**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 исследования развернутого связного списка

|  |  |  |
| --- | --- | --- |
| Студент гр. 3344 |  | Тукалкин. В.А. |
| Преподаватель |  | Иванов Д.В. |

Санкт-Петербург

2024

## Цель работы

Создание структуры данных, которая называется развёрнутым связным 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).

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. 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

1. Реализация методов класса Node:

* \_\_init\_\_ – O(n) – метод для создания узла

1. Реализация методов класса UnrolledLinkedList:

* \_\_init\_\_ – O(n) – инициализирует данные, для работы со списком
* \_\_str\_\_ – O(n) – метод для преобразования данных в строку
* \_\_iter\_\_ – O(n) – возвращает итератор
* find – O(n) – находит элементы в списке
* balanced – O(n) – пересоздаёт список для корректного распределения элементов
* pushback – O(n) – добавляет элементы в конец списка
* insert – O(n) – добавляет элементы по индексу
* pushstart – O(1) – добавляет элементы в начало списка
* remove – O(n) – удаляет элементы из списка

1. Была создана функция для расчёта оптимальной длины массива для списка.
2. Написаны функции для тестирования всего вышеперечисленного.

## Тестирование программы

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 корректную работу балансировки и других функций.

## Выводы

Была создана структура данных, которая позволяет хранить большое количество информации, тратя меньшее количество памяти.

# Приложение А Исходный код программы

Название файла: main.py

from modules.ULL import UnrolledLinkedList

def check(arr\_1, arr\_2):

unroll\_list = UnrolledLinkedList(arr\_1)

print(unroll\_list)

print('-' \* 20)

for i in arr\_2:

unroll\_list.remove(i)

print(unroll\_list)

print('-' \* 20)

unroll\_list.balanced()

print(unroll\_list)

check([1, 2, 3, 4, 5, 6, 7, 8, 9, 10], [1, 6, 4, 9])

#print(UnrolledLinkedList(input('Enter your numbers separated by a space:').split()))

a=UnrolledLinkedList([1,2,3,4,5,6,7,8,9,10])

a.remove(2)

print(a)

Название файла: calculating.py

def calculate\_optimal\_node\_size(num):

return (num \* 4 + 63) // 64 + 1

Название файла: Node.py

class Node:

def \_\_init\_\_(self, val=None):

self.val = val

self.next = None

Название файла: ULL.py

from modules.Node import Node

from modules.calculating import calculate\_optimal\_node\_size

class UnrolledLinkedList:

def \_\_init\_\_(self, values=[]):

self.head = None

if values:

length = calculate\_optimal\_node\_size(len(values))

self.head = Node(values[:length])

current = self.head

for i in range(1, (len(values) + length - 1) // length):

current.next = Node(values[i \* length:(i + 1) \* length])

current = current.next

def \_\_str\_\_(self):

if not self.head:

return 'empty list'

output = ''

for num, item in enumerate(self):

nodes = ' '.join(str(x) for x in item)

output += f'Node {num}: {nodes}\n'

return output

def \_\_iter\_\_(self):

current = self.head

while current:

yield current.val

current = current.next

def find(self, val):

if isinstance(val, list):

message = ''

for i in val:

flag = 0

for j in self:

if i in j:

flag = 1

if flag == 1:

message += f'{i} in list\n'

else:

message += f'{i} not in list\n'

return message

if isinstance(val, int):

for i in self:

if val in i:

return True

return False

def balanced(self):

newList = []

while self.head:

newList.extend(self.head.val)

self.head = self.head.next

new = UnrolledLinkedList(newList)

self.head = new.head

def pushback(self, val):

if isinstance(val, int):

val = [val]

newNode = Node([int(x) for x in val])

current = self.head

while current.next:

current = current.next

newNode.next = current.next

current.next = newNode

def insert(self, val, index):

counter = 0

current = self.head

while current:

counter += len(current.val)

if counter >= index:

current.val.insert(index, val)

return

current = current.next

def pushstart(self, val):

if isinstance(val, int):

val = [val]

newNode = Node(val)

newNode.next = self.head

self.head = newNode

def remove(self, val):

if isinstance(val, int):

for i in self:

if val in i:

i.pop(i.index(val))

return self

return f'{val} not in list'

if isinstance(val, list):

for i in val:

for j in self:

if i in j:

j.pop(j.index(i))

if len(j) == 0:

self.head = self.head.next

return self

Название файла: tests.py

from modules.Node import Node

from modules.calculating import calculate\_optimal\_node\_size

from modules.ULL import UnrolledLinkedList

def create\_list():

return UnrolledLinkedList([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])

def test\_calculate\_optimal\_node\_size():

# Проверка расчёта длины узла

assert calculate\_optimal\_node\_size(10) == 2

def test\_Node():

