

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
«Национальный исследовательский Нижегородский государственный
университет им. Н.И. Лобачевского»

ОТЧЕТ К ЛАБОРАТОРНОЙ РАБОТЕ

«Реализация и экспериментальное сравнение базовых структур
данных на примере телефонного справочника»

Выполнил: студент В. В. Пронин

Преподаватель: Н. С. Морозов

Нижний Новгород

2024

Содержание

1	Введение	2
2	Реализация структур данных	2
2.1	Связный список	3
2.2	Хеш-таблица	3
2.3	Двоичное дерево поиска	3
3	Методика эксперимента	3
4	Результаты и анализ	3

1 Введение

Эффективность программных систем во многом определяется выбором способов организации данных в оперативной памяти. Задача разработки телефонного справочника является классическим примером, требующим баланса между скоростью вставки новых записей, поиском по ключу и эффективным удалением.

В рамках данной работы исследуются три фундаментальные структуры данных, реализованные «с нуля» в процедурной парадигме программирования на языке Python:

- **Связный список (Linked List)** — динамическая структура, позволяющая оценить базовые механизмы управления указателями и демонстрирующая линейную сложность операций $O(n)$.
- **Хеш-таблица (Hash Table)** — ассоциативный массив, использующий хеширование для обеспечения прямого доступа к данным. Реализация позволяет изучить методы разрешения коллизий и преимущества константной сложности $O(1)$.
- **Двоичное дерево поиска (BST)** — иерархическая структура, обеспечивающая логарифмическую скорость доступа $O(\log n)$ и поддерживающая упорядоченность данных «из коробки».

Цель работы: Изучить внутренние алгоритмы работы перечисленных структур, реализовать их без использования встроенных высокоуровневых контейнеров и экспериментально подтвердить теоретические оценки временной сложности на случайных и отсортированных наборах данных.

