит элементы данного узла. При создании узла массив пуст.

*next*: Ссылка на следующий узел в *Unrolled Linked List*. По умолчанию равно *None*.

Класс *unrolled\_linked\_list* представляет саму *Unrolled Linked List* и содержит методы для вставки, удаления и поиска элементов, а также методы для управления структурой данных.

Класс имеет следующие поля:

*n\_array*: Максимальная длина массива в каждом узле. По умолчанию равно 4.

*head*: Ссылка на первый узел в *Unrolled Linked List*. При создании списка равно *None*.

*tail*: Ссылка на последний узел в *Unrolled Linked List*. При создании списка равно *None*.

*all\_length*: Общая длина списка, то есть количество всех элементов в *Unrolled Linked List*. При создании списка равно 0.

Методы, реализованные для работы со *unrolled linked list*:

*append(self, value)*: Данный метод добавляет элемент *value* в конец списка. Если текущий последний узел полон (длина массива достигла *n\_array*), создается новый узел и элемент добавляется в него.

*insert(self, value, index)*: Этот метод вставляет элемент *value* в

указанный индекс внутри списка. Если индекс выходит за пределы списка, ничего не происходит. Если индекс больше или равен общей длине, элемент добавляется в конец списка. Если индекс в пределах списка, элемент добавляется в соответствующий узел.

*delete(self, value)*: Этот метод удаляет все вхождения элемента *value* из *Unrolled Linked List*. Если после удаления узел становится недозаполненным (длина массива меньше *n\_array // 2*), узлы объединяются.

*delete\_by\_index(self, index)*: Метод удаляет элемент по глобальному индексу. Если после удаления узел становится недозаполненным, узлы объединяются.

*search(self, value)*: Этот метод выполняет поиск элемента *value* в *Unrolled Linked List* и выводит индексы всех найденных элементов.

*search\_by\_index(self, index)*: Метод выполняет поиск элемента по глобальному индексу и выводит найденный элемент.

*print\_info(self):* Метод выводит все элементы списка в порядке их следования.

**Исследование работы Unrolled Linked List:**

Ниже предоставлены графики с сравнением времени работы реализованного развернутого списка с *list* и обычным односвязным списком.