# Проверка создания узла

a = Node(1)

a.next = Node(2)

assert a.next.val == 2

def test\_initialization\_1():

# Проверка пустого списка

empty\_list = UnrolledLinkedList()

assert str(empty\_list) == "empty list"

def test\_initialization\_2():

# Проверка создания списка с значениями

unrolled\_list = create\_list()

expected\_output = 'Node 0: 1 2\nNode 1: 3 4\nNode 2: 5 6\nNode 3: 7 8\nNode 4: 9 10\n'

assert str(unrolled\_list) == expected\_output

def test\_iteration():

unrolled\_list = create\_list()

actual\_result = [item for item in unrolled\_list]

expected\_result = [[1, 2], [3, 4], [5, 6], [7, 8], [9, 10]]

assert actual\_result == expected\_result

def test\_find\_1():

unrolled\_list = create\_list()

# Поиск одного числа

result = unrolled\_list.find(5)

assert result == True

def test\_find\_2():

unrolled\_list = create\_list()

# Поиск списка чисел

result = unrolled\_list.find([5, 7])

assert result == '5 in list\n7 in list\n'

def test\_find\_3():

unrolled\_list = create\_list()

# Поиск отсутствующего числа

result = unrolled\_list.find(11)

assert result == False

def test\_balanced():

unrolled\_list = create\_list()

unrolled\_list.pushback([11, 12, 13])

unrolled\_list.balanced()

expected\_output = "Node 0: 1 2\nNode 1: 3 4\nNode 2: 5 6\nNode 3: 7 8\nNode 4: 9 10\nNode 5: 11 12\nNode 6: 13\n"

assert str(unrolled\_list) == expected\_output

def test\_pushback\_1():

unrolled\_list = create\_list()

# Добавление списка чисел

unrolled\_list.pushback([11, 12])

expected\_output = "Node 0: 1 2\nNode 1: 3 4\nNode 2: 5 6\nNode 3: 7 8\nNode 4: 9 10\nNode 5: 11 12\n"

assert unrolled\_list.\_\_str\_\_() == expected\_output

def test\_pushback\_2():

unrolled\_list = create\_list()

# Добавление числа

unrolled\_list.pushback(11)

expected\_output = "Node 0: 1 2\nNode 1: 3 4\nNode 2: 5 6\nNode 3: 7 8\nNode 4: 9 10\nNode 5: 11\n"

assert str(unrolled\_list) == expected\_output

def test\_insert():

unrolled\_list = create\_list()

unrolled\_list.insert(100, 5)

unrolled\_list.balanced()

expected\_output = "Node 0: 1 2\nNode 1: 3 4\nNode 2: 5 6\nNode 3: 100 7\nNode 4: 8 9\nNode 5: 10\n"

assert str(unrolled\_list) == expected\_output

def test\_pushstart\_1():

# Проверка вставки числа

unrolled\_list = create\_list()

unrolled\_list.pushstart(10)

expected\_output = "Node 0: 10\nNode 1: 1 2\nNode 2: 3 4\nNode 3: 5 6\nNode 4: 7 8\nNode 5: 9 10\n"

assert str(unrolled\_list) == expected\_output

def test\_pushstart\_2():

# Проверка вставки списка чисел

unrolled\_list = create\_list()

unrolled\_list.pushstart([10, 11])

expected\_output = "Node 0: 10 11\nNode 1: 1 2\nNode 2: 3 4\nNode 3: 5 6\nNode 4: 7 8\nNode 5: 9 10\n"

assert str(unrolled\_list) == expected\_output

def test\_remove\_1():

# Проверка удаления числа, не входящего в список

unrolled\_list = create\_list()

assert unrolled\_list.remove(-1) == "-1 not in list"

def test\_remove\_2():

# Проверка удаления числа

unrolled\_list = create\_list()

unrolled\_list.remove(2)

expected\_output = "Node 0: 1\nNode 1: 3 4\nNode 2: 5 6\nNode 3: 7 8\nNode 4: 9 10\n"

assert str(unrolled\_list) == expected\_output

def test\_remove\_3():

# Проверка удаления списка чисел

unrolled\_list = create\_list()

unrolled\_list.remove([1, 2])

expected\_output = "Node 0: 3 4\nNode 1: 5 6\nNode 2: 7 8\nNode 3: 9 10\n"

assert str(unrolled\_list) == expected\_output

test\_initialization\_1()

test\_initialization\_2()

test\_iteration()

test\_find\_1()

test\_find\_2()

test\_find\_3()

test\_balanced()

test\_pushback\_1()

test\_pushback\_2()

test\_insert()

test\_pushstart\_1()

test\_pushstart\_2()

test\_remove\_1()

test\_remove\_2()

test\_remove\_3()