2 Реализация структур данных

2.1 Связный список

2.2 Хеш-таблица

2.3 Двоичное дерево поиска

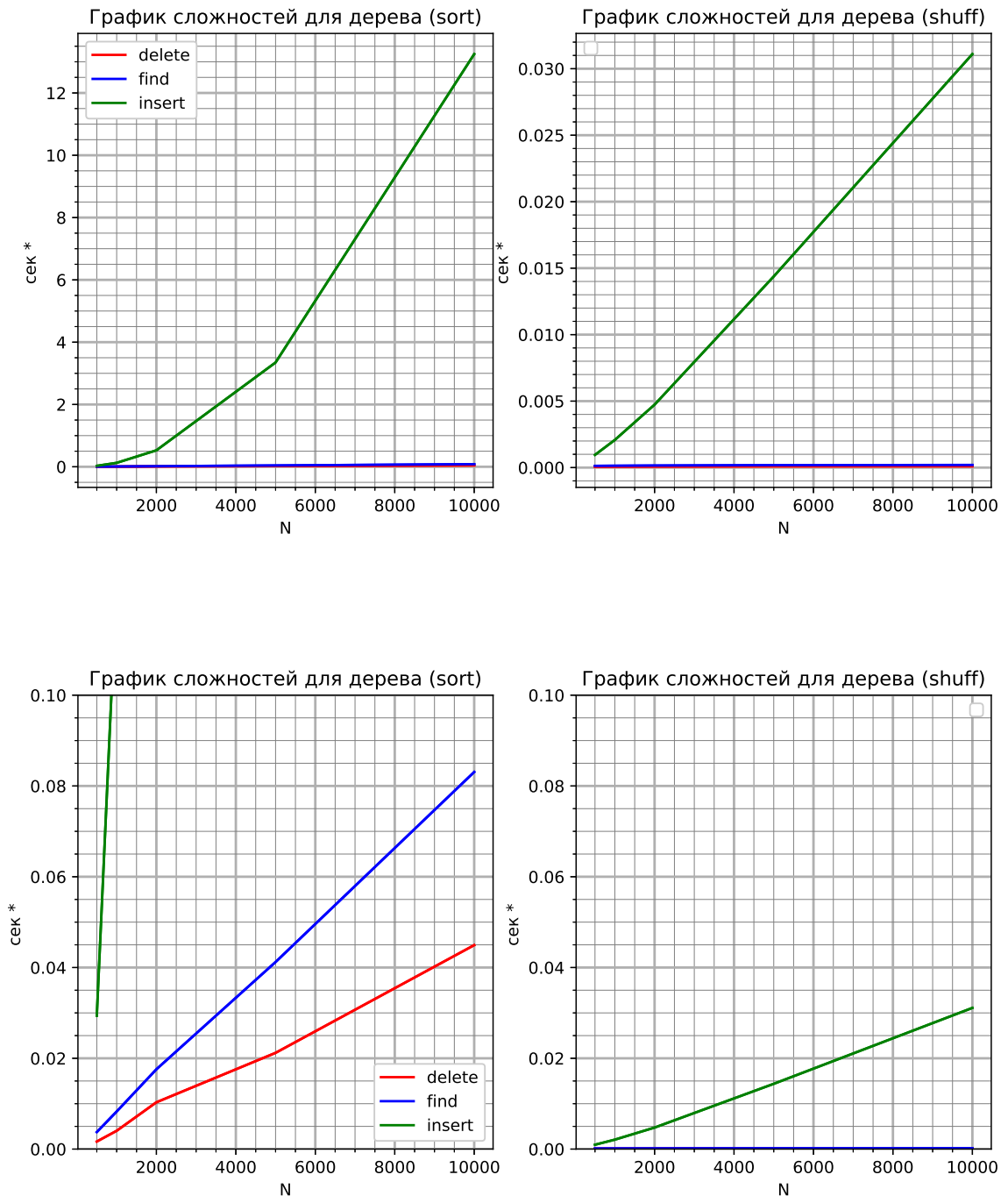
3 Методика эксперимента

Замеры производились для наборов данных объемом $N = 500, 1000, 2000, 5000$, элементов. Использовались два сценария: перемешанные (*shuffled*) и отсортированные по алфавиту (*sorted*) записи. Каждая операция выполнялась 5 раз с последующим вычислением среднего арифметического значения с помощью функции `time.perf_counter()`.

4 Результаты и анализ

Было проведено 5 опытов.

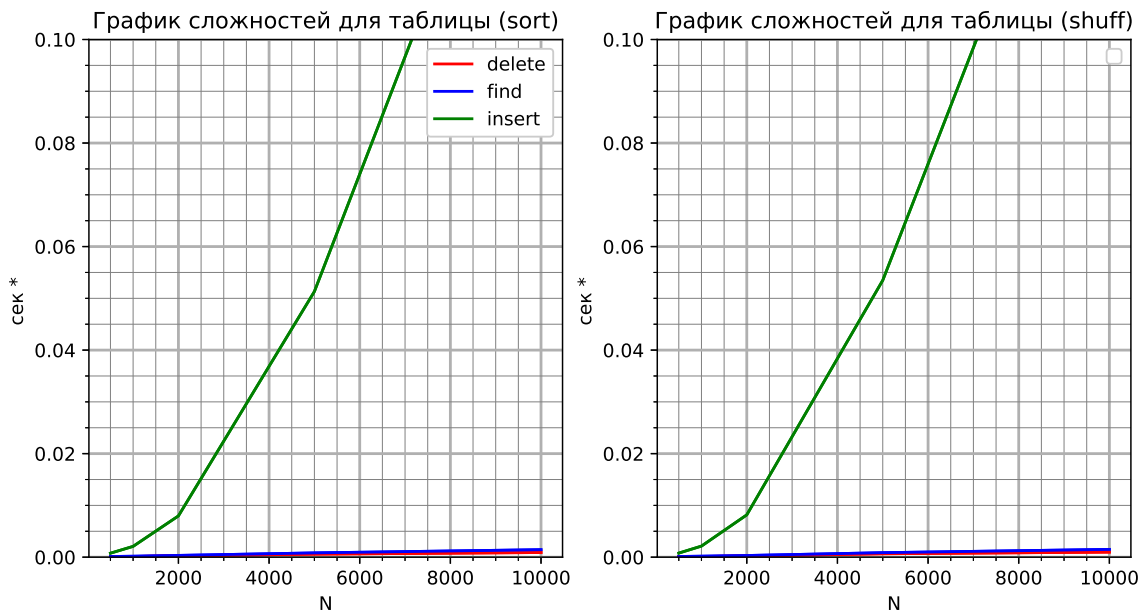
1. Бинарное дерево поиска (BST) и влияние порядка



- **Деградация на отсортированных данных:** При вставке отсортированных данных время увеличилось с **0.12с** (1000 эл.) до **13.24с** (10000 эл.). Рост более чем в 100 раз при увеличении данных в 10 раз указывает на сложность $O(n^2)$ для заполнения. Дерево выродилось в список.

