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 _____ Песчатский С. Д.

Преподаватель _____ Иванов Д.В.

Санкт-Петербург
2024

Цель работы

Реализация структуры данных «развернутый связный список» на языке программирования Python и исследование ее эффективности по сравнению с массивом и односвязным списком.

Задание

Развёрнутый связный список — список, каждый физический элемент которого содержит несколько логических элементов (обычно в виде массива, что позволяет ускорить доступ к отдельным элементам). 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 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

Класс LinkedList представляет собой расширенный связный список, в котором каждый узел содержит несколько элементов в виде массива. Это позволяет сократить количество узлов, что ускоряет доступ к данным и оптимизирует использование памяти по сравнению с традиционным связным списком. Основная концепция заключается в том, что элементы разбиваются на блоки фиксированного размера, где каждый узел хранит несколько логических элементов, что уменьшает количество ссылок и операций при обращении к данным. Размер узла определяется с помощью функции «calculate_optimal_size_node».

Основные операции, реализованные в классе:

Конструктор класса («__init__»): принимает на вход массив данных для заполнения, а также максимальное количество элементов в одном узле. Инициализирует первый узел и создает связный список. Массив данных и количество элементов – необязательные параметры, при создании класса, можно не передавать туда данные сразу.

Метод «IndexSearch»: находит узел, содержащий элемент с заданным линейным индексом, и возвращает значение элемента.

Метод «ValSearch»: ищет элемент в списке и возвращает индекс узла и элемента в узле

Метод «IndexRemove»: удаляет элемент по заданному индексу узла и элемента.

Метод «ValRemove»: удаляет первое вхождение элемента.

Метод «insert»: вставляет элемент в список по указанному линейному индексу.

Метод «rebalance»: балансирует все узлы, удаляет пустые и создает новый если находит переполнение.

Метод «push»: вставляет элемент в конец списка.

Метод «showlist»: выводит весь список по узлам.

Метод «__len__»: возвращает общее количество элементов в списке.

Разработанный программный код см. в приложении А.

