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

Студентка гр. 2384 Валеева А.А.

Преподаватель Иванов Д.В.

Санкт-Петербург
2023

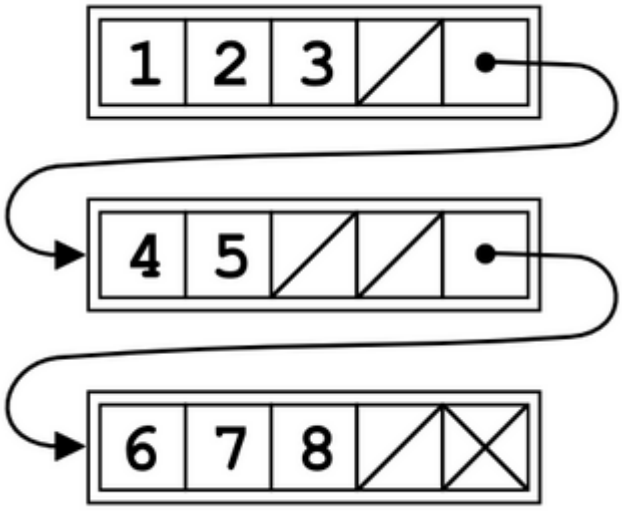
Цель работы

Изучить принцип работы с развернутыми связными списками. Реализовать развернутый связный список и необходимые методы для работы с ним, а также исследовать работу программы с помощью тестирования.

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. 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 также 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

Описание кода:

Файл *node.py*

Класс *Node* представляет узел в связанном списке. Он имеет три атрибута: *length*, который хранит длину массива, *array*, который хранит массив элементов в узле, и *next*, который указывает на следующий узел в списке.

Файл *unrolled_list.py*

Класс *Unrolled_list* представляет связанный список, в котором каждый узел содержит не один, а несколько элементов. Он имеет четыре атрибута: *tail*, который указывает на последний элемент в списке, *head*, который указывает на первый узел в списке, *len_list*, который хранит общее количество элементов в списке. А также конструктор класса принимает необязательный параметр *n_array*, который задает максимальное количество элементов в узле (по дефолту равен 4).

Функция *search_index* ищет элемент в связанном списке с массивами по заданному индексу. Она принимает один аргумент: индекс, который должен быть меньше длины списка. Она возвращает элемент, который находится в массиве нужного узла. Она выбрасывает исключение, если индекс выходит за границы списка.

Функция *seach_value* принимает один параметр: *value*, который является целым числом, которое нужно найти в списке. Она возвращает индекс первого встреченного искомого значения, если его нет в списке, то -1.

Функция *insert_to_tail* добавляет элемент в конец связанного списка. Она проверяет, есть ли место в последнем узле списка, и если нет, создает новый узел. Далее половину элементов из прошлого узла «перебрасывает» в

новый узел, куда впоследствии устанавливает элемент в конец. Если место в последнем массиве есть, то функция вставляет переданное значение в последнюю свободную ячейку массива и увеличивает количество элементов в узле и в списке в целом.

Функция *insert_to_head* добавляет переданный ей элемент в начало связанного списка. Если «голова» списка пустая, то вызывается ранее описываемая функция - *insert_to_tail.* Со вставкой поступаем также, как и в прошлой функции, если место в *head* нет, то при создании новой ноды следует не забыть связать через поле *next* новую ноду со следующей нодой после *head.*

Функция insert_to_index вставляет элемент в связанный список с массивами по заданному индексу. Она принимает два аргумента: индекс и элемент. Она проверяет, что индекс не превышает размер списка. Функция находит нужный узел и массив, сдвигает элементы и вставляет элемент. Также создает создает новый узел, если массив заполнен. Работает подобно предыдущим двум функциям. Удаляемый индекс находим по формуле = заданный индекс — *count*. Не забываем уменьшить длину всего списка и длину массива.

Функция *delete_index* удаляет все элементы, найденными по заданному индексу из связанного списка массивов. Индекс проверяется на корректность, далее используем дополнительную переменную *count,* к которой прибавляем длину массива в ноде, если в нем нет искомого индекса.

Функция *delete_value* удаляет все элементы, равные заданному числу, из связанного списка массивов. Работа функции: с помощью реализованных функцих *search_value* находится индекс удаляемого значение и через *delete_index* удаляем элемент, продолжаем искать, если переданное число встречается несколько раз.

