**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра САПР**

отчет

**по лабораторной работе №1**

**по дисциплине «Алгоритмы и структуры данных»**

**Вариант 1**

|  |  |  |
| --- | --- | --- |
| Студент гр. 1302 |  | Романова О.В. |
| Преподаватель |  | Родионова Е.А. |

Санкт-Петербург

2022

Постановка задачи

Задача: реализовать объект в виде связного списка с набором методов/функций. Данные, хранящиеся в списке, могут быть любого типа.

**Классы:**

|  |  |
| --- | --- |
| Node | Элемент списка. Состоит из value (значения элемента, тип int) и next, указывающего на следующий элемент |
| LinkedList | Связный список. Состоит из head типа Node, указывающий на первый элемент списка и size – размер списка. |

**Методы LinkedList:**

|  |  |
| --- | --- |
| is\_empty() | True, если список пуст, иначе – False |
| get\_size() | Текущий размер |
| get(index: int) | Возвращает элемент по индексу |
| print() | Выводит список |
| insert(index: int, value: int) | Вставка value перед index |
| add\_left(value: int) | Вставляет value в конец списка |
| add\_right(value: int) | Вставляет value в начало списка |
| remove(index: int) | Удаляет элемент по индексу |
| remove\_last() | Удаляет последний элемент |
| remove\_first() | Удаляет первый элемент |
| replace(index: int, value: int) | Заменяет элемент под индексом index на value |
| delete\_all() | Удаляет все элементы |
| find\_last\_list\_entrance(li: LinkedList) | Возвращает индекс последнего вхождения li в список |

Оценка временной сложности методов

|  |  |
| --- | --- |
| is\_empty() | O(1) |
| get\_size() | O(1) |
| get(index: int) | O(n) |
| print() | O(n) |
| insert(index: int, value: int) | O(n) |
| add\_left(value: int) | O(1) |
| add\_right(value: int) | O(n) |
| remove(index: int) | O(n) |
| remove\_last() | O(n) |
| remove\_first() | O(1) |
| replace(index: int, value: int) | O(n) |
| delete\_all() | O(n) |
| insert\_list(index: int, li: LinkedList) | O(n) |

Пример работы

Листинг

class Node:

pass

class LinkedList:

pass

class Node:

def \_\_init\_\_(self, val: int):

self.value = val

self.next = None

def \_\_eq\_\_(self, other: Node):

return self.value == other.value

def \_\_repr\_\_(self):

next\_value = self.next.value if self.next else None

return f"<Node(value={self.value}, next=<Node(value={next\_value})>)>"

class LinkedList:

def \_\_init\_\_(self, els: list = None):

self.size = 0

if els is None or len(els) == 0:

self.head = None

else:

self.head = Node(els[0])

node = self.head

self.size = 1

for el in els[1:]:

node.next = Node(el)

node = node.next

self.size += 1

def \_\_repr\_\_(self):

return f"<LinkedList(head={self.head}, size={self.size})>"

def is\_empty(self):

# O(1)

return self.size == 0

def get\_size(self):

# O(1)

return self.size

def get(self, index: int):

# O(n)

if index >= self.size or index < 0:

return None

node = self.head

for i in range(index):

node = node.next

return node

def print(self):

# O(n)

node = self.head

for i in range(self.size):

if (node is None):

ValueError("Error: size do not match actual nodes number")

return

print(i, ": ", node.value, sep='')

node = node.next

def insert(self, index: int, value: int):

# O(n)

# index == 0 => O(1)

# index =! 0 => O(n)

if not 0 <= index <= self.size:

print("Error: invalid index")

return

if not isinstance(value, int):

print("Error: value must be int")

return

node = Node(value)

if index == 0:

# O(1)

node.next = self.head

self.head = node

elif index == self.size:

# O(n)

last\_el = self.get(self.size - 1)

if last\_el is None:

return

last\_el.next = node

else:

# O(n)

prev\_el = self.get(index - 1)

next\_el = self.get(index)

prev\_el.next = node

node.next = next\_el

self.size += 1

def add\_left(self, value: int):

# O(1)

self.insert(0, value)

def add\_right(self, value: int):

# O(n)

if not isinstance(value, int):

print("Error: value must be int")

return

self.insert(self.size, value)

def remove(self, index: int):

# O(n)

if not 0 <= index < self.size:

print("Error: invalid index")

return

prev\_el = self.get(index - 1)

next\_el = self.get(index + 1)

if prev\_el is None and next\_el is None:

# O(1)

self.head = None

elif prev\_el is None:

# O(1)

self.head = self.get(1)

elif next\_el is None:

# O(1)

prev\_el.next = None

else:

# O(1)

prev\_el.next = next\_el

self.size -= 1

def remove\_last(self):

# O(n)

self.remove(self.size - 1)

def remove\_first(self):

# O(1)

self.remove(0)

def replace(self, index: int, value: int):

# O(n)

if not 0 <= index < self.size:

print("Error: invalid index")

return

self.remove(index)

self.insert(index, value)

def delete\_all(self):

# O(n)

while (self.head is not None):

self.remove\_first()

def insert\_list(self, index: int, li: LinkedList):

if li.size == 0 or index < 0 or index > self.size:

print("Error: invalid index")

return

if index == 0:

head = self.head

self.head = li.head

li.get(li.size - 1).next = head

elif index == self.size:

self.get(self.size - 1).next = li.head

else:

prev\_el = self.get(index - 1)

next\_el = prev\_el.next

prev\_el.next = li.head

li.get(li.size - 1).next = next\_el

self.size += li.size

def main():

li = LinkedList([2, 3, 4, 0, -1, 0, 1, 3, 4, 5, 6, -1, 0, 1, 2, 3, 4, 5, 6, 7])

li.remove\_first()

li.remove\_last()

li.add\_left(0)

li.add\_right(10)

li.insert(3, 13)

li.print()

li.insert\_list(4, LinkedList([5, -2, 4, 0]))

li.print()

print(li.is\_empty())

li.delete\_all()

li.print()

print(li.is\_empty())

if \_\_name\_\_ == "\_\_main\_\_":

main()