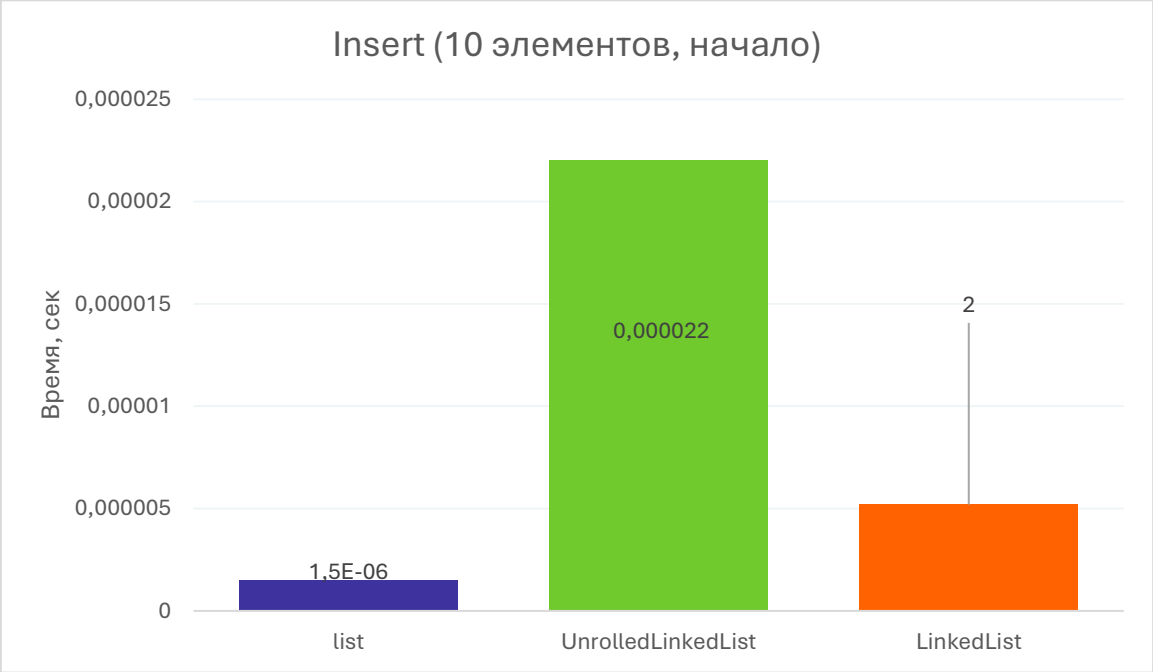
Тестирование

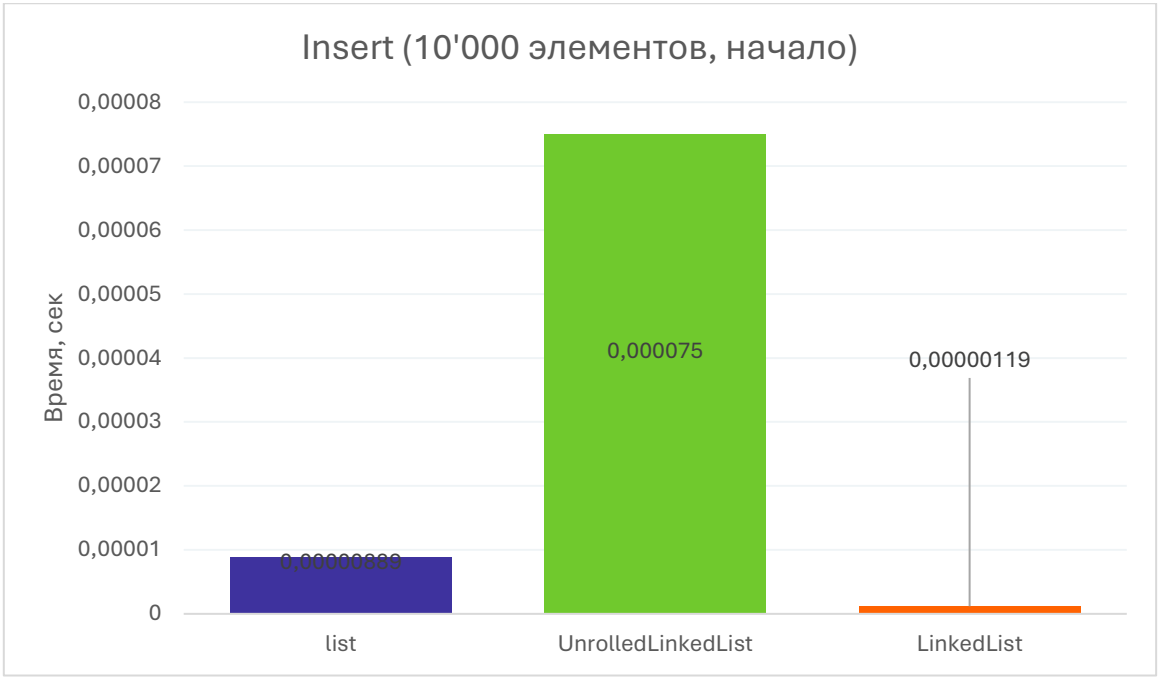
Тесты для проверки корректности работы реализованного развернутого связного списка находятся в файле tests.py. Каждый тест покрывает основные

операции, такие как вставка, удаление, поиск, а также их корректную работу на «граничных» случаях.