Функция *delete_empty_arr* удаляет все пустые массивы, которые останутся после использования функций удаления.

Функция *print_list* выводит все элементы *Unrolled_lise.*

Функция *to_arr* добавляет все элементы из всех нод листа в один массив.

Файл *main.py*

В основном файле реализована функция *check(arr1, arr2, n_array = 4).* С помощью метода *insert_to_tail()* добавляем в развернутый список элементы из *arr1*. Далее, используя метод *delete_value(i),* удаляем значения, встречаемые в *arr2.* После каждого изменения списка выводим его при помощи метода *print_list.*

Файл *test_main.py*

В файле *tests.py* содержится 8 тестов (по одному для каждой функции) для проверки работы развернутого связного списка с помощью библиотеки *pytest.*

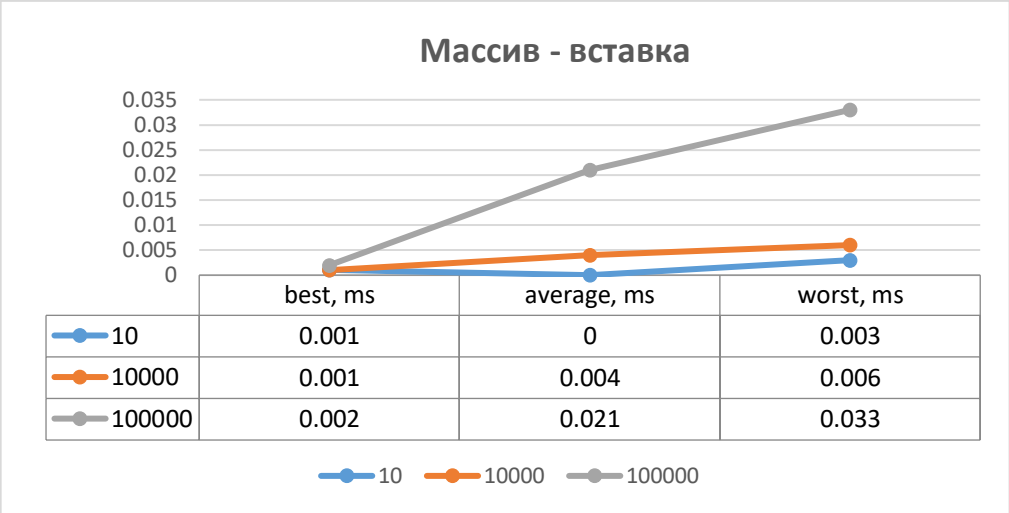
Исходный код программы представлен в приложении А.

Результаты тестирования представлены в приложении В.

Исследование сложности различных операций

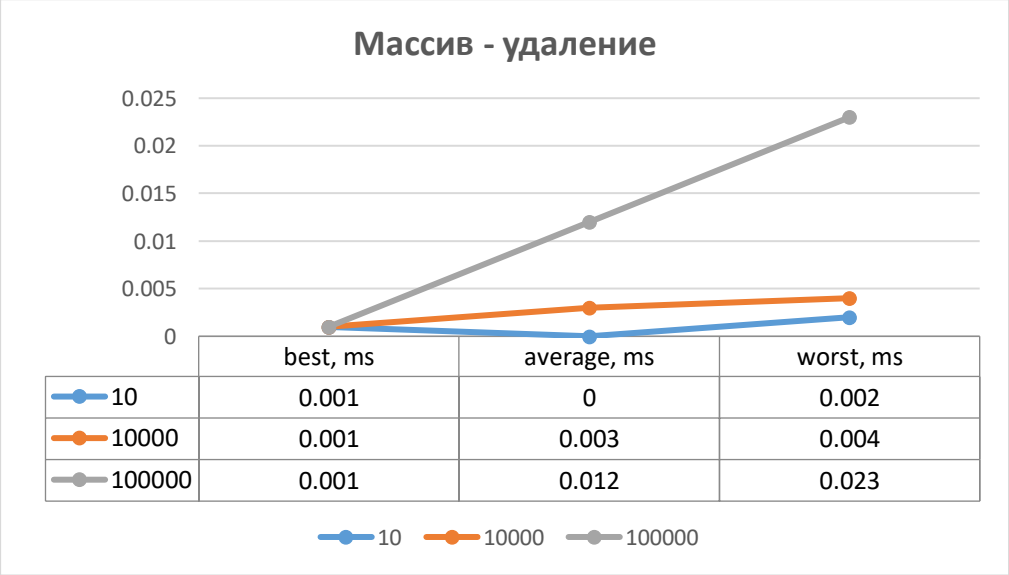
Далее были протестированы различные функции — вставка, удаление и поиск элемента в массиве, односвязном связном списке и в развернутом связном списке. Результаты представлены на рисунках 1-9.