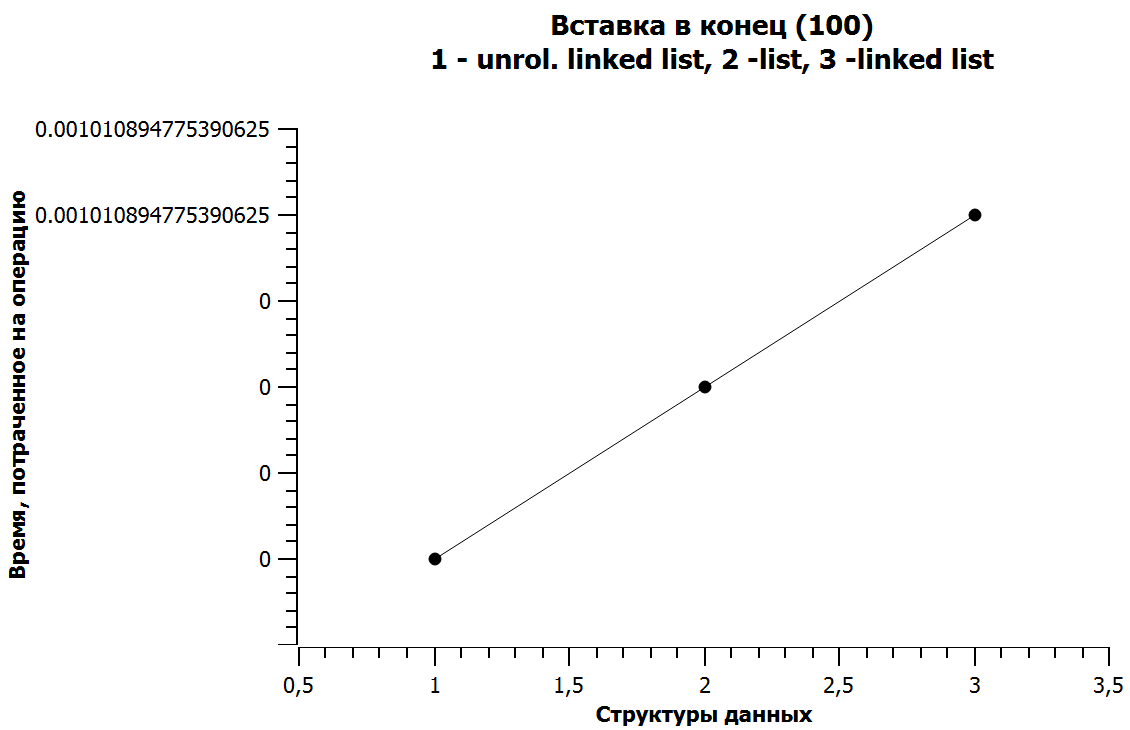


Рисунок 1 – Вставка в конец

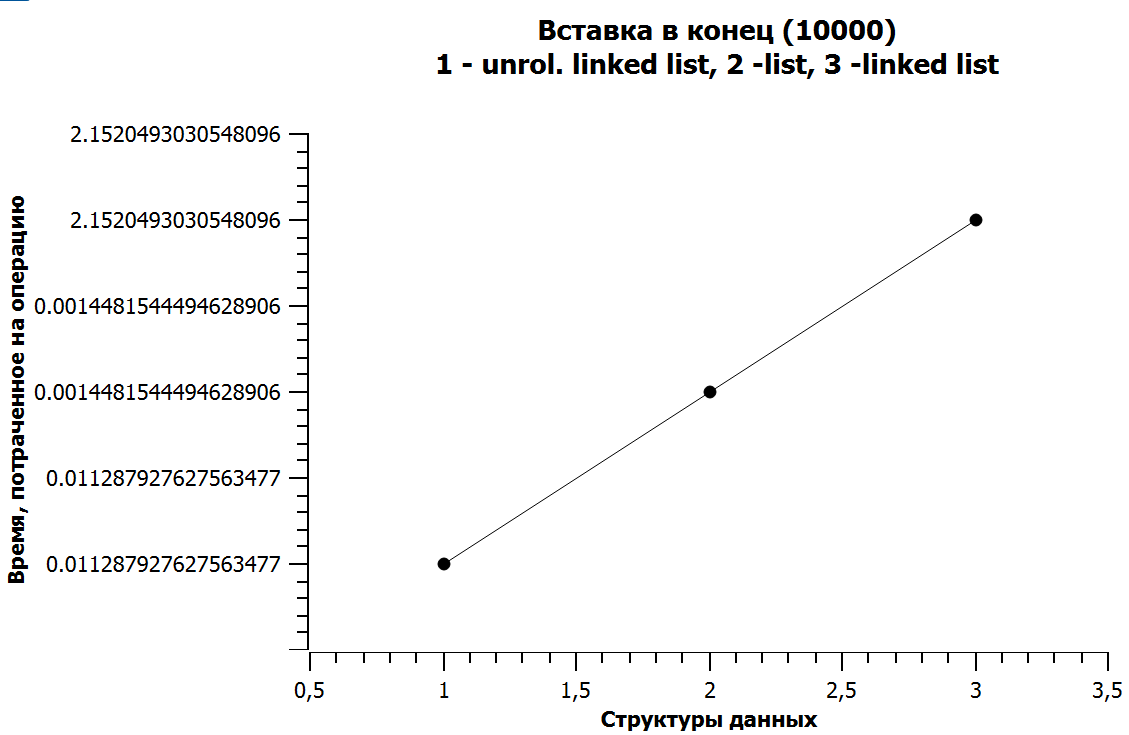