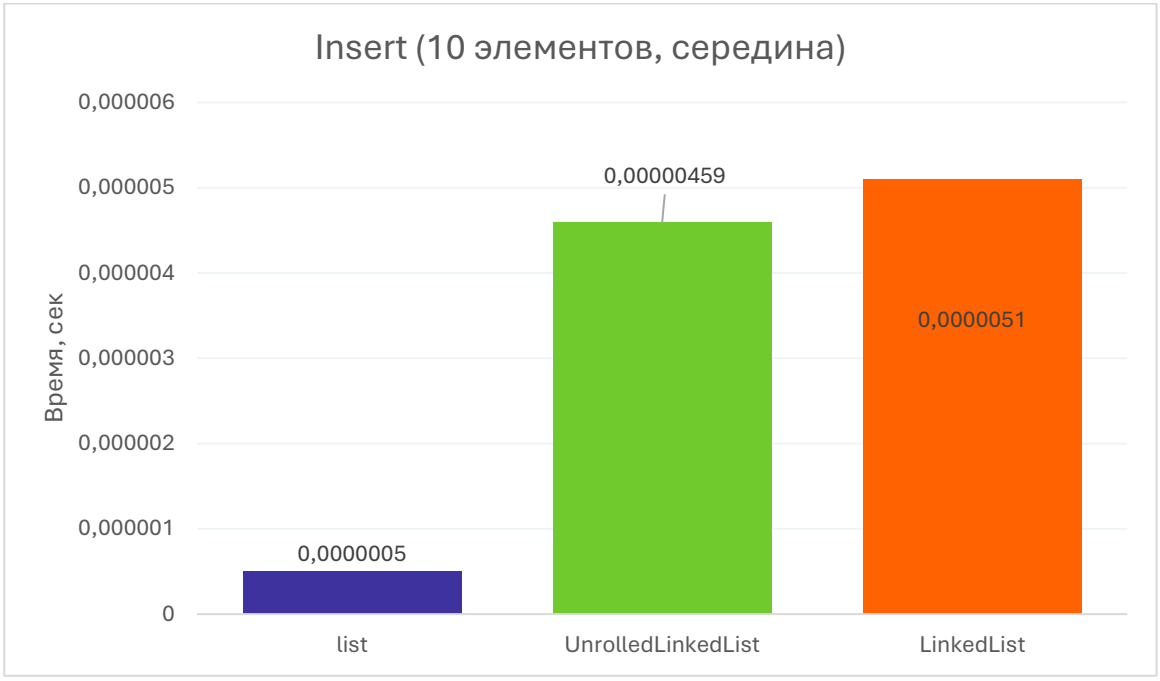
Исследование

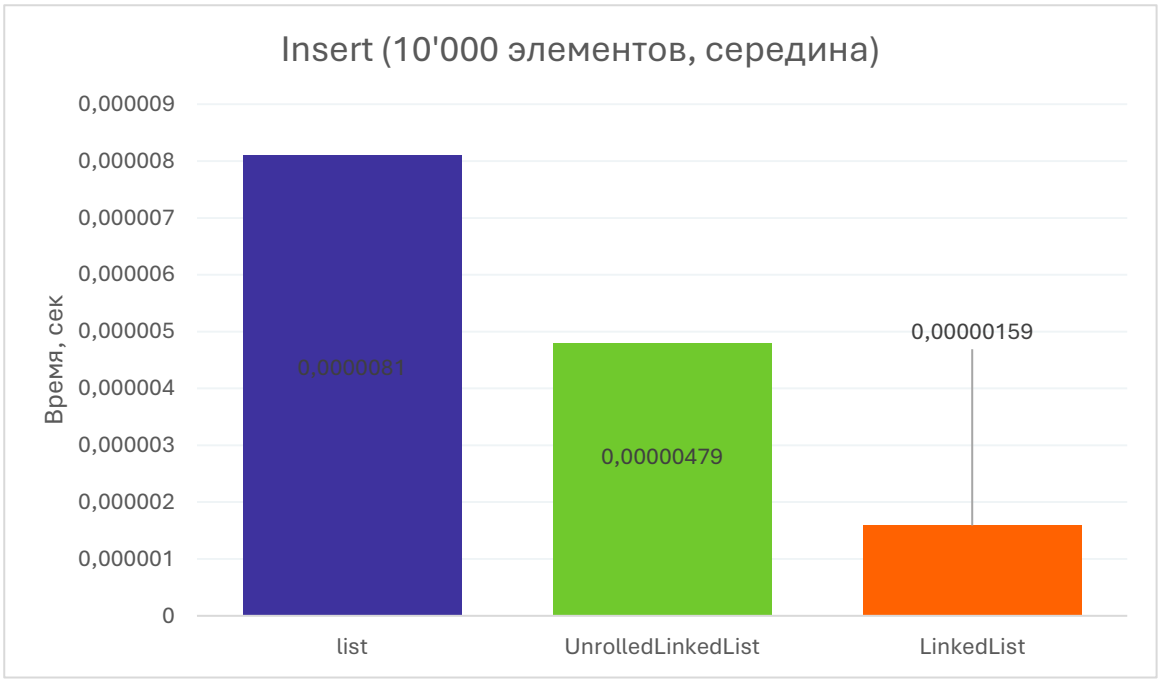
Ниже представлены сравнительные графики и таблицы трех структур данных – стандартного списка list в Python, односвязного списка LinkedList и развернутого связного списка LinkedList. Тестирование проводилось для операций вставки в начало/середину/конец каждой из структур данных.