Результаты тестирования:



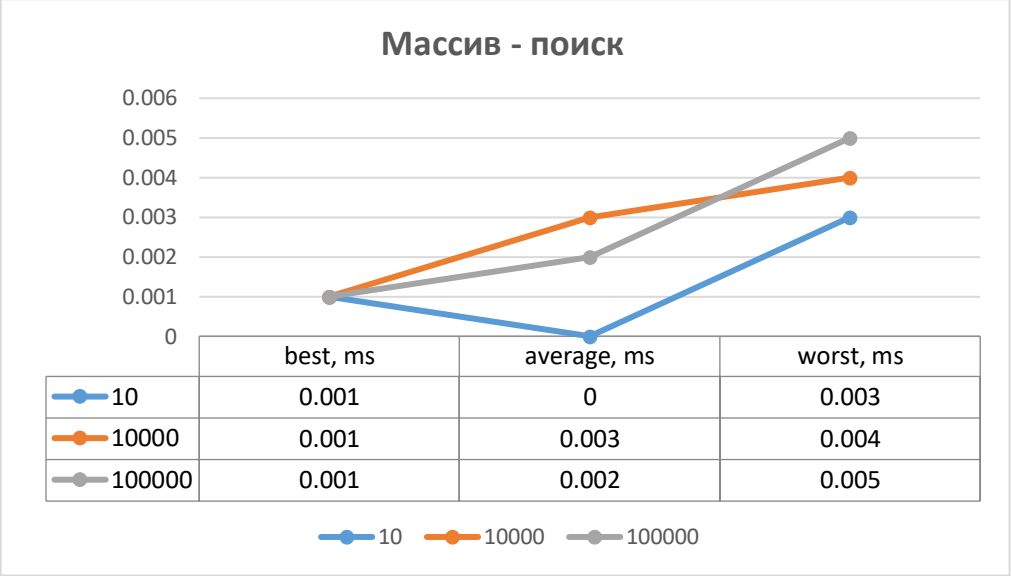
	best, ms	average, ms	worst, ms
10	0.001	0	0.003
10000	0.001	0.004	0.006
100000	0.002	0.021	0.033

Рисунок 1 — Результаты исследования вставки элемента в массив



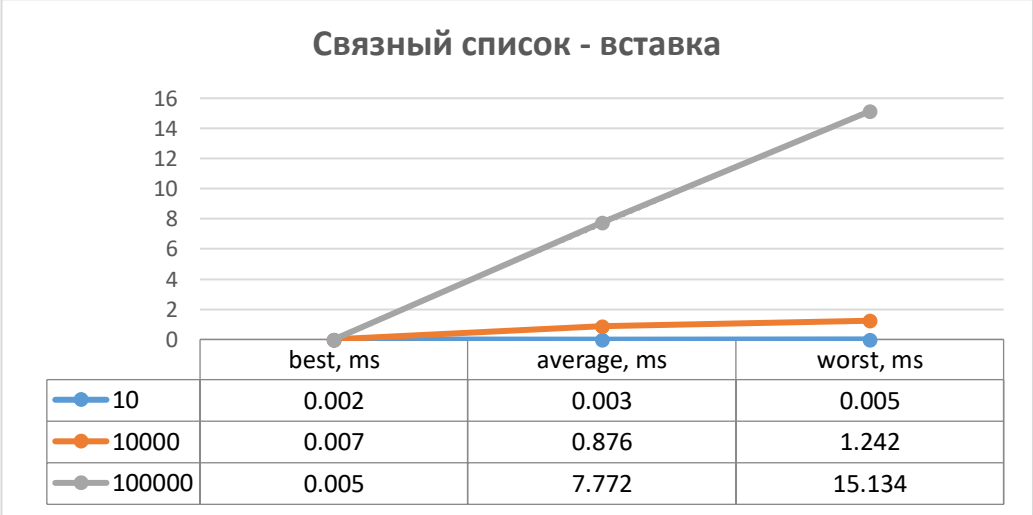
	best, ms	average, ms	worst, ms
10	0.001	0	0.002
10000	0.001	0.003	0.004
100000	0.001	0.012	0.023

