Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Российский химико-технологический университет имени Д.И. Менделеева»

Кафедра информационных компьютерных технологий

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № \_

Выполнил студент группы КС-36 (Потапов Никита Александрович)

Ссылка на репозиторий: (Ссылка на лабораторную в репозитории)

Приняли: Пысин Максим Дмитриевич

Краснов Дмитрий Олегович

Лобанов Алексей Владимирович

Крашенинников Роман Сергеевич

Дата сдачи: (07.04.2025)

Оглавление

[Описание задачи. 2](#_Toc63548272)

[Описание метода/модели. 2](#_Toc63548273)

[Выполнение задачи. 4](#_Toc63548274)

[Заключение. 7](#_Toc63548275)

# Описание задачи.

В рамках лабораторной работы необходимо изучить и реализовать бинарное дерево поиска и его самобалансирующийся вариант в лице AVL дерева.

Для проверки анализа работы структуры данных требуется провести 10 серий тестов.

В каждой серии тестов требуется выполнять 20 циклов генерации и операций. При этом первые 10 работают с массивом заполненным случайным образом, во второй половине случаев, массив заполняется в порядке возрастания значений индекса, т.е. является отсортированным по умолчанию.

Требуется создать массив состоящий из 2^(10 + i) элементов, где i это номер серии.

Массив должен быть помещен в оба варианта двоичных деревьев. При этому замеряется время затраченное на всю операцию вставки всего массива.

После заполнения массива, требуется выполнить 1000 операций поиска по обоим вариантам дерева, случайного числа в диапазоне генерируемых значений, замерев время на все 1000 попыток и вычислив время 1 операции поиска.

Провести 1000 операций поиска по массиву, замерить требуемое время на все 1000 операций и найти время на 1 операцию.

После, требуется выполнить 1000 операций удаления значений из двоичных деревьев, и замерить время затраченное на все операции, после чего вычислить время на 1 операцию.

После выполнения всех серий тестов, требуется построить графики зависимости времени затрачиваемого на операции вставки, поиска, удаления от количества элементов. При этом требуется разделить графики для отсортированного набора данных и заполненных со случайным распределением. Так же, для операции поиска, требуется так же нанести для сравнения график времени поиска для обычного массива.

# Описание метода/модели.

В данном коде реализованы две структуры данных: дерево бинарного поиска (BST) и AVL-дерево. Обе структуры предназначены для хранения данных в виде узлов, где каждый узел содержит ключ и ссылки на левого и правого потомков. Основное отличие между ними заключается в способе поддержания сбалансированности дерева.

**Дерево бинарного поиска (BST)**

Определение: BST — это структура данных, в которой для каждого узла выполняется следующее условие: ключи в левом поддереве меньше ключа узла, а ключи в правом поддереве больше.

Сложность алгоритмов:

Вставка: O(h), где h — высота дерева. В худшем случае (несбалансированное дерево) h может быть равно n, что приводит к O(n).

Поиск: O(h), аналогично вставке.

Удаление: O(h), также аналогично.

Потребление памяти: O(n), где n — количество узлов в дереве. Каждый узел хранит указатели на своих потомков.

*Проблемы и преимущества:*

Проблемы: В худшем случае (например, при вставке отсортированных данных) дерево может вырождаться в линейную структуру (список), что ухудшает производительность.

Преимущества: Простота реализации. Быстрый доступ к данным при сбалансированном состоянии.

**AVL-дерево**

Определение: AVL-дерево — это самобалансирующееся бинарное дерево поиска, которое поддерживает балансировку путем ограничения разницы высот левого и правого поддеревьев для каждого узла до 1.

Сложность алгоритмов:

Вставка: O(log n). После вставки может потребоваться выполнение операций поворота для восстановления баланса.

Поиск: O(log n). Высота дерева всегда остается логарифмической благодаря балансировке.

Удаление: O(log n). Как и при вставке, могут потребоваться операции поворота.

Потребление памяти: O(n), аналогично BST. Каждый узел хранит дополнительную информацию о высоте или балансе.

*Проблемы и преимущества:*

Проблемы: более сложная реализация по сравнению с обычным BST из-за необходимости поддержания балансировки. Операции поворота могут добавлять накладные расходы при частых вставках и удалениях.

Преимущества: Гарантированное логарифмическое время выполнения операций благодаря поддержанию сбалансированности. Более эффективен при частых операциях поиска по сравнению с несбалансированными деревьями.