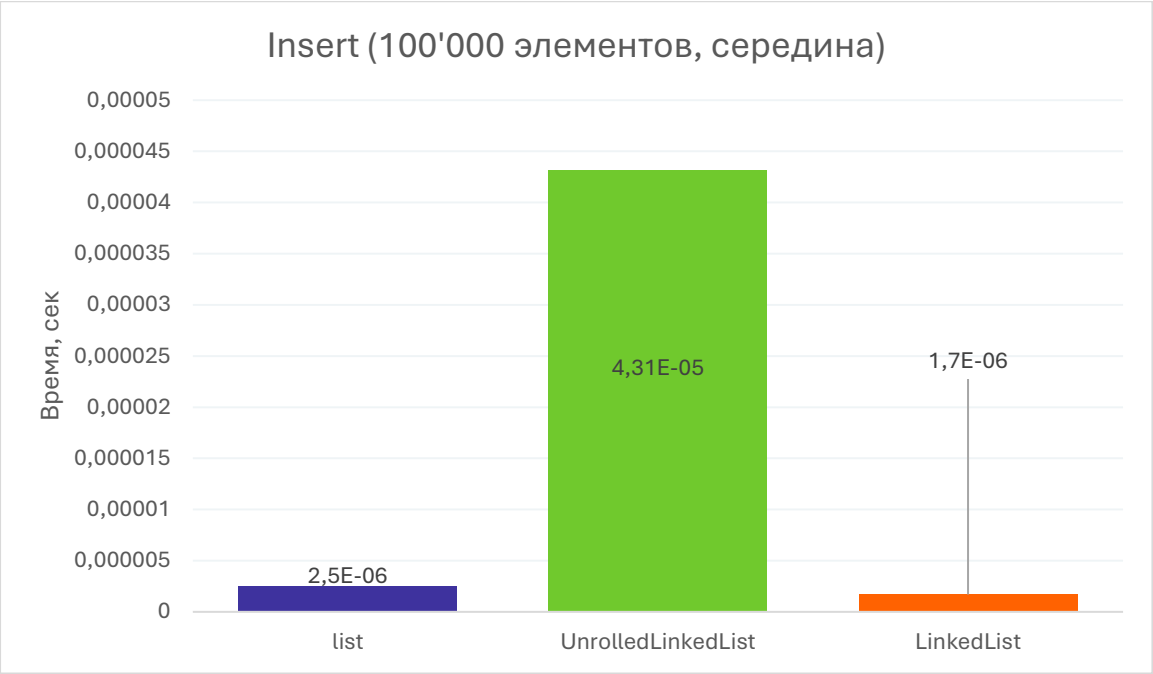



Insert (10'000 элементов, начало)
Время, сек
0,00008
0,00007
0,00006
0,00005
0,00004
0,00003
0,00002
0,00001
0
0,000075
0,00000119
list
UnrolledLinkedList
LinkedList