Рисунок 2 — Результаты исследования удаления элемента из массива



	best, ms	average, ms	worst, ms
10	0.001	0	0.003
10000	0.001	0.003	0.004
100000	0.001	0.002	0.005

Рисунок 3 — Результаты исследования поиска элемента в массиве



	best, ms	average, ms	worst, ms
10	0.002	0.003	0.005
10000	0.007	0.876	1.242
100000	0.005	7.772	15.134

Рисунок 4 — Результаты исследования вставки элемента в связной список

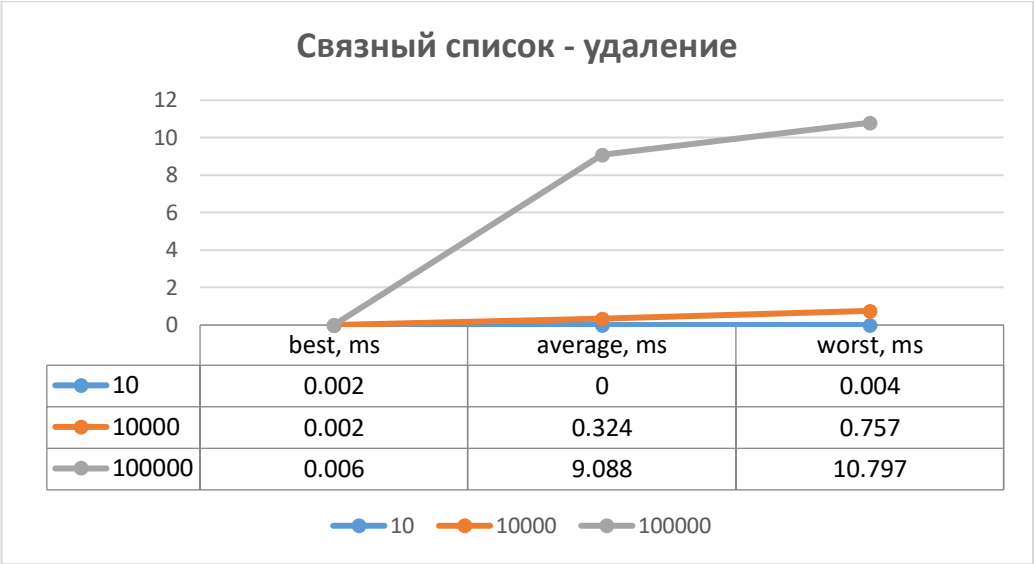


Рисунок 5 – Результаты исследования удаления элемента из связного списка

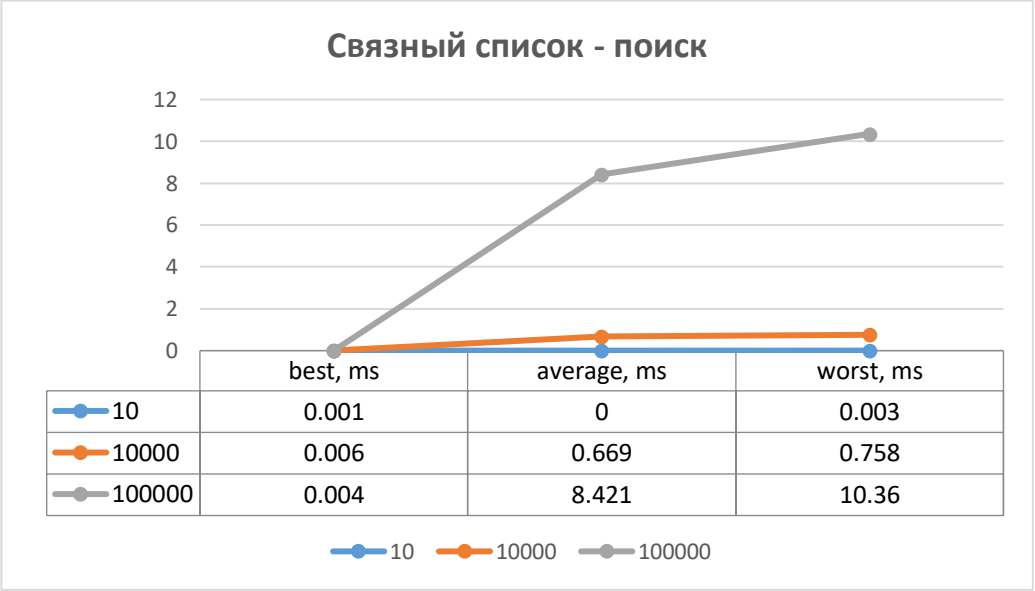
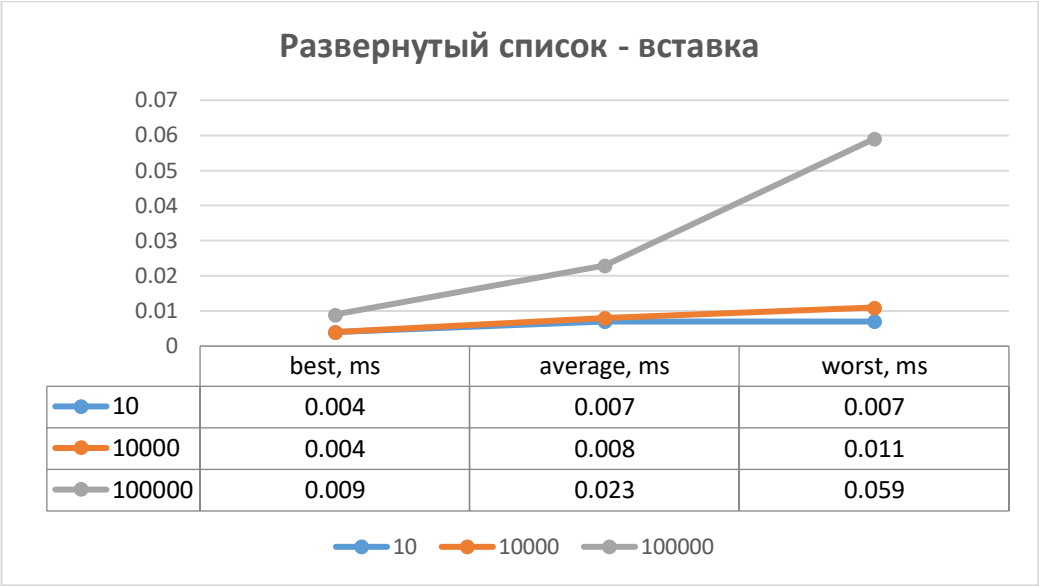
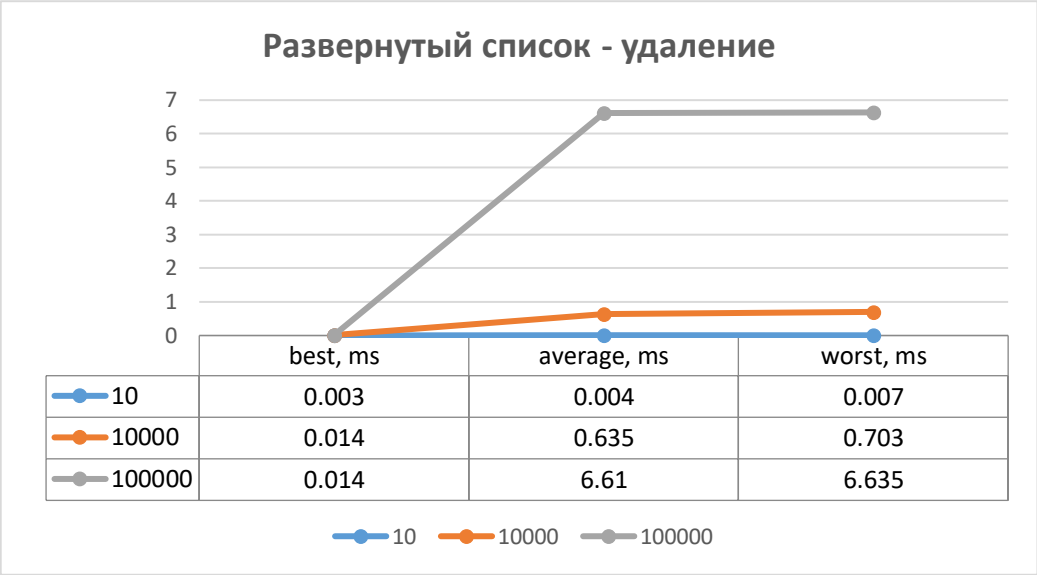