Рисунок 2 – Вставка в конец

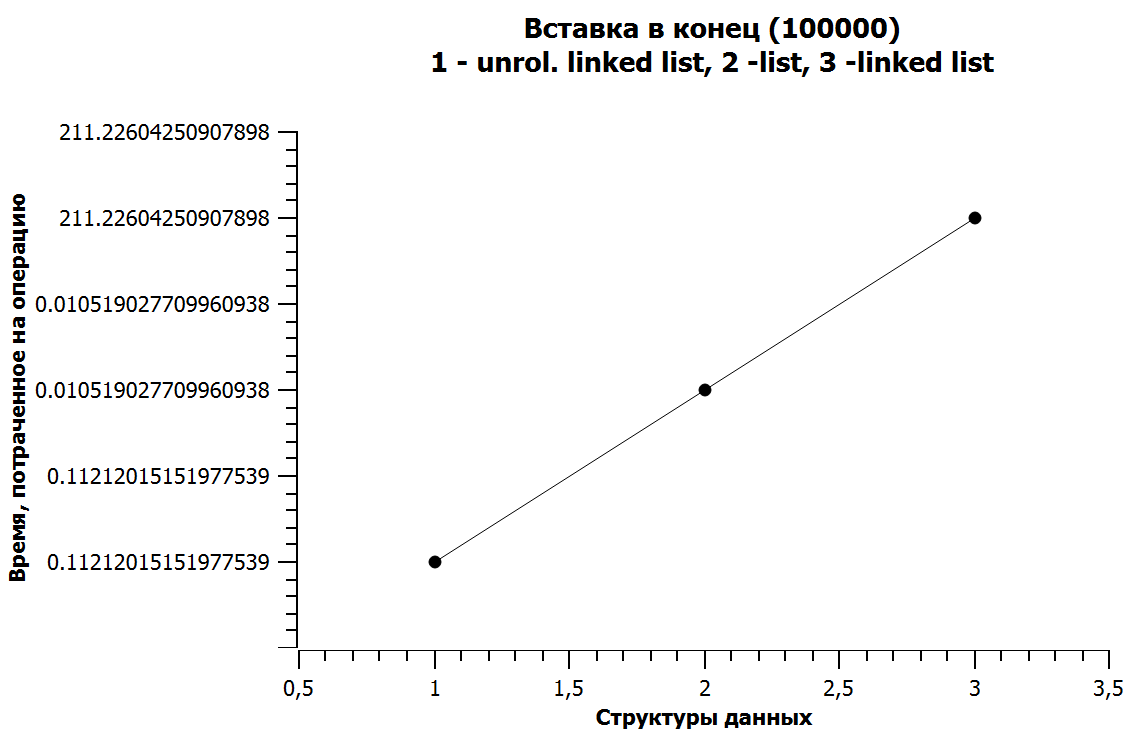
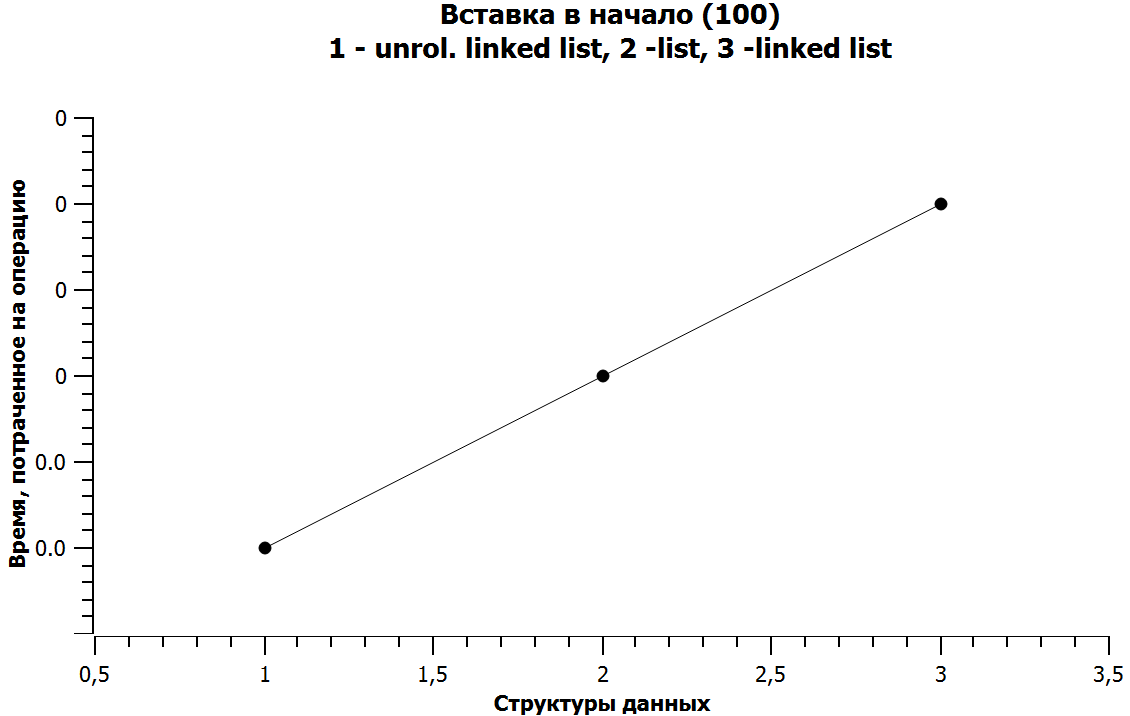