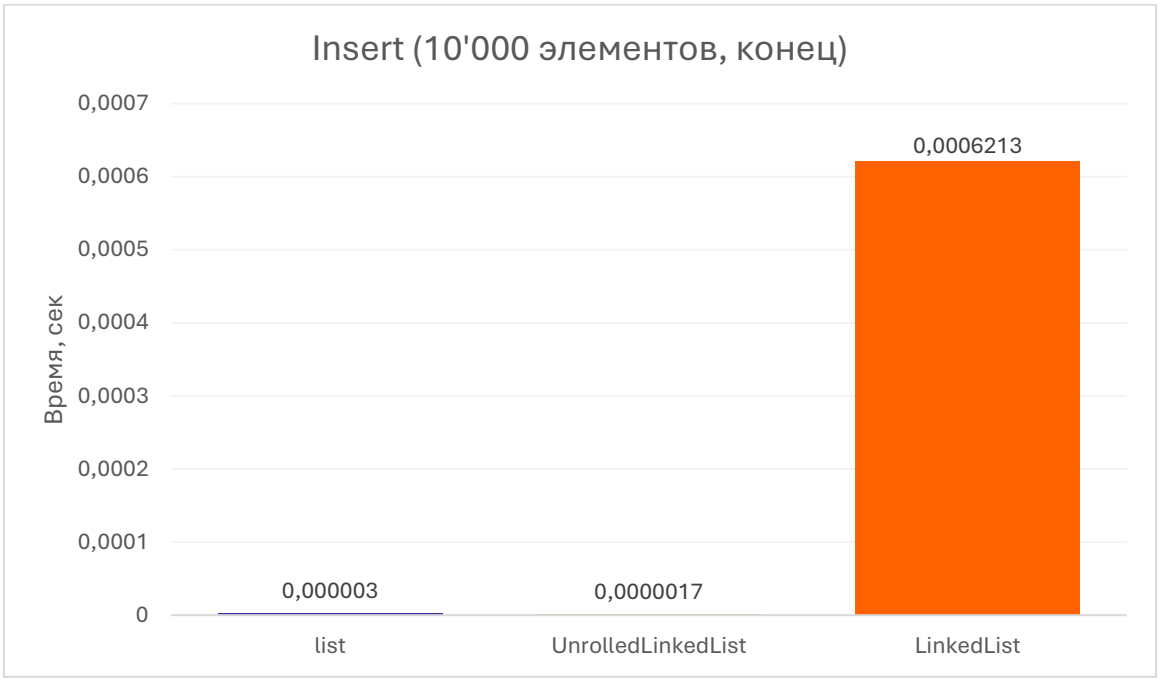

Insert (100'000 элементов, начало)
Время, сек
0,0012
0,001
0,0008
0,0006
0,0004
0,0002
0
0,0000813
0,001038
0,0000038
list
UnrolledLinkedList
LinkedList

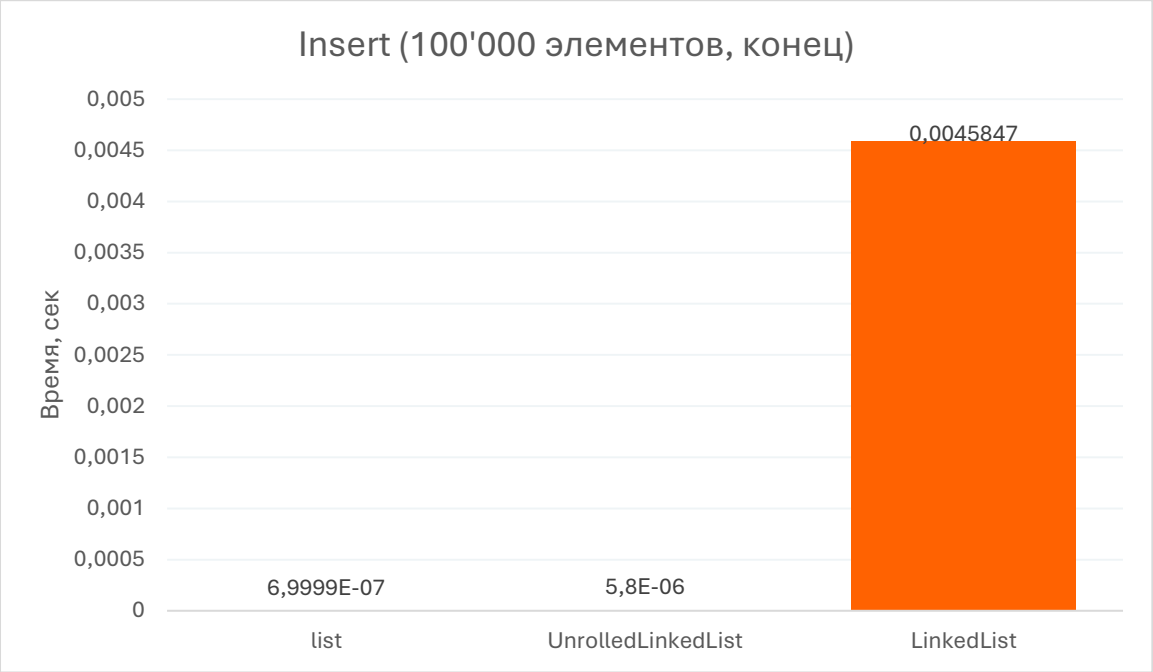

Insert (10 элементов, середина)
Время, сек
0,000006
0,000005
0,000004
0,000003
0,000002
0,000001
0
0,0000005
0,00000459
0,0000051
list
UnrolledLinkedList
LinkedList


