##### ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

##### «НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

##### КАФЕДРА ВТ

##### ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №2

##### «Разработка класса для представления структуры данных» по дисциплине «Технологии разработки программного обеспечения»

##### Выполнили: студенты гр. АММ2-24 Атласюк Игорь Романович

##### Ириков Евгений Алексеевич

##### Проверил: к.т.н., доцент Кафедры ВТ Токарев Вадим Геннадьевич

##### Новосибирск 2024

## Содержание

[Постановка задачи 3](#_bookmark0)

[Выполнение работы 4](#_bookmark1)

[Результат работы 12](#_bookmark2)

[Приложение 13](#_bookmark3)

# Постановка задачи

# Разработать класс для представления структуры данных. Набор операция для структуры данных:

# добавление в конец, получение, вставка и удаление по логическому номеру (индексу). Логический номер – порядковый номер хранимого элемента в соответствии с последовательностью обхода структуры данных (перебора элементов)

# для исходно упорядоченной (например, «двоичное дерево») вставка с сохранением порядка, получение и удаление по по логическому номеру (индексу)

# Итератор forEach – для Java: интерфейс обратного вызова с методом void toDo(T v), итератора получает ссылку на интерфейсный объект (анонимный класс), обходит структуру данных и вызывает метод для каждого выбранного элемента. В Scala – формальный параметр = функция.

# сортировка, для исходно упорядоченной – балансировка = создание экземпляра структуры данных с равными размерностями компонент (например, сбалансированное дерево, либо список списков со списками нижнего уровня равной длины)

# Элемент структуры данных хранит целое число. Вариант задания 7 - односвязный список.

# Выполнение работы

# Для реализации операций добавления, удаления, поиска элементов, сортировки односвязного списка был разработан класс LinkedList. Ниже идет описание функций, реализованных в классе.

# 

# Рисунок 1. Класс LinkedList

# На рисунке 1 изображены поля класса LinkedList и метод get. Поля:

# ListNode head: cсылка на первый элемент списка.

# int size: количество элементов в списке.

# Метод get возвращает элемент списка по указанному индексу. Итерирует по списку от головы (head) до нужного индекса и возвращает данные, если элемент найден. Если индекс выходит за пределы списка, вызывает исключение.

# 

# Рисунок 2. Метод insert

# На рисунке 2 изображен метод insert, который вставляет новый элемент в список на указанной позиции, но, если индекс равен 0, элемент добавляется в начало списка (новая голова). В противном случае итерирует по списку до позиции перед указанным индексом и вставляет новый элемент, увеличивая размер списка на 1.

# 

# Рисунок 3. Метод delete

# На рисунке 3 изображен метод delete, который удаляет элемент по указанному индексу, но, если индекс равен 0, удаляет первый элемент (перемещает голову на следующий элемент). Итерирует до элемента перед указанным индексом и изменяет ссылки для удаления элемента, уменьшая размер списка на 1.

# 

# Рисунок 4. Методы add и forEach

# На рисунке 4 изображен метод add и forEach. Метод add - добавляет новый элемент в конец списка, если список пуст, новый элемент становится головой. В противном случае итерирует до конца списка и добавляет новый элемент в конец, увеличивая размер списка на 1. Метод forEach - применяет переданную функцию к каждому элементу списка.

# 

# Рисунок 5. Сортировка

# На рисунке 5 изображен метод sort, который сортирует список с использованием метода слияния, вызывая вспомогательный метод mergeSort в котором реализована логика сортировки слиянием. Также изображен метод size, который возвращает количество элементов в списке.

# 

# Рисунок 6. Сортировка

# На рисунке 6 изображены методы для реализации сортировки слиянием.

# 

# Рисунок 7. Класс ListNode

# На рисунке 7 изображены класс ListNode представляющий узел списка. Каждый узел содержит:

# Integer data: данные узла.

# ListNode next: ссылка на следующий узел.

# Также в классе реализованы методы:

# getData(): возвращает данные текущего узла.

# setData(int data): устанавливает новые данные в узел.

# getNext(): возвращает ссылку на следующий узел.

# setNext(ListNode next): устанавливает ссылку на следующий узел.

# Далее идет описание класса Main.

# 

# Рисунок 8. Класс Main

# На рисунке 8 изображен класс Main, где создается экземпляр класса LinkedList, который изначально пуст. Далее в список добавляются элементы и выводится список на экран. Также происходит получение элемента списка по индексу, вставка нового элемента, удаление элемента, сортировка

