

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 серию опытов для N от 500 до 10000.

1. Бинарное дерево поиска (BST) и влияние порядка

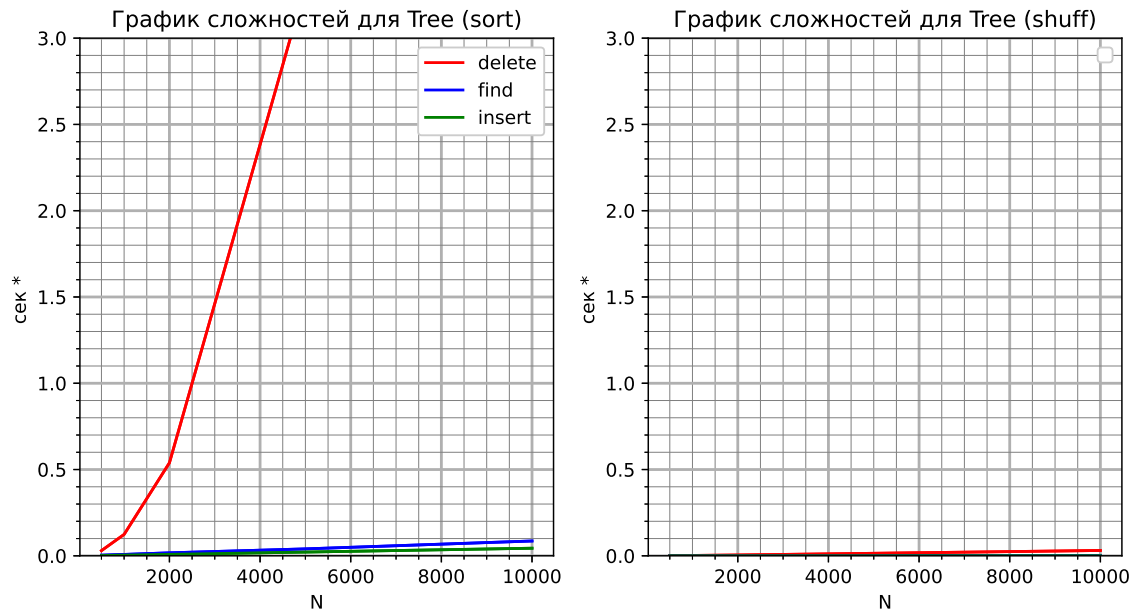
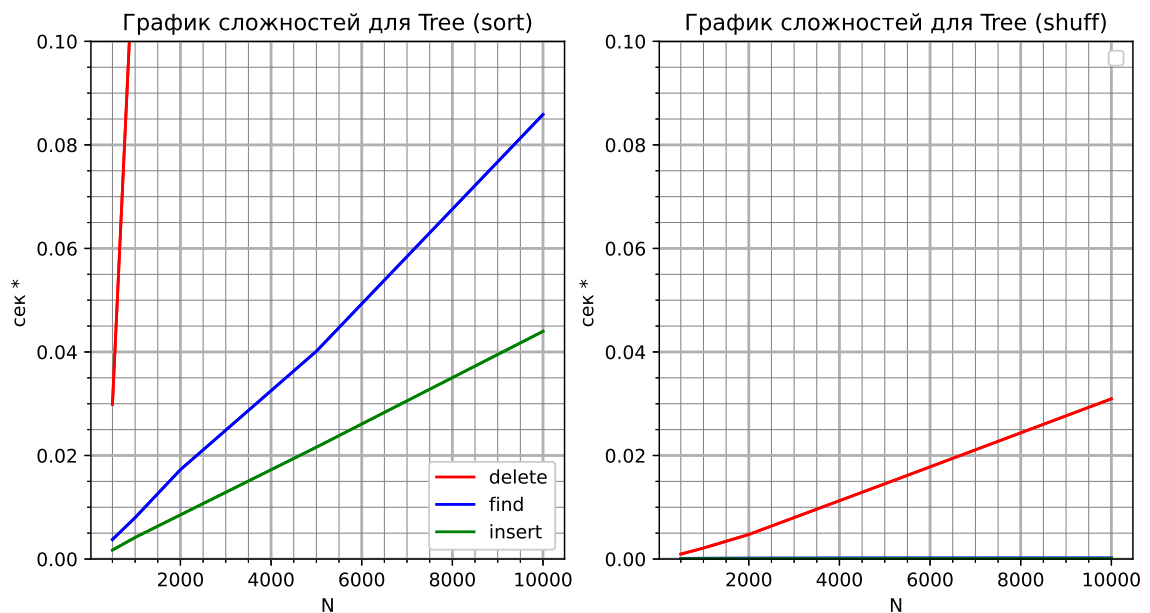