Рисунок 6 – Результаты исследования поиска элемента в связном списке



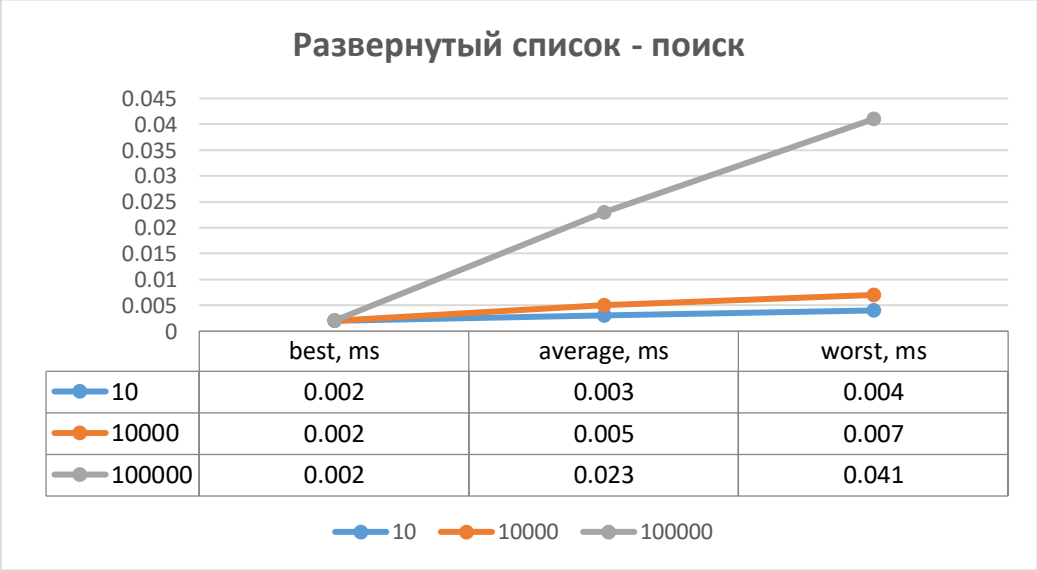
	best, ms	average, ms	worst, ms
10	0.004	0.007	0.007
10000	0.004	0.008	0.011
100000	0.009	0.023	0.059

Рисунок 7 – Результаты исследования вставки элемента в развернутый список



	best, ms	average, ms	worst, ms
10	0.003	0.004	0.007
10000	0.014	0.635	0.703
100000	0.014	6.61	6.635

Рисунок 8 – Результаты исследования удаления элемента из связного списка



	best, ms	average, ms	worst, ms
10	0.002	0.003	0.004
10000	0.002	0.005	0.007
100000	0.002	0.023	0.041

Рисунок 9 – Результаты исследования поиска элемента в развернутом списке

Выводы

Был проведен анализ работы развёрнутого связного списка, была разработана программа с функциями вставки, удаления и поиска элементов в нем. Были выполнены тесты и исследование, чтобы определить, какие преимущества и недостатки имеет такая структура данных.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.py

```
from unrolled_list import Unrolled_list
import time
def check(arr1, arr2, n_array = 4):
    my_unrolled_list = Unrolled_list(n_array)
    for i in arr1:
        my_unrolled_list.insert_to_tail(i)
        my_unrolled_list.print_list()
    for i in arr2:
        my_unrolled_list.delete_value(i)
        my_unrolled_list.print_list()
    my_unrolled_list.print_list()
if __name__ == '__main__':
    check([1, 2, 3], [3, 1], 4)
```