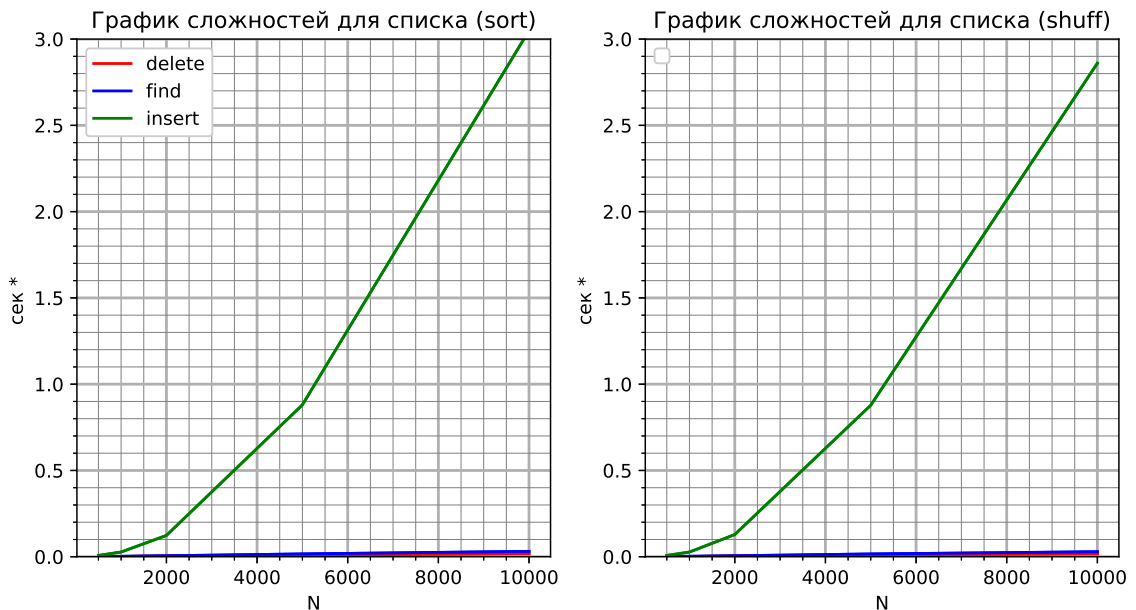
- **Эффективность на перемешанных данных:** На `shuffled` данных вставка 10000 элементов заняла всего **0.03с**. Это подтверждает логарифмическую сложность $O(\log n)$ для сбалансированного дерева.

2. Хеш-таблица: Стабильность и скорость



- **Чувствительность к порядку:** Хеш-таблица показала идентичные результаты как на `shuffled`, так и на `sorted` данных (около **0.16с** для 10000 вставок). Это объясняется тем, что хеш-функция распределяет ключи по бакетам независимо от их исходного порядка.
- **Превосходство:** На больших объемах хеш-таблица оказалась самой быстрой структурой для поиска и удаления ($\approx 0.001с$), что подтверждает среднюю сложность $O(1)$.
- **Замечание** так как таблица содержит списки со вставкой в конец, при вставке наблюдается отклонение от линейной зависимости в сторону квадратичной

3. Связный список: Стабильная медлительность



- **Поиск и удаление:** Связный список показал худшие результаты среди всех структур на случайных данных. Время поиска при 10000 элементах (**0.03с**) на два порядка медленнее, чем у хеш-таблицы. Это подтверждает линейную сложность $O(n)$.
- **Вставка:** Вставка в конец без указателя на хвост дает $O(n^2)$ при заполнении (**3.04с** на 10000 эл.), что сопоставимо с выродившимся деревом.

Вывод: выбор структуры данных

1. **Хеш-таблица** — лучший выбор для реальных задач «ключ-значение». Она обеспечивает стабильное $O(1)$ и не зависит от порядка входящих данных.
2. **BST** — эффективен только при условии случайного распределения данных или использовании самобалансирующихся деревьев. В противном случае велик риск деградации до скорости списка.

3. **Связный список** — в данной реализации неэффективен для поиска и массовой вставки. Его стоит использовать только для специфических задач (стеки, очереди), где работа идет преимущественно с головой списка за $O(1)$.