# Результат работы

# Пример использования программы.

# Введем элементы списка [1, 8, 2, 3, 0, 1, 5];

# Выведем третий элемент списка (второй по индексу)

# 

# Добавим на число 66 на третье место и выведем его

# 

# Удалим третий элемент списка и выведем его

# 

# Добавим в конец число 666 и выведем последний элемент списка

# 

# Отсортируем список

# 

# Полное изображение работы программы

# 

# Приложение

# Class Main.java

# package ru.atlasyk;

# import list.LinkedList;

# public class Main {

# public static void main(String[] args) {

# var linkedList = new LinkedList();

# linkedList.add(1);

# linkedList.add(8);

# linkedList.add(2);

# linkedList.add(3);

# linkedList.add(0);

# linkedList.add(1);

# linkedList.add(5);

# linkedList.forEach(x -> System.out.print(x + " "));

# System.out.println();

# System.out.println("Third element: " + linkedList.get(2));

# linkedList.insert(2, 66);

# System.out.println("After insert: ");

# linkedList.forEach(x -> System.out.print(x + " "));

# System.out.println();

# System.out.println("Third element: " + linkedList.get(2));

# linkedList.delete(2);

# System.out.println("After delete: ");

# linkedList.forEach(x -> System.out.print(x + " "));

# System.out.println();

# System.out.println("Third element: " + linkedList.get(2));

# System.out.println("After adding to end: ");

# linkedList.add(666);

# linkedList.forEach(x -> System.out.print(x + " "));

# System.out.println();

# System.out.println("Last element: " + linkedList.get(linkedList.size() - 1));

# System.out.println("Before sort: ");

# linkedList.forEach(x -> System.out.print(x + " "));

# System.out.println();

# linkedList.sort();

# System.out.println("After sort:");

# linkedList.forEach(x -> System.out.print(x + " "));

# System.out.println();

# }

# }

# Class LinkedList.java

# package list;

# import java.util.Objects;

# import java.util.function.Consumer;

# public class LinkedList {

# private ListNode head;

# private int size;

# // Получить элемент по индексу

# public Integer get(int index) {

# ListNode current = head;

# int count = 0;

# while (current != null) {

# if (count == index) {

# return current.getData();

# }

# count++;

# current = current.getNext();

# }

# throw new IndexOutOfBoundsException("Index: " + index + ", Size: " + count);

# }

# // Вставить элемент по индексу

# public void insert(int index, Integer data) {

# if (index < 0) {

# throw new IndexOutOfBoundsException("Index cannot be negative: " + index);

# }

# ListNode newNode = new ListNode(data);

# if (index == 0) {

# newNode.setNext(head);

# head = newNode;

# return;

# }

# ListNode current = head;

# for (int i = 0; i < index - 1; i++) {

# if (current == null) {

# throw new IndexOutOfBoundsException("Index: " + index);

# }

# current = current.getNext();

# }

# newNode.setNext(current.getNext());

# current.setNext(newNode);

# size++;

# }

# // Удалить элемент по индексу

# public void delete(int index) {

# if (index < 0 || head == null) {

# throw new IndexOutOfBoundsException("Index: " + index);

# }

# if (index == 0) {

# head = head.getNext();

# return;

# }

# ListNode current = head;

# for (int i = 0; i < index - 1; i++) {

# if (current == null || current.getNext() == null) {

# throw new IndexOutOfBoundsException("Index: " + index);

# }

# current = current.getNext();

# }

# if (current.getNext() == null) {

# throw new IndexOutOfBoundsException("Index: " + index);

# }

# current.setNext(current.getNext().getNext());

# size--;

# }

# // Добавить элемент в конец списка

# public void add(Integer data) {

# ListNode newNode = new ListNode(data);

# if (head == null) {

# head = newNode;

# size++;

# return;

# }

# ListNode current = head;

# while (current.getNext() != null) {

# current = current.getNext();

# }

# current.setNext(newNode);

# size++;

# }

# // Применить функцию к каждому элементу списка

# public void forEach(Consumer<? super Number> func) {

# Objects.requireNonNull(func);

# ListNode current = head;

# while (current != null) {

# func.accept(current.getData());

# current = current.getNext();

# }

# }

# // Сортировка списка (метод слияния)

# public void sort() {

# head = mergeSort(head);

# }

# public int size() {

# return size;

# }

# // Вспомогательный метод для сортировки

# private ListNode mergeSort(ListNode node) {

# if (node == null || node.getNext() == null) {

# return node; // База рекурсии

# }

# // Разделяем список на две половины

# ListNode middle = getMiddle(node);

# ListNode nextOfMiddle = middle.getNext();

# middle.setNext(null); // Разделяем на две части

# // Рекурсивно сортируем обе половины

# ListNode left = mergeSort(node);

# ListNode right = mergeSort(nextOfMiddle);

# // Объединяем отсортированные половины

# return sortedMerge(left, right);

# }

# // Метод для получения середины списка

# private ListNode getMiddle(ListNode node) {

# if (node == null) {

# return null;

# }

# ListNode slow = node;

# ListNode fast = node.getNext();

# // Используем два указателя для нахождения середины

# while (fast != null) {

# fast = fast.getNext();

# if (fast != null) {

# slow = slow.getNext();

# fast = fast.getNext();

# }

# }

# return slow;

# }

# // Метод для слияния двух отсортированных списков

# private ListNode sortedMerge(ListNode left, ListNode right) {

# ListNode result;

# // База рекурсии

# if (left == null) {

# return right;

# } else if (right == null) {

# return left;

# }

# // Сравниваем данные и сливаем

# if (left.getData() <= right.getData()) {

# result = left;

# result.setNext(sortedMerge(left.getNext(), right));

# } else {

# result = right;

# result.setNext(sortedMerge(left, right.getNext()));

# }

# return result;

# }

# }

# class ListNode {

# Integer data;

# ListNode next;

# ListNode(int data, ListNode next) {

# this.data = data;

# this.next = next;

# }

# ListNode(int data) {

# this.data = data;

# }

# Integer getData() {

# return data;

# }

# void setData(int data) {

# this.data = data;

# }

# ListNode getNext() {

# return next;

# }

# void setNext(ListNode next) {

# this.next = next;

# }

# }