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. 3342 _____ Иванов Д. М.

Преподаватель _____ Иванов Д. В.

Санкт-Петербург
2024

Цель работы

Изучить такую структуру данных, как развернутый список. С помощью языка программирования Python реализовать класс по работе с этой структурой. Протестировать список и оценить эффективность его работы.

Задание

В рамках данной лабораторной работы необходимо реализовать “новую” структуру, которая называется развернутым связным списком.

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.

Выполнение работы

1) Создание класса Node

Прежде всего необходимо создать класс, из объектов которого будет состоять наш список. Класс Node будет иметь следующие поля:

self.arr – массив чисел

self.next – ссылка на следующий объекта класса Node. Изначально наш элемент никуда не ссылается, поэтому присваиваем None.

Методы:

__init__(self, array=[]) – конструктор класса. Необязательным аргументов является передача массива чисел, который присваивается полю self.arr.

2) Создание класса UnrolledLinkedList

Поля:

self.head – первый элемент списка, который относится к классу Node

self.length – количество связанных между собой массивов

self.len_of_node_array – фиксированный размер одного узла. Он расчитывается в функции calculate_optimal_node_size(num_elements) по специальной формуле(файл Calculate_size.py, в котором находится формула, приводится в приложении). Внутри списка массивы не могут иметь длину больше этого значения. В противном случае придется в дальнейшем данные разбивать на несколько узлов.

Методы:

__init__(self, arr=[]) – конструктор класса. Устанавливает для полей нужные значения: self.head = None, self.lenght = 0. Запускает метод make_linked_list(описан ниже).

make_linked_list(self, arr) – метод, который по переданному при инициализации массиву формирует развернутый связаный список с фиксированной длиной. Через цикл идет заполнение структуры и балансировка узлов при их заполнении. До тех пор пока не будут взяты все числа массива.

push(self, element) – вставка переданного числа в конец списка. Через поле next переходим к последнему узлу списка, который ссылается на None. И в сам массив добавляется число. Если происходит перебор чисел, то массив разбивается на 2 части, и вторая часть переходит в новый созданный узел списка.

insert(self, element, index) – вставка переданного числа по индексу. В первую очередь через этот индекс находится индекс самого узла, внутрь которого нужно вставить число. После вставик в массив, по аналогии с методом push, присходит проверка на размер получившегося массива. И в случае необходимости разбиение узла на две части и возможное соединение со следующим элементов списка.

find_node_by_index_of_element(self, index) – метод для нахождения узла списка через переданный индекс относительно всех чисел списка и его индекса относительно остальных узлов. Создается счеткчик, в который добавляется длина массива каждого нового узла, параллельно через поле next происходит переход к следующему узлу. Процесс идет до момента, пока счетчик меньше переданного в функцию аргумента.

delete_number(self, index) – удаление числа из списка по переденному индексу. В начале при помощи функции find_node_by_index_of_element, описанной выше, находится сам узел, его индекс и индекс первого элемента относительно всего списка. Через эти данные через метод del удаляется число из массива. Идет проверка: если массив получился пустым, весь узел удалается через метод delete_arr(будет описана дальше), и если возможно данный массив связать с массивом следующего узла, то происходит объединение.

delete_arr(self, index) – метод для удаление узла через переданный индекс узла. С помощью цикла программа доходит до нужного узла. Затем у предыдущего узла меняется поле next, после чего он ссылается на другой элемент списка.

find_index_of_number(self, number) – нахождение индкса переданного числа в списке. Идет обход, начиная с head, всех узлов списка, пока в данном

массиве не будет найдено нужное нам число. После нахождения суммируем все найденные ранее длины массивов и возвращаем полученный индекс.

print_list(self) – вывод списка. В каждой строке программа выводит индекс узла и его массив чисел.

3) Проверка работоспособности программы

В первую очередь напишем файл с тестами для программы test.py(код приводится в приложении). В этом файле будут проверяться такие методы, как поиск индекса числа, вставка и удаление в разных частях списка(начало, середина, конец). Для вставки и удаления будет проверяться, что при верно введенных индексах метод выполняет работу и возвращает True. Однако при неверных индексах не происходит аварийное завершение программы, а метод просто возвращает False.

Дальше пойдет проверка правильности выполнения тех же методов в файле main.py(код приводится в приложении). Функция check принимает два массива чисел: arr1 – для заполнения, arr2 – для поиска и удаления. 50 чисел добавляются постепенно через push. Еще 3 элемента подаются в разные части списка через insert. После этого находится индекс каждого элемента из arr2 и удаляется

Анализ полученных значений

В конце необходимо оценить эффективность данной структуры, сравнив её с односвязанным списком и массивом (list() в Python). Для замерки времени работы использовалась библиотека time. Оценивались такие методы, как нахождение индексов элементов, вставка и удаление чисел в разных частях структур. Проверились операции на маленьких (1000), средних (10000) и больших (100000) наборах данных. Результаты представлены в виде графиков (см. ри. 1).