Название файла: test_main.py

```
from unrolled_list import Unrolled_list
import pytest

def test_len_list():
   my_list = Unrolled_list()
   my_list.insert_to_tail(3)
   my_list.insert_to_tail(4)
   my_list.insert_to_tail(8)
   my_list.insert_to_tail(9)

   answer_arr = [3, 4, 8, 5]
   assert len(answer_arr) == my_list.len_list

def test_delete_value():
   my_list = Unrolled_list()
   my_list.insert_to_tail(3)
```

```
    my_list.insert_to_tail(4)
    my_list.insert_to_tail(8)
    my_list.insert_to_tail(5)
    my_list.delete_value(4)
    answer_arr = [3, 4, 8, 5]
    answer_arr.remove(4)
    assert my_list.to_arr() == answer_arr


def test_delete_index():
    my_list = Unrolled_list()
    my_list.insert_to_tail(3)
    my_list.insert_to_tail(4)
    my_list.insert_to_tail(8)
    my_list.insert_to_tail(5)
    my_list.delete_index(1)
    answer_arr = [3, 4, 8, 5]
    answer_arr.pop(1)
    assert my_list.to_arr() == answer_arr

def test_search_value():
    my_list = Unrolled_list()
    my_list.insert_to_tail(3)
    my_list.insert_to_tail(4)
    my_list.insert_to_tail(8)
    my_list.insert_to_tail(5)
    answer_arr = [3, 4, 8, 5]
    answer_ind = answer_arr.index(8)
    assert my_list.search_value(8) == answer_ind

def test_search_index():
    my_list = Unrolled_list()
    my_list.insert_to_tail(3)
    my_list.insert_to_tail(4)
    my_list.insert_to_tail(8)
    my_list.insert_to_tail(5)
    answer_arr = [3, 4, 8, 5]
    answer_ind = answer_arr[2]
```

```
    assert my_list.search_index(2) == answer_ind

def test_insert_to_tail():
    my_list = Unrolled_list()
    my_list.insert_to_tail(3)
    answer_arr = [3, 4]
    assert my_list.to_arr() == answer_arr

def test_insert_to_head():
    my_list = Unrolled_list()
    my_list.insert_to_tail(3)
    my_list.insert_to_tail(4)
    my_list.insert_to_tail(8)
    my_list.insert_to_head(5)
    answer_arr = [5, 3,  4, 8]
    assert my_list.head.array == answer_arr

def test_insert_to_index():
    my_list = Unrolled_list()
    my_list.insert_to_tail(3)
    my_list.insert_to_tail(4)
    my_list.insert_to_tail(8)
    my_list.insert_to_index(5, 1)
    answer_arr = [3, 5,  4, 8]
    assert my_list.head.array == answer_arr
```

Название файла: node.py

```
class Node:
    def __init__(self):
        self.length_arr = 0
        self.array = []
        self.next = None
```