# Выполнение задачи.

Практическое описание решения

В данном проекте была реализована программа для сравнения производительности двух структур данных: дерева бинарного поиска (BST) и AVL-дерева. Программа написана на языке программирования c++, который обеспечивает простоту и удобство в реализации алгоритмов и структур данных.

*Организация программы*

Программа организована в виде классов, каждый из которых отвечает за определенную функциональность:

1. Класс `Node`: представляет узел дерева, содержащий ключ, ссылки на левого и правого потомков, а также высоту узла.

2. Класс `BinarySearchTree`: реализует основные операции для дерева бинарного поиска, включая вставку, поиск и удаление узлов.

3. Класс `AVLTree`: наследует от класса `BinarySearchTree` и добавляет функциональность для поддержания сбалансированности дерева при выполнении операций вставки и удаления.

4. Функции генерации массивов: включают функции для создания случайных и отсортированных массивов, которые используются для тестирования производительности.

5. Линейный поиск: реализован как отдельная функция для сравнения с деревьями.

6. Основная функция `main()`: отвечает за выполнение тестов, измерение времени выполнения различных операций и запись результатов в файл.

*Проведенные тесты*

Для оценки производительности были проведены следующие тесты:

1. Генерация массивов:

- Случайные массивы размером от 1024 до 524288 (1024 \* 2^9).

- Отсортированные массивы того же размера.

2. Измерение времени выполнения:

- Линейный поиск по массиву.

- Вставка элементов в BST и AVL-дерево.

- Поиск элементов в BST и AVL-дереве.

- Удаление элементов из BST и AVL-дереве.

Каждый тест проводился 10 раз для каждого размера массива, чтобы получить более точные результаты.

*Полученные результаты*

Результаты тестирования были записаны в файл `output.txt`, где для каждого размера массива были указаны следующие показатели:

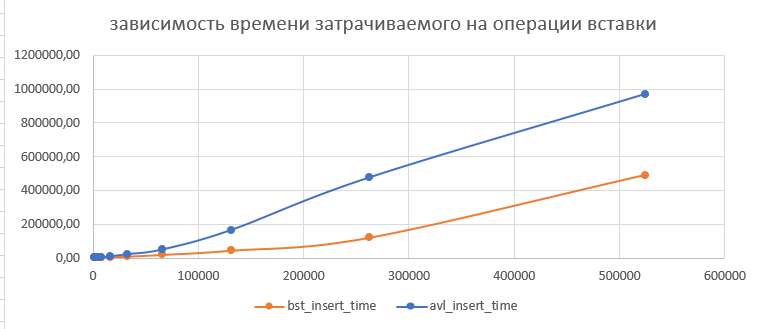
- Время выполнения линейного поиска.

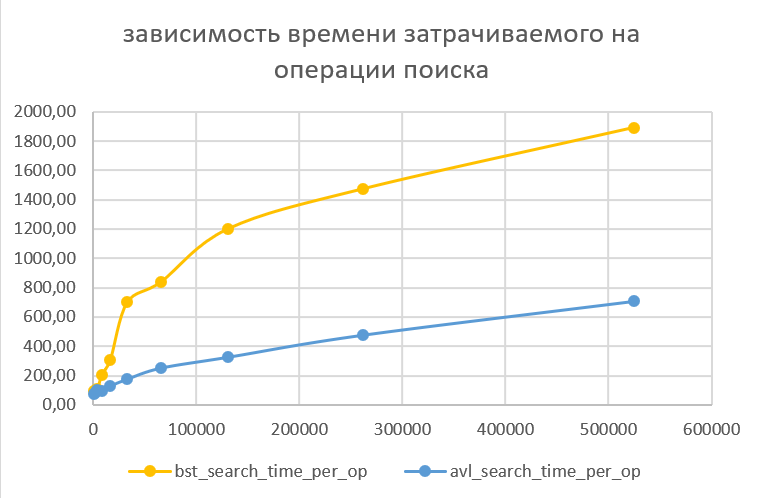
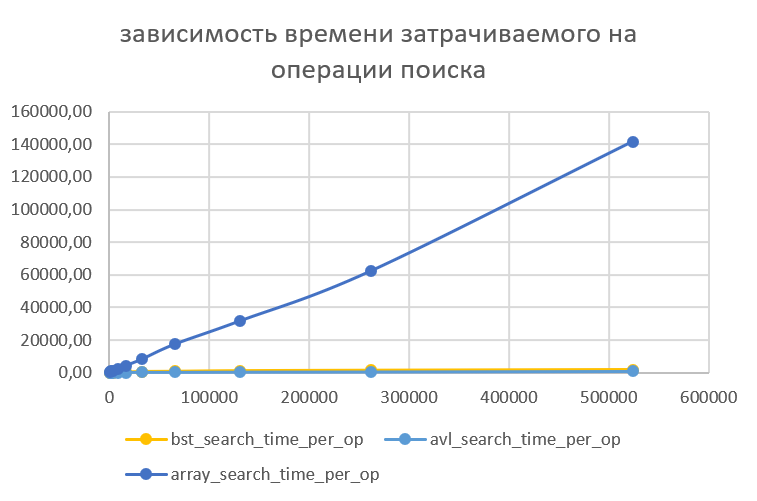
- Время выполнения вставки в BST и AVL-дерево.