По результатм ее исследования и сравнения с другими структурами данных (см. рис. 1) можно сделать следующие выводы.

На малых объемах данных разница в большинстве случаев незаметна. Однако с увеличением размеров структур различие по времени выполнения операций становится более существенным.

Вставка и удаление элементов в середину и конец выполняется намного быстрее, чем для связанного списка, и практически одинаково с массивом. Такая же ситуация с нахождением элементов в тех же местах данных. При вставке и удалении в начало структуры развернутый список работает медленнее, чем односвязанный, но быстрее, чем массив. Единственное место, где проигрывает наша структура – это нахождение элемента в начале.

В плане потребления памяти развернутый связный список и массив потребляют меньше ресурсов по сравнению с односвязным списком, так как односвязный список требует дополнительной памяти для хранения указателей на следующие элементы. В нашей структуре данных это используется более эффективно.

Таким образом, развернутый связанных список показывает хорошую эффективность по времени выполнения операция и использования памяти по сравнению с другими структурами данных. Это делает его более привлекательным при работе большим объемом данных.

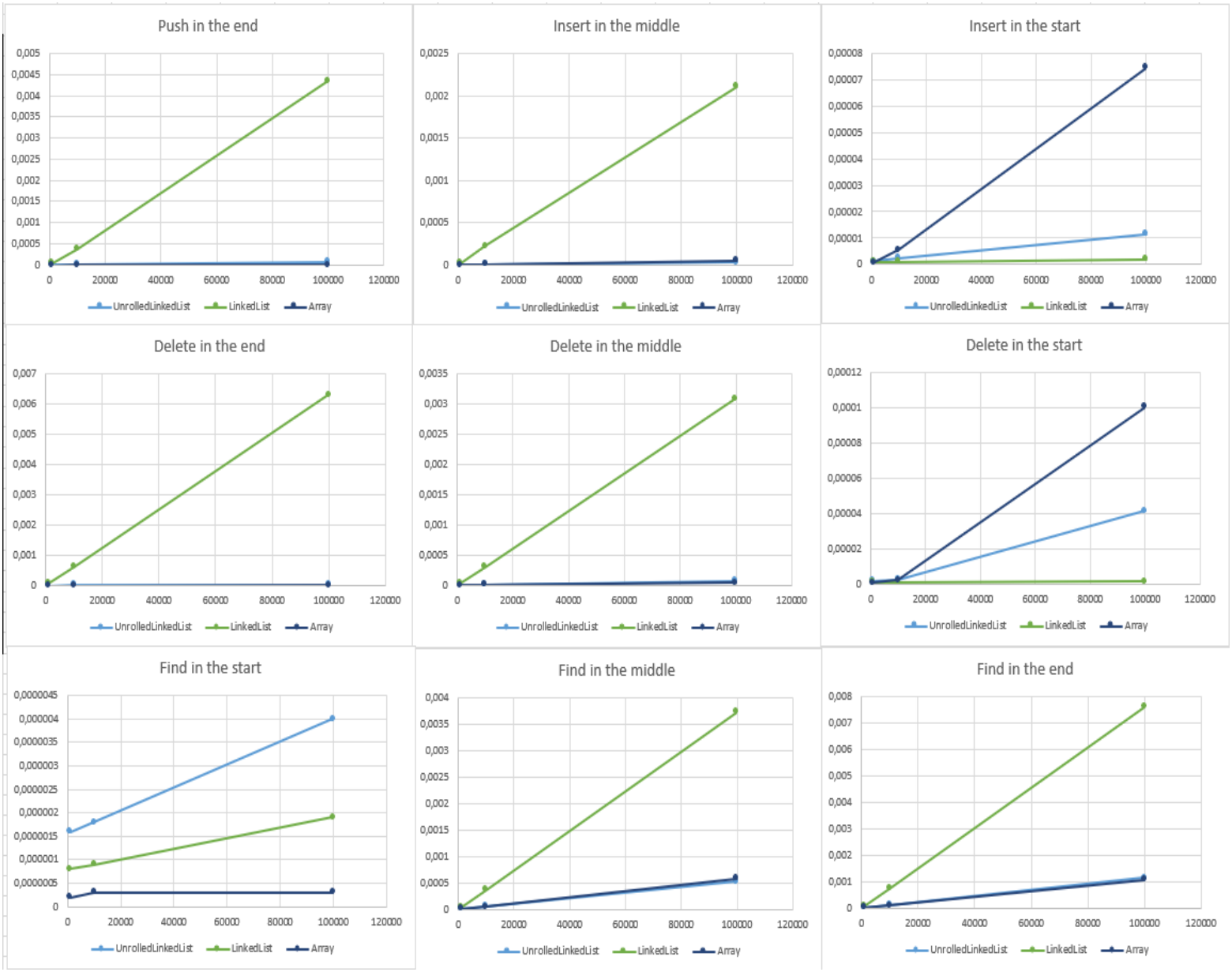

Рисунок 1 – графики, показывающие время выполнения операций для различных структур