Рисунок 3 – Вставка в конец

  
Рисунок 4 – Вставка в начало

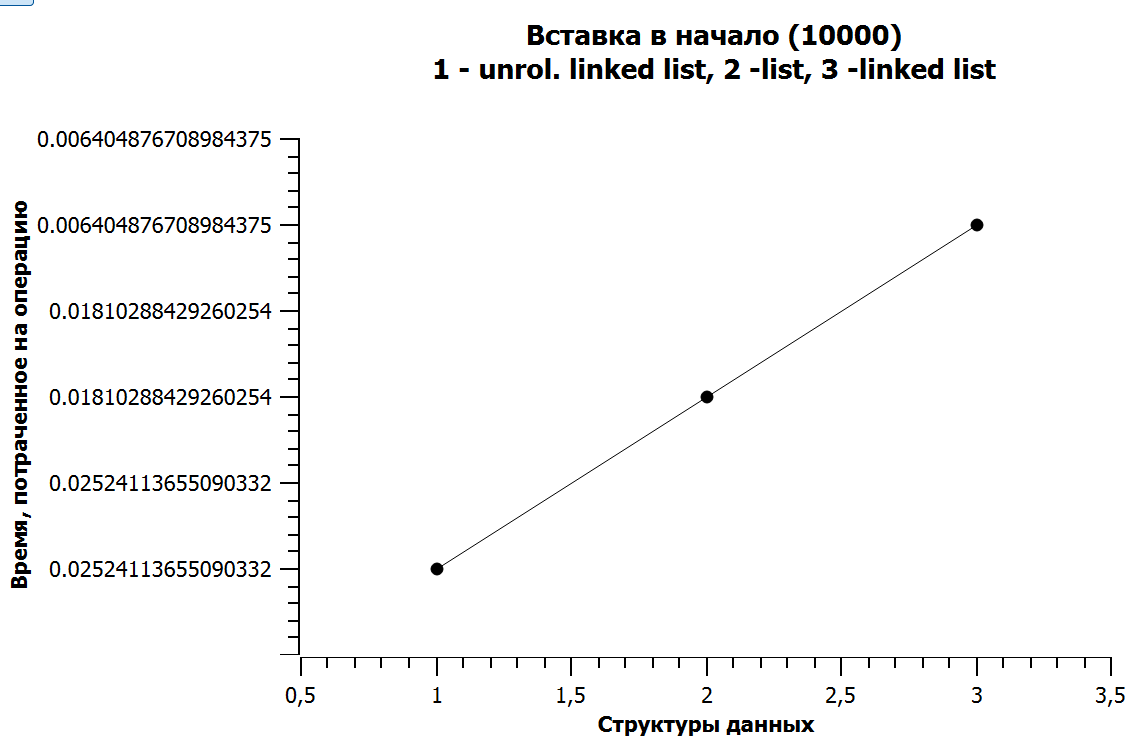


Рисунок 5 – Вставка в начало

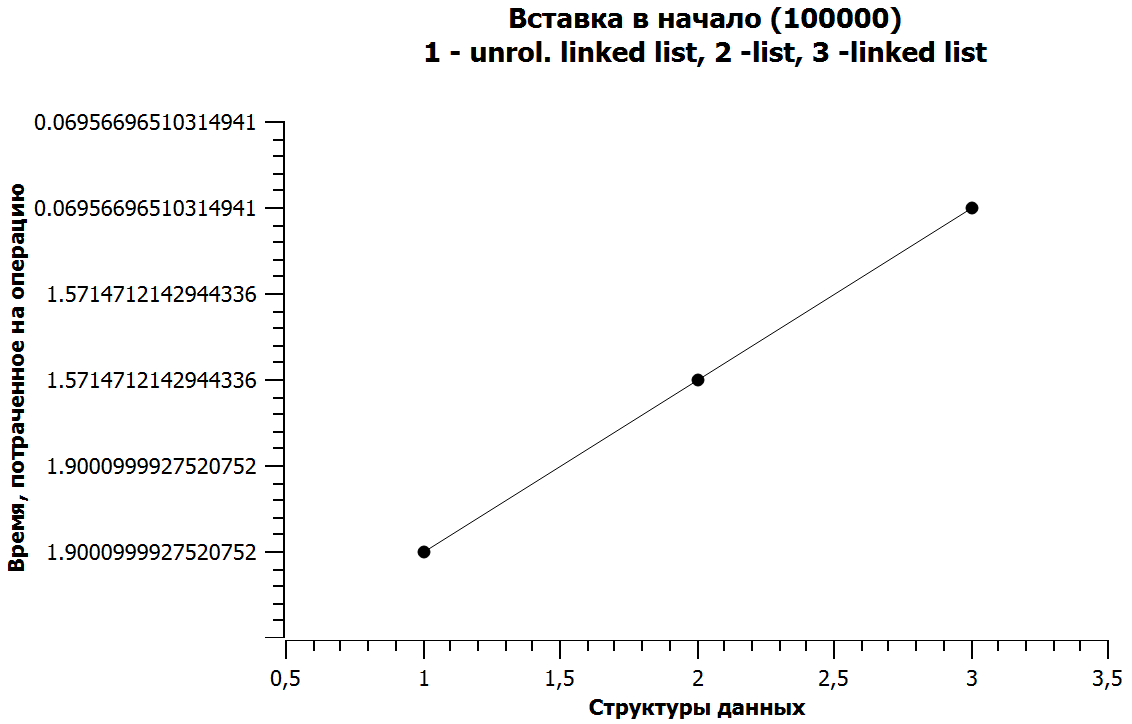


Рисунок 6 – Вставка в начало

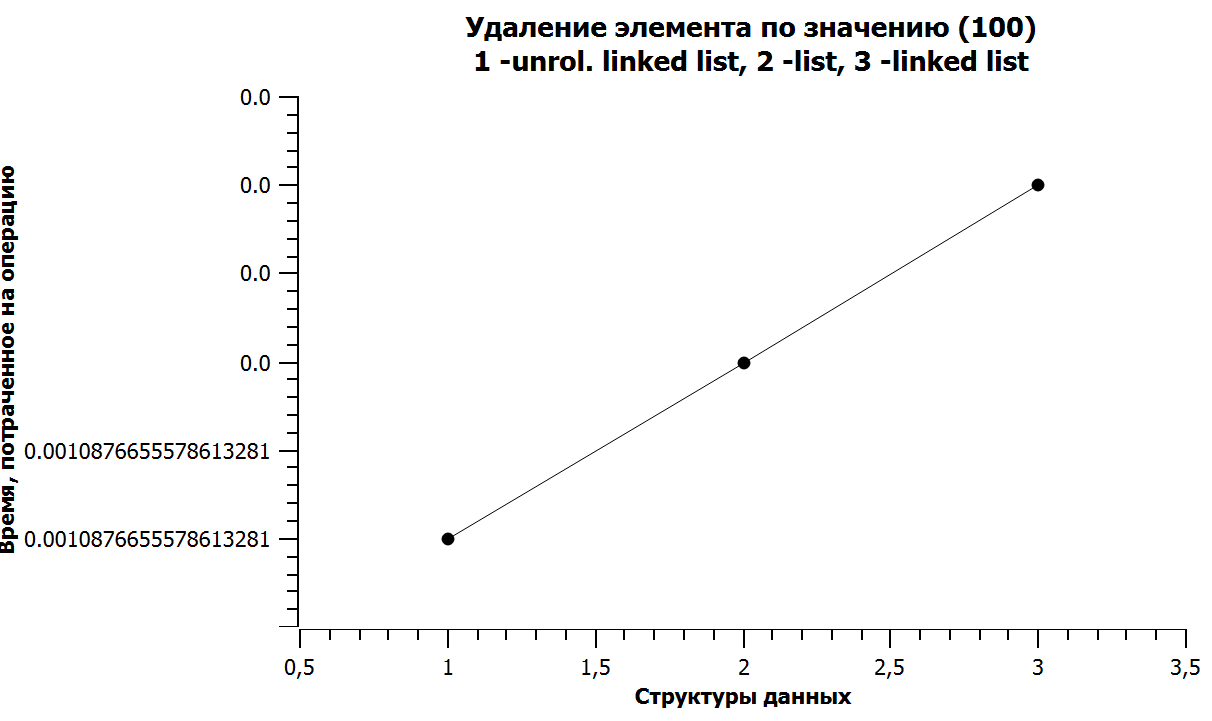


Рисунок 7 – Удаление элемента

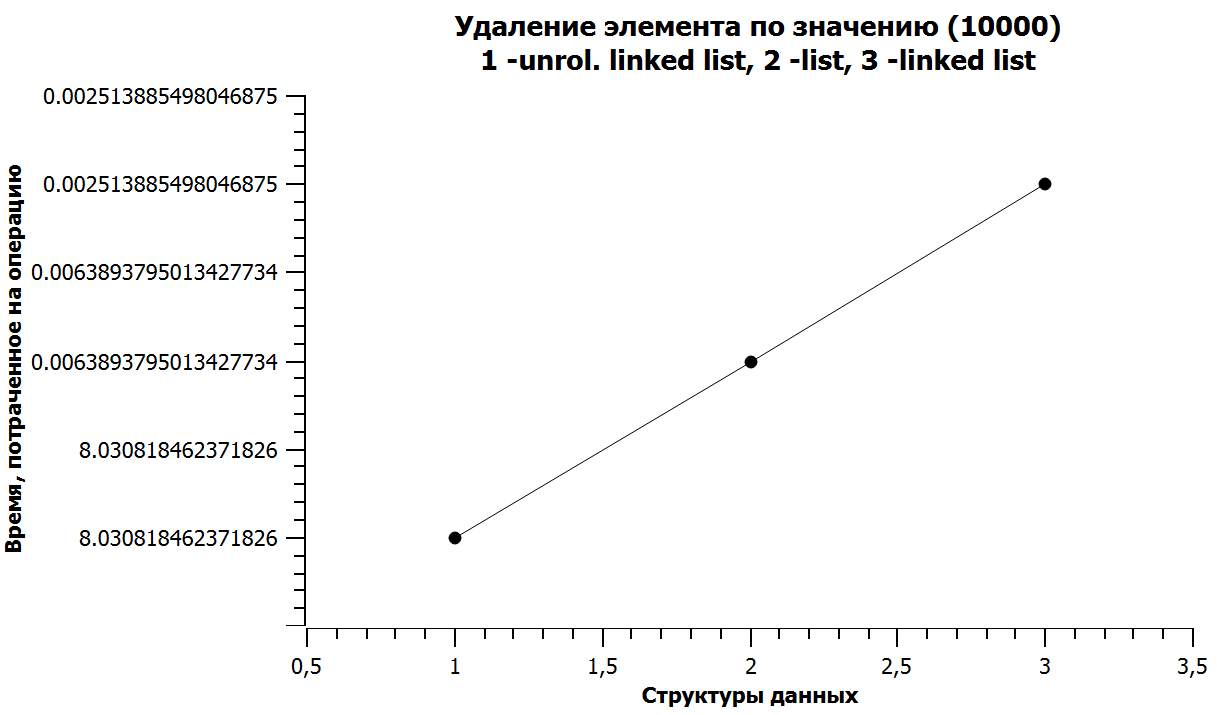


Рисунок 8 – Удаление элемента

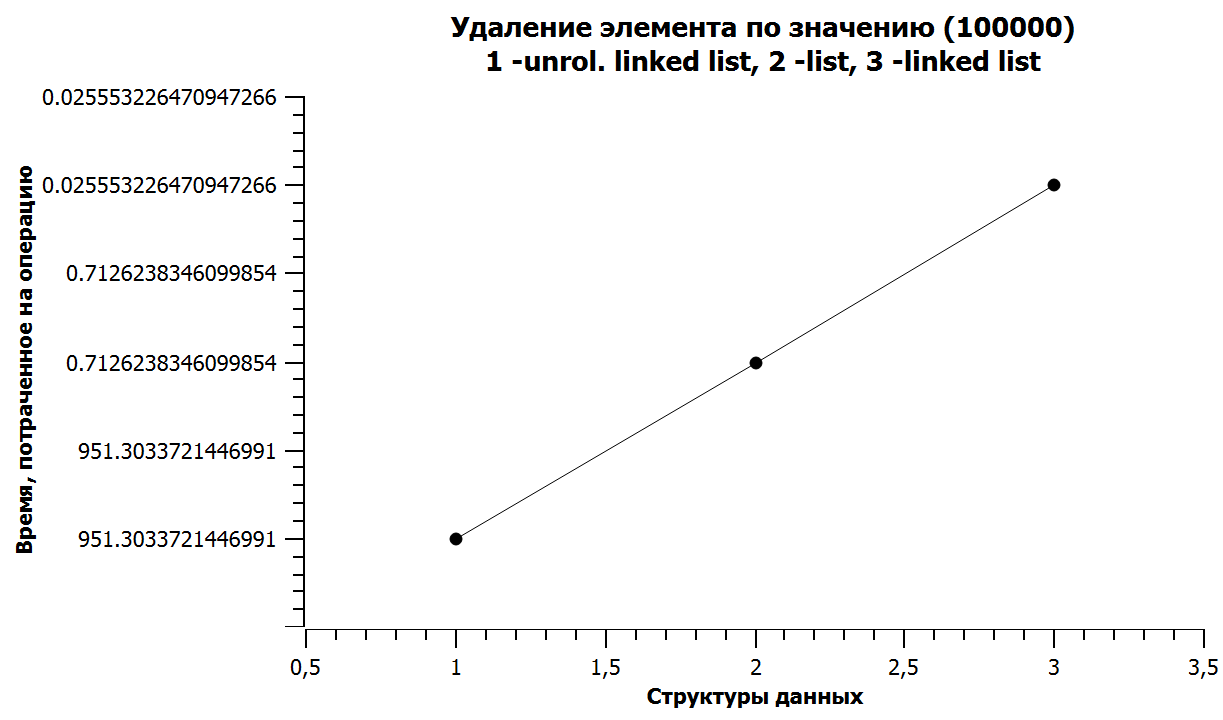


Рисунок 9 – Удаление элемента

# Тестирование.

В ходе написания лабораторной работы для тестирования был создан файл *tests.py* с функцией *check,* принимающую на вход два массива: массив *arr\_1* для заполнения структуры, массив *arr\_2* для поиска и удаления. Тесты проводились с помощью *pytest.*

|  |  |  |  |
| --- | --- | --- | --- |
| № | Входные данные | Выходные данные | Комментарий |