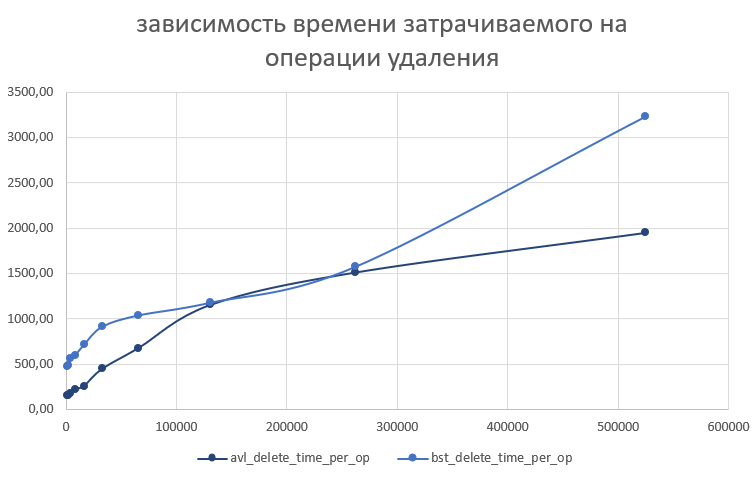
- Время выполнения поиска в BST и AVL-дереве.

- Время выполнения удаления из BST и AVL-дерева.