Выводы

Была реализована такая структура данных, как развернутый связанный список. На языке Python написаны основные методы для нее. Реализация включала такие основные операции, как добавление, удаление и поиск. Также структура была проверена на работоспособность и эффективность по времени относительно массива и односвязанного списка.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: UnrolledLinkedList.py

```
"""Module include structure unrolled linked list"""

import dataclasses
from Calculate_size import calculate_optimal_node_size


@dataclasses.dataclass
class Node:
    """Class of element of list"""

    def __init__(self, array=None):
        if array is None:
            array = []
        self.arr = array.copy()
        self.next = None


class UnrolledLinkedList:
    """Class of structure unrolled linked list"""

    def __init__(self, arr=None):
        if arr is None:
            arr = []
        self.head = None
        self.length = 0
        self.len_of_node_array = calculate_optimal_node_size(len(arr))

        self.make_linked_list(arr)

    def make_linked_list(self, arr):
        """Function of make linked list with initialization"""
        if len(arr) == 0:
            return
        self.length = 0
        k = 0
        lis = []
        index = 0
        flag_head = False
        el = self.head
        while index < len(arr):
            if k < self.len_of_node_array:
                lis.append(arr[index])
                k += 1
                index += 1
            if k >= self.len_of_node_array:
                if not flag_head:
                    self.head = Node(lis)
                    self.length += 1
                    flag_head = True
                    el = self.head
                else:
                    el.next = Node(lis)
                    self.length += 1
```

```
                    el = el.next
                lis = []
                k = len(lis)
        if len(lis) != 0:
            if not flag_head:
                if len(lis) > self.len_of_node_array:
                    self.head = Node(lis[:self.len_of_node_array // 2])
                    self.head.next = Node(lis[self.len_of_node_array //
2:])
                    return
                self.head = Node(lis)
                return
            if   el   is   None   and   len(el.arr)   +   len(lis)   <=
self.len_of_node_array:
                el.arr += lis
            else:
                el.next = Node(lis)

    def push(self, element):
        """Function of push new element to the end"""
        if self.head is None:
            self.head = Node([element])
            return True
        el = self.head
        while el.next is not None:
            el = el.next
        if len(el.arr) + 1 > self.len_of_node_array:
            massive = el.arr
            el.arr = massive[:self.len_of_node_array // 2]
            el.next  =  Node(massive[self.len_of_node_array  //  2:]  +
[element])
            self.length += 1
            return True

        el.arr.append(element)
        return True

    def insert(self, element, index):
        """Function of insert element with some index"""
        if index < 0:
            return False
        el = self.head
        start_index = len(el.arr)
        while start_index <= index:
            el = el.next
            if el is None:
                return False
            start_index += len(el.arr)
        if el is None:
            return False

        el.arr.insert(index - start_index, element)

        if len(el.arr) <= self.len_of_node_array:
            return True

        half_length = self.len_of_node_array // 2
        new_array = el.arr[half_length:]
```

```
        el.arr = el.arr[:half_length]

        tmp = el.next
        el.next = Node(new_array)
        el.next.next = tmp
        self.length += 1

        return True

    def delete_number(self, index):
        """Function of delete element with some index"""
        el,            start_index,            index_node            =
self.find_node_by_index_of_element(index)
        if el is None:
            return False
        index_node_next = index_node + 1
        del el.arr[index - start_index]
        if len(el.arr) == 0:
            self.delete_arr(index_node)
            return True
        if el.next is not None:
            if len(el.next.arr) + len(el.arr) <= self.len_of_node_array:
                el.arr += el.next.arr
                self.delete_arr(index_node_next)
        return True

    def find_node_by_index_of_element(self, index):
        """Function of find node with some index"""
        index_node = 0
        if index < 0:
            return None, -1, -1
        el = self.head
        k = len(el.arr)
        while k <= index:
            el = el.next
            index_node += 1
            if el is None:
                return None, -1, -1
            k += len(el.arr)
        return (el, k - len(el.arr), index_node)

    def delete_arr(self, index):
        """Function of delete some array from the list"""
        if index == 0:
            self.head = self.head.next
            self.length -= 1
            return

        el = self.head
        k = 0
        while k < index - 1 and el.next is not None:
            el = el.next
            k += 1

        el.next = el.next.next
        self.length -= 1

    def find_index_of_number(self, number):
```

```
        """Function of find number index in all list"""
        k = 0
        index = 0
        el = self.head
        while el is not None:
            if number in el.arr:
                return index + el.arr.index(number)
            index += len(el.arr)
            el = el.next
            k += 1
        return None

    def print_list(self):
        """Function printing list"""
        k = 0
        el = self.head
        while el is not None:
            string = ' '.join([str(x) for x in el.arr])
            print(f"Node {k}: {string}")
            el = el.next
            k += 1
```