Рис. 1: Зависимость времени выполнения операций в BST от объема данных

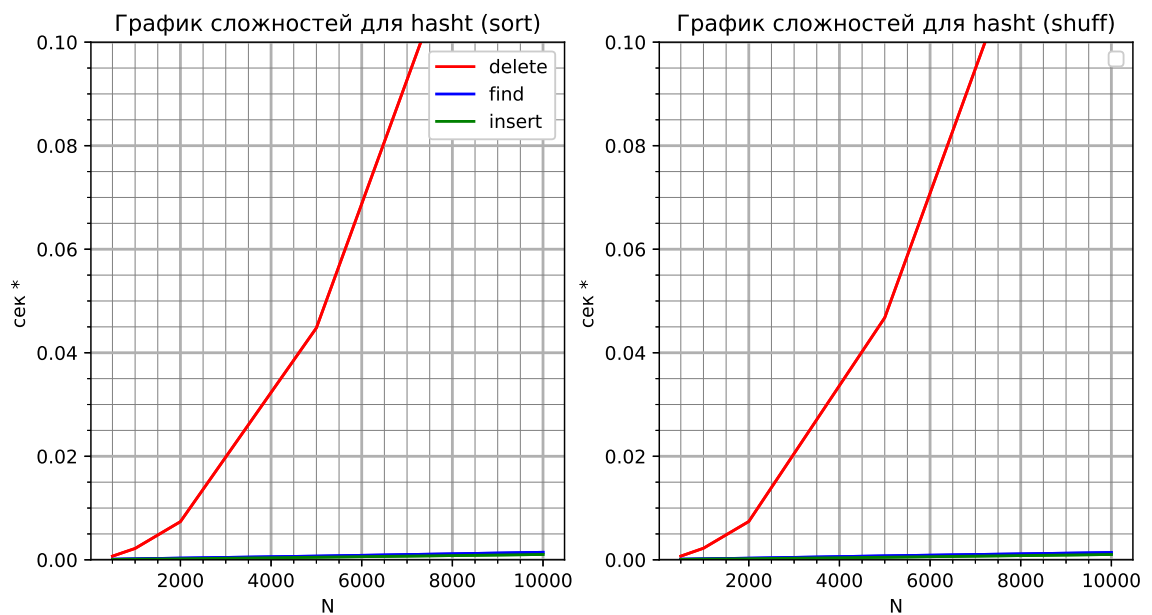


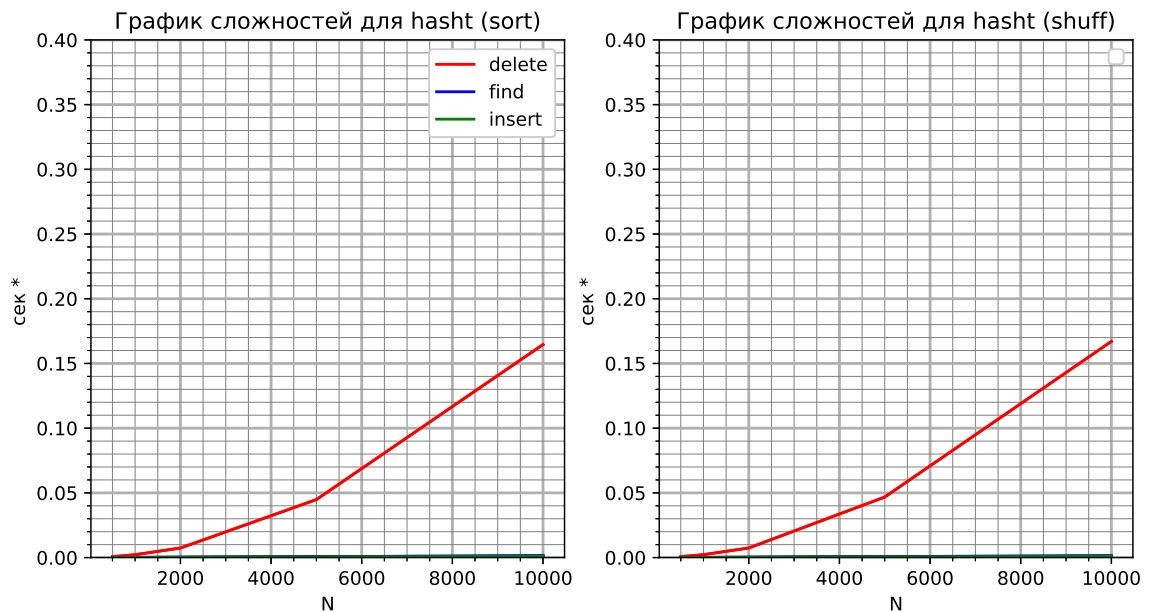
- **Деграция на отсортированных данных:** При вставке отсортированных данных время увеличилось с **0.124с** ($N = 1000$) до **13.27с** ($N = 10000$). Рост времени в 100 раз при увеличении объема данных в 10 раз