*Случайные данные*

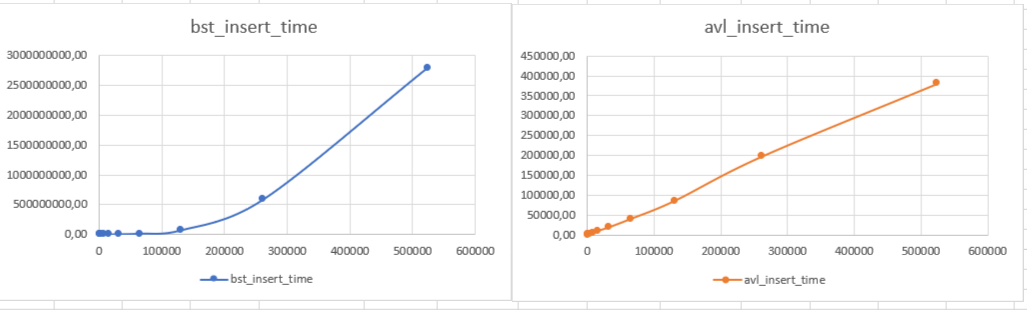




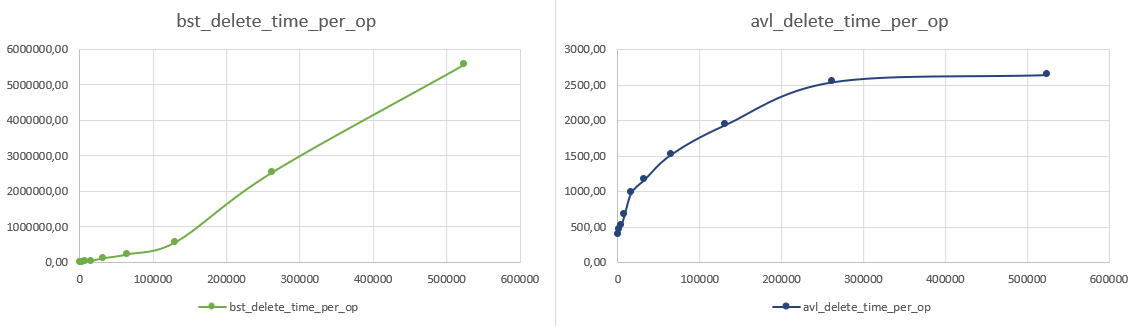


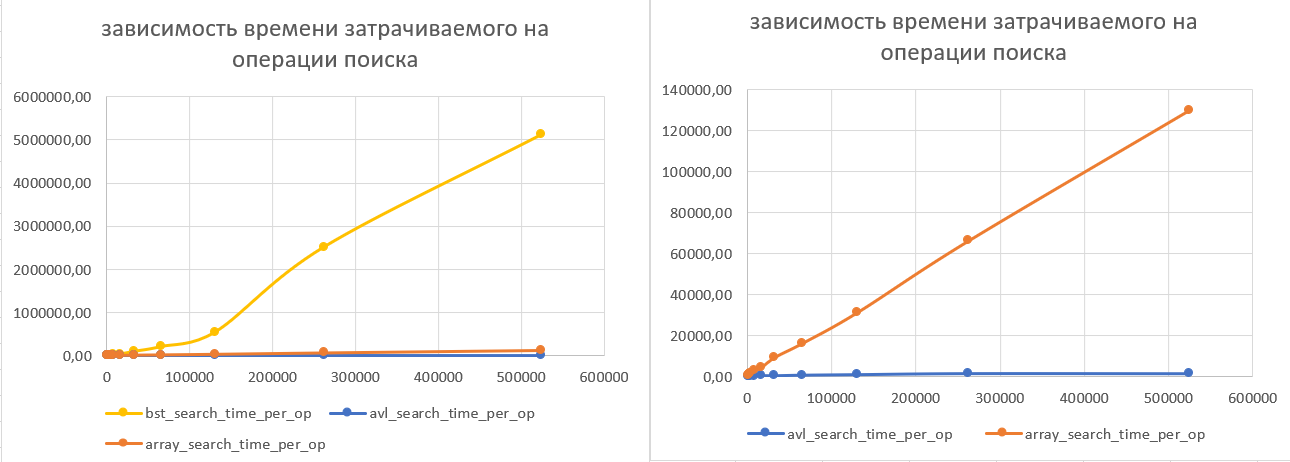
*Отсортированные данные*

зависимость времени, затрачиваемого на операции вставки

**

зависимость времени затрачиваемого на операции удаления

**

**

Основные выводы из результатов:

1. Линейный поиск показал значительно более высокое время выполнения по сравнению с деревьями, особенно на больших массивах, что подтверждает его неэффективность при работе с большими объемами данных.

2. AVL-дерево продемонстрировало лучшие результаты по сравнению с обычным BST во всех операциях (вставка, поиск, удаление), что подтверждает его преимущество в производительности благодаря сбалансированной структуре.

3. В случае использования отсортированных массивов время выполнения операций вставки в BST значительно увеличивалось из-за вырождения дерева в линейную структуру, тогда как AVL-дерево сохраняло логарифмическое время выполнения благодаря своей балансировке.

Таким образом, результаты тестирования подтвердили теоретические ожидания о преимуществах использования сбалансированных деревьев (AVL) по сравнению с обычными деревьями бинарного поиска (BST) и линейными структурами данных при работе с большими объемами информации.

# Заключение.

В ходе выполнения лабораторной работы были сделаны несколько ключевых выводов относительно применимости структур данных, сложности их реализации и общего опыта работы с ними.

1. Применимость структур данных:

- Дерево бинарного поиска (BST): Эта структура данных подходит для задач, где требуется быстрое выполнение операций поиска, вставки и удаления. Однако ее эффективность значительно снижается при работе с отсортированными данными, что может привести к вырождению дерева в линейную структуру. Поэтому BST лучше использовать в ситуациях, когда данные не имеют строгой упорядоченности.

- AVL-дерево: Эта структура данных является более предпочтительной для приложений, требующих частых операций поиска и модификации. Благодаря автоматической балансировке AVL-дерево обеспечивает логарифмическое время выполнения операций даже в худших случаях, что делает его более надежным выбором для работы с большими объемами данных.

2. Сложность реализации:

- Реализация BST оказалась достаточно простой и интуитивно понятной, что делает ее хорошим выбором для начинающих программистов. Однако при добавлении функциональности (например, удаление узлов) возникли некоторые сложности.

- Реализация AVL-дерева была более сложной задачей из-за необходимости поддержания балансировки после каждой операции вставки и удаления. Это потребовало понимания дополнительных концепций, таких как повороты узлов и высота дерева. Тем не менее, этот опыт оказался полезным для углубления знаний о сбалансированных структурах данных.

Лабораторная работа продемонстрировала важность выбора подходящей структуры данных для решения конкретных задач. AVL-дерево показало себя как более эффективное решение по сравнению с обычным BST в условиях частых изменений данных. Опыт реализации этих структур обогатил мои знания о алгоритмах и углубил понимание их применения в реальных сценариях программирования.