Название файла: test.py

```
"""Module include test of unrolled linked list"""


from UnrolledLinkedList import UnrolledLinkedList


def make_list():
    """function of make list object"""
    lis = list(range(1, 40))
    inrolled_lis = UnrolledLinkedList(lis)
    return inrolled_lis


def test_find_start():
    """test_1"""
    lis = make_list()
    assert lis.find_index_of_number(1) == 0


def test_find_middle():
    """test_2"""
    lis = make_list()
    assert lis.find_index_of_number(14) == 13


def test_find_end():
    """test_3"""
    lis = make_list()
    assert lis.find_index_of_number(39) == 38


def test_delete_number_with_right_index_in_the_end():
    """test_4"""
    lis = make_list()
    assert lis.delete_number(38) is True
```

```
def test_delete_number_with_right_index_in_the_middle():
    """test_5"""
    lis = make_list()
    assert lis.delete_number(14) is True


def test_delete_number_with_right_index_in_the_start():
    """test_6"""
    lis = make_list()
    assert lis.delete_number(0) is True


def test_delete_number_with_wrong_high_index():
    """test_7"""
    lis = make_list()
    assert lis.delete_number(55) is False


def test_delete_number_with_wrong_low_index():
    """test_8"""
    lis = make_list()
    assert lis.delete_number(-2) is False


def test_add_numbers_with_right_index_int_the_middle():
    """test_9"""
    lis = make_list()
    assert lis.insert(99, 3) is True


def test_add_numbers_with_right_index_int_the_start():
    """test_10"""
    lis = make_list()
    assert lis.insert(66, 0) is True


def test_add_numbers_with_right_index_int_the_end():
    """test_11"""
    lis = make_list()
    assert lis.push(77) is True


def test_add_numbers_with_wrong_low_index():
    """test_12"""
    lis = make_list()
    assert lis.insert(123, -3) is False


def test_add_numbers_with_wrong_high_index():
    """test_13"""
    lis = make_list()
    assert lis.insert(70, 61) is False
```

Название файла: main.py

```
"""Module include check of workable unrolled linked list"""
from UnrolledLinkedList import UnrolledLinkedList
```

```
def check(arr1, arr2, n_array=None):
    """function of check work of unrolled linked list"""
    lis = UnrolledLinkedList(arr1[:47])
    if n_array is not None:
        lis.len_of_node_array = n_array
    print("List node size:", lis.len_of_node_array)
    lis.print_list()
    print('------------')

    for i in arr1[47:50]:
        lis.push(i)

    print("Push elements")
    lis.print_list()
    print('------------')

    list_for_insert = [23, 49, 0]
    for i in range(3):
        lis.insert(arr1[50+i], list_for_insert[i])
        print(f"Insert {arr1[50+i]} to index {list_for_insert[i]}")
        lis.print_list()
        print('------------')

    for i in arr2:
        index = lis.find_index_of_number(i)
        lis.delete_number(index)
        print(f"Found {i} with index {index} and deleted it")
        lis.print_list()
        print('------------')


arr_with_start_elements = [54, 22, 60, 68, 31, 5, 17, 37, 32, 46, 49, 6,
62, 77, 16, 19, 20, 14, 3, 56, 56, 11, 58, 77, 95, 44, 11, 21,
        32, 88, 58, 23, 30, 41, 89, 48, 89, 29, 76, 45, 37, 99, 97, 17,
46, 59, 91, 21, 61, 36, 101, 102, 103]
arr_with_elements_for_finding = [68, 21, 54, 36]
check(arr_with_start_elements, arr_with_elements_for_finding)
```

Название файла: Calculate_size.py

```
"""Module include function of find node size"""

from math import ceil


def calculate_optimal_node_size(num_elements):
    """function of calculate node size"""
    memory = num_elements * 4
    return ceil(memory / 64) + 1
```