Insert (10'000 элементов, середина)
Время, сек
0,000009
0,000008
0,000007
0,000006
0,000005
0,000004
0,000003
0,000002
0,000001
0
0,0000081
0,00000479
0,00000159
list
UnrolledLinkedList
LinkedList


Insert (100'000 элементов, середина)
Время, сек
0,00005
0,000045
0,00004
0,000035
0,00003
0,000025
0,00002
0,000015
0,00001
0,000005
0
2,5E-06
4,31E-05
1,7E-06
list
UnrolledLinkedList
LinkedList


Insert (10 элементов, конец)
Время, сек
0,000005
0,0000045
0,000004
0,0000035
0,000003
0,0000025
0,000002
0,0000015
0,000001
0,0000005
0
0,0000003
0,0000047
0,00000269
list
UnrolledLinkedList
LinkedList


Insert (10'000 элементов, конец)
Время, сек
0,0007
0,0006
0,0005
0,0004
0,0003
0,0002
0,0001
0
0,000003
0,0000017
0,0006213
list
UnrolledLinkedList
LinkedList


Insert (100'000 элементов, конец)
Время, сек
0,005
0,0045
0,004
0,0035
0,003
0,0025
0,002
0,0015
0,001
0,0005
0
0,0045847
6,9999E-07
5,8E-06
list
UnrolledLinkedList
LinkedList

Выводы

В ходе исследования была реализована и проанализирована структура данных — развернутый связный список. Основное внимание было уделено исследованию временных затрат на операцию вставки элементов.

Проанализировав тесты, видно, что развернутый связный список показал себя хуже по скорости выполнения задач, это может быть связано с неоптимизированным методом балансировки узлов.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.py

```
from UnrolledLinkedList import UnrolledLinkedList
from CalculateNodeSize import calculate_optimal_node_size
def check(arr1, arr2, nSize=None):
    if nSize==None:
        nSize=calculate_optimal_node_size(len(arr1))
    List=UnrolledLinkedList(n_size=nSize)
    for i in arr1:
        List.push(i)
        List.showList()
    for i in arr2:
        List.valRemove(i)
        List.showList()
check([i for i in range(10000)], [1,2,20,25,40,100,30,15,3,0,2,5], calculate_optimal_node_size(10000))
```

Название файла: CalculateNodeSize.py

```
import math
def calculate_optimal_node_size(num_elements):
    return math.ceil(num_elements/16)+1
```

Название файла: Node.py

```
class Node:
    def __init__(self, n=None):
        self.data=[]
        self.next=n
    def add(self, data):
        self.data.append(data)

    def __str__(self):
        return ' '.join(map(str, self.data))
```