| 1 | def check(arr1, arr2):  my\_unrolled\_list = unrolled\_linked\_list()  for i in range(len(arr1)):  my\_unrolled\_list.append(arr1[i])  my\_unrolled\_list.print\_info()  for i in range(len(arr2)):  my\_unrolled\_list.delete(arr2[i])  my\_unrolled\_list.print\_info()  assert (my\_unrolled\_list.all\_length==0) arr1 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]  arr2 = [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]  check(arr1, arr2) | 0==0 | Ответ верный |
| 2 | def test\_assert(value):  my\_unrolled\_list = unrolled\_linked\_list()  for i in range(10):  my\_unrolled\_list.insert(value, i)  assert (my\_unrolled\_list.all\_length==10) check\_assert(100) | 10==10 | Ответ верный |
| 3 | def test\_print():  my\_unrolled\_list = unrolled\_linked\_list()  for i in range(10):  my\_unrolled\_list.append(i)  assert my\_unrolled\_list.print\_info()=='0 1 2 3 4 5 6 7 8 9 ' | '0 1 2 3 4 5 6 7 8 9 '== '0 1 2 3 4 5 6 7 8 9 ' | Ответ верный |

**Вывод.**

В ходе выполнения лабораторной работы была реализована структура данных - развернутый связный список, операции для взаимодействия с ним. Кроме этого, были написаны тесты для проверки корректности работы реализованного списка, использован *pytest* для тестирования программы. Произведен анализ скорости работы развернутого связного списка по сравнению с списком и связным списком

# Приложение А.

**Исходный код программы.**

**Main.py**

class Node:

def \_\_init\_\_(self):

self.length = 0

self.array = [0] \* self.length

self.next = None

class unrolled\_linked\_list:

def \_\_init\_\_(self, n\_array=4):

self.n\_array = n\_array

self.head = None

self.tail = None

self.all\_length = 0

def insert(self, value, index):

if index < 0 or index > self.all\_length:

return

if self.all\_length == 0 or self.all\_length <= index:

self.append(value)

else:

global\_index = 0

my\_temp = self.head

prev\_temp = None

while my\_temp is not None:

for i in range(len(my\_temp.array)):

if global\_index == index:

if my\_temp.length + 1 <= self.n\_array:

my\_temp.array.insert(i, value)

else:

new\_node = Node()

split\_index = self.n\_array // 2

new\_node.array = my\_temp.array[split\_index:]

my\_temp.array = my\_temp.array[:split\_index]

if prev\_temp is None:

self.head = new\_node

else:

prev\_temp.next = new\_node

new\_node.next = my\_temp.next

if i < split\_index:

my\_temp.array.insert(i, value)

else:

new\_node.array.insert(i - split\_index, value)

self.all\_length += 1

return

global\_index += 1

prev\_temp = my\_temp

my\_temp = my\_temp.next

def append(self, value):

if self.head == None:

self.head = Node()

self.head.array.append(value)

self.head.length += 1

self.all\_length += 1

self.tail = self.head

elif self.tail.length + 1 <= self.n\_array:

self.tail.array.append(value)

self.tail.length += 1

self.all\_length += 1

else:

new\_node = Node()

half\_length = self.tail.length // 2

mini\_array = self.tail.array[-half\_length:]

new\_node.array.extend(mini\_array)

self.tail.array = self.tail.array[:-half\_length]

new\_node.array.append(value)

new\_node.length = len(new\_node.array)

self.all\_length += 1

self.tail.length = half\_length

self.tail.next = new\_node

self.tail = new\_node

def print\_info(self):

temp = self.head

result\_string = ''

while temp != None:

for i in range(len(temp.array)):

result\_string += f'{temp.array[i]} '

temp = temp.next

return result\_string

def delete(self, value):

temp = self.head

prev\_temp = None

while temp is not None:

i = 0

while i < len(temp.array):

if temp.array[i] == value:

temp.array.pop(i)

temp.length -= 1

else:

i += 1

if len(temp.array) < self.n\_array // 2:

if prev\_temp is not None and len(prev\_temp.array) + len(temp.array) <= self.n\_array:

prev\_temp.array.extend(temp.array)

prev\_temp.next = temp.next

temp = prev\_temp

elif temp.next is not None and len(temp.array) + len(temp.next.array) <= self.n\_array:

temp.array.extend(temp.next.array)

temp.next = temp.next.next

prev\_temp = temp

temp = temp.next

if self.head is not None and len(self.head.array) == 0:

self.head = self.head.next

self.all\_length -= 1

def delete\_by\_index(self, index):

if index < 0 or index >= self.all\_length:

return

temp = self.head

prev\_temp = None

global\_index = 0

while temp is not None:

i = 0

while i < len(temp.array):

if global\_index == index:

temp.array.pop(i)

temp.length -= 1

if len(temp.array) < self.n\_array // 2:

if prev\_temp is not None and len(prev\_temp.array) + len(temp.array) <= self.n\_array:

prev\_temp.array.extend(temp.array)

prev\_temp.next = temp.next

elif temp.next is not None and len(temp.array) + len(temp.next.array) <= self.n\_array:

temp.array.extend(temp.next.array)

temp.next = temp.next.next

self.all\_length -= 1

return

i += 1

global\_index += 1

prev\_temp = temp

temp = temp.next

if self.head is not None and len(self.head.array) == 0:

self.head = self.head.next

def search(self, value):

temp = self.head

while temp.next != None:

for i in range(len(temp.array)):

if value == temp.array[i]:

return temp.array[i]

temp = temp.next

def search\_by\_index(self, index):

global\_index = 0

temp = self.head

while temp != None:

for i in range(len(temp.array)):

if global\_index == index:

return temp.array[i]

global\_index += 1

temp = temp.next

**Tests.py**

from main import unrolled\_linked\_list

from main import Node

def check(arr1, arr2):

my\_unrolled\_list = unrolled\_linked\_list()

for i in range(len(arr1)):

my\_unrolled\_list.append(arr1[i])

print(my\_unrolled\_list.print\_info())

for i in range(len(arr2)):

my\_unrolled\_list.delete(arr2[i])

print(my\_unrolled\_list.print\_info())

assert (my\_unrolled\_list.all\_length == 0)

def test\_assert(value=100):

my\_unrolled\_list = unrolled\_linked\_list()

for i in range(10):

my\_unrolled\_list.insert(value, i)

assert (my\_unrolled\_list.all\_length == 10)

def test\_search(len=10):

my\_unrolled\_list = unrolled\_linked\_list()

for i in range(len):

my\_unrolled\_list.append(i)

array = []

for i in range(my\_unrolled\_list.all\_length):

array.append(my\_unrolled\_list.search\_by\_index(i))

assert array == [x for x in range(len)]

def test\_print():

my\_unrolled\_list = unrolled\_linked\_list()

for i in range(10):

my\_unrolled\_list.append(i)

assert my\_unrolled\_list.print\_info() == ‘0 1 2 3 4 5 6 7 8 9 ’