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 № 6

Выполнил студент группы КС-36 Акулинин Андрей Игоревич

Ссылка на репозиторий: <https://github.com/MUCTR-IKT-CPP/AkulininAI_36/tree/main/algorithms>

Приняли: Пысин Максим Дмитриевич

Краснов Дмитрий Олегович

Лобанов Алексей Владимирович

Крашенинников Роман Сергеевич

Дата сдачи: 17.03.2025

Оглавление

[Описание задачи. 2](#_Toc63548272)

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

[Выполнение задачи. 2](#_Toc63548274)

[Заключение. 2](#_Toc63548275)

# Описание задачи.

Цель работы — реализовать и сравнить две структуры данных:

1. **Бинарное дерево поиска (BST)**
2. **Самобалансирующееся AVL-дерево**

Требуется провести тестирование их производительности при:

* **Вставке** элементов (случайных и отсортированных)
* **Поиске** (сравнение с линейным поиском в массиве)
* **Удалении** элементов

Тестирование проводится на наборах данных разного размера (от 211 до 220 элементов).

# Описание метода/модели.

**2.1. Бинарное дерево поиска (BST)**

* **Структура:**
  + Каждый узел содержит ключ, левого и правого потомка.
  + Левые потомки меньше родителя, правые — больше.
* **Сложность операций (средний случай):**
  + Вставка: *O*(log*n*)
  + Поиск: *O*(log*n*)
  + Удаление: *O*(log*n*)
* **Проблемы:**
  + В худшем случае (отсортированные данные) вырождается в связный список (O(n)*O*(*n*)).

**2.2. AVL-дерево**

* **Структура:**
  + BST с балансировкой по высоте поддеревьев.
  + После каждой вставки/удаления выполняется балансировка (повороты).
* **Сложность операций (гарантированная):**
  + Вставка: *O*(log*n*)
  + Поиск: *O*(log*n*)
  + Удаление: *O*(log*n*)
* **Преимущества:**
  + Гарантированная сбалансированность.
  + Стабильное время операций даже на отсортированных данных.

**2.3. Линейный поиск в массиве**

* **Сложность:** O(n)*O*(*n*) (в худшем случае).
* **Используется для сравнения эффективности деревьев.**

# Выполнение задачи.

**3.1. Реализация**

* **Язык:** C++ (использованы std::chrono для замера времени, std::vector для хранения данных).
* **Структура программы:**
  1. **BST:**
     + Рекурсивная очистка (clear()).
     + Итеративные вставка, поиск и удаление.
  2. **AVL:**
     + Балансировка через повороты (rotateLeft, rotateRight).
     + Учет высоты узлов (height, balance\_factor).
  3. **Тестирование:**
     + Генерация случайных и отсортированных данных.
     + Замер времени для 1000 операций поиска/удаления.

**3.2. Проведенные тесты**

* **10 серий** (размер данных: 211,212,...,220).
* **20 циклов** в каждой серии:
  + 10 циклов — случайные данные.
  + 10 циклов — отсортированные данные.
* Измерялось время:
  + Вставки всего массива.
  + Поиска (1000 операций, усредненное время).
  + Удаления (1000 операций, усредненное время).

**3.3. Результаты**

* **Графики (построены в Python):**
  1. **Вставка:**
     + AVL медленнее BST на случайных данных из-за балансировки.
     + На отсортированных данных BST вырождается (*O*(*n*)), AVL остается *O*(log*n*).
  2. **Поиск:**
     + AVL и BST близки на случайных данных.
     + На отсортированных BST деградирует до *O*(*n*), AVL — *O*(log*n*).
     + Массив всегда *O*(*n*), но быстрее BST на отсортированных.
  3. **Удаление:**
     + Аналогично вставке: AVL стабилен, BST деградирует.

Изображение выглядит как текст, снимок экрана, линия, число

Контент, сгенерированный ИИ, может содержать ошибки.

# Заключение.

1. **BST**
   * Быстрее AVL на случайных данных (нет накладных расходов на балансировку).
   * Непригоден для отсортированных данных (деградация до *O*(*n*)).
2. **AVL**
   * Гарантирует O(logn)для всех операций.
   * Лучший выбор для динамических данных с частыми изменениями.
3. **Массив**
   * Поиск медленнее, но может быть эффективнее BST на отсортированных данных.

**Вывод:**

* Для неизменяемых данных подойдет BST.
* Для динамических данных с частыми вставками/удалениями — AVL.
* Балансировка увеличивает время вставки, но обеспечивает стабильность.