Название файла: UnrolledLinkedList.py

```
from Node import Node
```

```
    class UnrolledLinkedList:
        def __init__(self, arr=None, n_size=16):
            self.head=Node()
            self.nodeSize=n_size
            self.last=self.head
            if arr is not None:
                for i in arr:
                    self.push(i)

        def push(self, item):
            self.last.add(item)
            self.rebalance()

        def rebalance(self):
            currNode=self.head
            while currNode:
                if len(currNode.data)>self.nodeSize:
                    newNode=Node()
                    currNode.next=newNode
                    self.last=currNode.next
                if currNode.next:
                    while len(currNode.data) > self.nodeSize:
                        currNode.next.data.insert(0,
currNode.data.pop())
                    while len(currNode.data) < self.nodeSize:
                        if currNode.next.data:
                            currNode.data.insert(self.nodeSize-1,
currNode.next.data.pop(0))
                        else:
                            break

                if len(self.last.data)==0 and self.last != self.head:
                    prev=self.head
                    while prev.next!=self.last:
                        prev=prev.next
                    prev.next=None
                    self.last=prev
                currNode=currNode.next
```

```
def valSearch(self, value):
    currNode=self.head
    i=0
    while currNode.next:
        for j in currNode.data:
            if j == value:
                return (i, j)
        i+=1
        currNode=currNode.next
    return (-1, -1)

def IndexRemove(self, nIndex, pIndex):
    currNode = self.head
    for i in range(nIndex):
        if currNode is None or currNode.next is None:
            return
        currNode=currNode.next
    if pIndex<=self.nodeSize:
        return currNode.data.pop(pIndex)
    self.rebalance()

def IndexSearch(self, nIndex, pIndex):
    currNode = self.head
    for i in range(nIndex):
        if currNode is None or currNode.next is None:
            return
        currNode=currNode.next
    if pIndex<=self.nodeSize:
        return currNode.data[pIndex]
    self.rebalance()
def valRemove(self, value):
    currNode=self.head
    i=0
    while currNode:
        if i == 0:
            for j in range(len(currNode.data)):
                if currNode.data[j] == value:
                    currNode.data.pop(j)
                    i=+1
```

```
                        break
                currNode=currNode.next
            else:
                break
        self.rebalance()
    def insert(self, index, val):
        currNode = self.head
        while currNode is not None:
            if index < len(currNode.data):
                currNode.data.insert(index, val)
                break
            index-=len(currNode.data)
            currNode=currNode.next
        self.rebalance()
    def showList(self):
        currNode=self.head
        i=0
        while currNode is not None:
            print(f'Node {i}:', end=' ')
            print(*currNode.data)
            i+=1
            currNode=currNode.next
```

Название файла: tests.py

```
from UnrolledLinkedList import UnrolledLinkedList
def test_init():
    subject = UnrolledLinkedList()
    assert subject.head.data == []
    assert subject.nodeSize == 16

    subject = UnrolledLinkedList([1, 10, 11])
    assert subject.head.data == [1,10,11]
    assert subject.nodeSize == 16

    subject = UnrolledLinkedList([1,10,11], 8)
    assert subject.head.data == [1,10,11]
    assert subject.nodeSize == 8

def test_push():
```

```
    subject=UnrolledLinkedList()
    subject.push(1)
    assert subject.head.data==[1]

    subject.push(10)
    assert subject.head.data==[1, 10]

def test_rebalance():
    subject = UnrolledLinkedList([1,2,3,4,5,6,7,8,9,10,11,12,13], 5)
    assert subject.head.data==[0, 1, 2, 3, 4]
    assert subject.head.next.data==[5,6,7,8,9]
    assert subject.head.next.next.data==[10,11,12]

def test_valSearch():
    subject = UnrolledLinkedList([1,2,3,4,5,6,7,8,9,10,11,12,13])
    assert subject.valSearch(3) == (0, 4)
    assert subject.valSearch(9) == (1, 4)
    assert subject.valSearch(12) == (2, 2)

def test_IndexRemove():
    subject = UnrolledLinkedList([1, 10, 11, 100, 101])
    subject.IndexRemove(0, 4)
    assert subject.head.data==[1,10,11,100]

    subject.IndexRemove(0, 0)
    assert subject.head.data==[10, 11, 100]

    subject.IndexRemove(0, 1)
    assert subject.head.data==[10,100]

def test_IndexSearch():
    subject = UnrolledLinkedList([1, 10, 11, 101, 110, 111], 2)
    assert subject.IndexSearch(0, 1)==10
    assert subject.IndexSearch(1, 0)==11
    assert subject.IndexSearch(2, 0)==110

def test_valRemove():
    subject = UnrolledLinkedList([1, 10, 11, 100, 101, 110, 111])
    subject.valRemove(1)
```

```
    assert subject.head.data==[10, 11, 100, 101, 110, 111]
    subject.valRemove(101)
    assert  subject.head.data==[10, 11, 100, 110, 111]
    subject.valRemove(111)
    assert  subject.head.data==[10,11,100,110]

def test_insert():
    subject = UnrolledLinkedList([1,2,3])
    subject.insert(0, 8)
    assert subject.head.data==[8,1,2,3]
    subject.insert(2, 7)
    assert subject.head.data==[8,1,7,2,3]
    subject.insert(5)
    assert subject.head.data==[8,1,7,2,3,5])
```