Название файла: unrolled_list.py

```
from node import Node
```

```
class Unrolled_list:
    def __init__(self, n_array = 4):
        self.n_array = n_array # максимальный размер массива(ноды списка)
        self.head = None
        self.tail = None
        self.len_list = 0 #общая длина всеэ элементов списка



    def search_index(self, index):
        if index >= self.len_list:
            print("Индекс выходит за границы спика")
        else:
            count = 0
            temp = self.head
            while temp != None:
                if (count + temp.length_arr > index) and (count <= index):
                    return temp.array[index-count]
                else:
                    count += temp.length_arr
                temp = temp.next


            print("Элемент под таким индексом не найден")
    def search_value(self, value):
        temp = self.head
        flag = False
        ind = 0
        while temp != None and ind < self.len_list:
            if value not in temp.array:
                ind += temp.length_arr
            else:
                for i in range(len(temp.array)):
                    if temp.array[i] == value:
                        ind += i
                        flag = 1
                break
            temp = temp.next
        if flag:
```

```
            return ind
        return -1


    def insert_to_tail(self, elem):
        if self.head == None:
            self.head = Node()
            self.head.array.append(elem)
            self.head.length_arr += 1
            self.tail = self.head


        # если еще есть место в последнем элементе
        elif self.tail.length_arr + 1 <= self.n_array: #если еще есть место в последнем элементе
            self.tail.array.append(elem) #вставляем туда элемент
            self.tail.length_arr += 1


        else:
            new_node = Node()
            half_length = self.tail.length_arr // 2
            new_node.array = self.tail.array[half_length:]
            new_node.array.append(elem)
            new_node.length_arr = len(new_node.array)


            self.tail.array = self.tail.array[0:half_length]
            self.tail.length_arr = half_length
            self.tail.next = new_node
            self.tail = new_node


        self.len_list += 1


    def insert_to_head(self, elem):
        if self.head == None:
            self.insert_to_tail(elem)
        elif (self.head.length_arr + 1 <= self.n_array):
            self.head.array.insert(0, elem)
            self.head.length_arr += 1
        else:
```

```
            prev = self.head.next

            new_node = Node()
            half_length = self.head.length_arr // 2
            new_node.array = self.head.array[half_length:]
            new_node.length_arr = len(new_node.array)
            self.head.array.insert(0, elem)
            self.head.array = self.head.array[0:half_length+1]
            self.head.length_arr = len(self.head.array)

            self.head.next = new_node
            new_node.next = prev

        self.len_list += 1

    def insert_to_index(self, value, index):
        if index < 0 or index > self.len_list:
            print("Введено неверное значение. Индекс выходит за границы списка")

        if self.len_list == 0 or self.len_list <= index:
            self.insert_to_tail(value)
        else:
            count = 0
            temp = self.head
            while temp is not None:
                for i in range(len(temp.array)):
                    if count == index:
                        if len(temp.array) + 1 <= self.n_array:
                            temp.array.insert(i, value)
                            temp.length_arr += 1
                        else:
                            new_node = Node()
                            half_ind = self.n_array // 2
                            new_node.array = temp.array[half_ind:]
                            new_node.length_arr = len(new_node.array)
                            temp.array = temp.array[:half_ind]
                            temp.length_arr = len(temp.array)    if (temp.next is not None):
```

```
                    new_node.next = temp.next
                    temp.next = new_node
                else:
                    temp.next = new_node

                if i < half_ind:
                    temp.array.insert(i, value)
                    temp.length_arr += 1
                else:
                    new_node.array.insert(i - half_ind, value)
                    new_node.length_arr += 1
                self.len_list += 1
                return
            count += 1
        temp = temp.next

def print_list(self):
    if self.head == None:
        print("Пустой список\n")
    temp = self.head
    while temp != None:
        for i in range(0, temp.length_arr):
            print(temp.array[i], end=" ")
        # print("end of array \n")

        temp = temp.next
    # print("длина всего списка = ", self.len_list)

def delete_index(self, index):
    if self.len_list == 0:
        return
    if index >= self.len_list or index < 0:
        print('Нет элемента с таким индексом')
    temp = self.head
    count = 0
    while index >= count + temp.length_arr and temp:
        count += temp.length_arr
```

```
            temp = temp.next
        ind = index - count
        del temp.array[ind]
        temp.length_arr -= 1
        self.len_list -= 1



    def delete_value(self, value):
        if self.len_list == 0:
            return
        ind = self.search_value(value)
        while ind >= 0:
            self.delete_index(ind)
            ind = self.search_value(value)


    def delete_empty_arr(self):
        temp = self.head
        while temp.next != None:
            if (temp.next.length_arr == 0):
                if (temp.next.next != None):
                    temp.next = temp.next.next
                    temp = temp.next
                else:
                    temp.next = None
    def to_arr(self):
        temp = self.head
        arr_result = []
        while temp:
            arr_result.extend(temp.array)
            temp = temp.next
        return arr_result
```

ПРИЛОЖЕНИЕ В

ИСХОДНЫЙ КОД ПРОГРАММЫ

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии

1.	alina=Unrolled_list() arr1 = [1, 2, 3, 4, 5, 6] arr2 = [2, 3, 8]	1 4 5 6	Заполнили список значениями из первого массива и удалили значения, равные значениям из второго списка
2.	alina.insert_to_tail(4) alina.insert_to_tail(5) alina.insert_to_tail(6)	[4, 5, 6]	Заполнение ноды значениями
3.	alina.insert_to_head(7)	[7, 4, 5, 6]	Вставили значение в начало ноды
4.	alina.seach_to_value(5)	2	Поиск по значению
5.	alina.seach_to_index(3)	6	Поиск по индексу
6.	alina.delete_to_value(4)	[7, 5, 6]	Удаление по значению