четко указывает на квадратичную сложность $O(n^2)$ для процесса заполнения всей структуры. Дерево выродилось в линейный список, и поиск места вставки стал занимать $O(n)$ вместо ожидаемого $O(\log n)$.

- **Эффективность на перемешанных данных:** На `shuffled` данных вставка 10000 элементов заняла всего **0.031с**. Это подтверждает логарифмическую сложность $O(\log n)$ для операций в дереве при случайном распределении ключей.

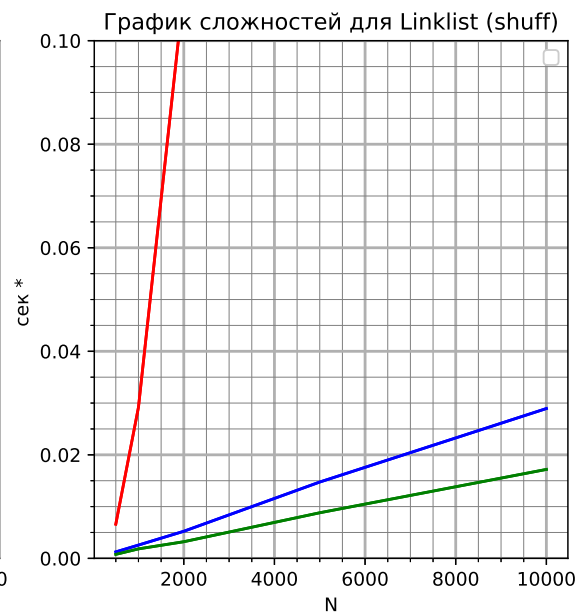
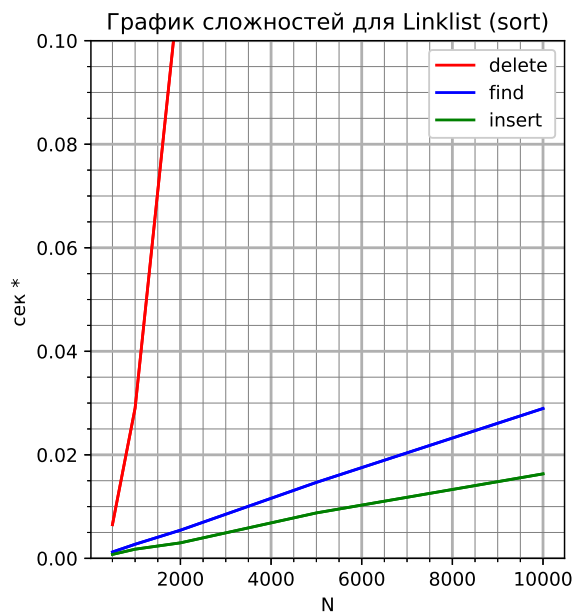
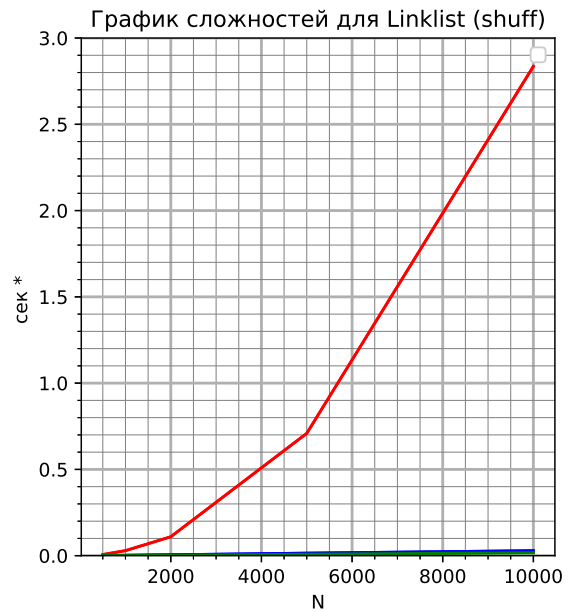
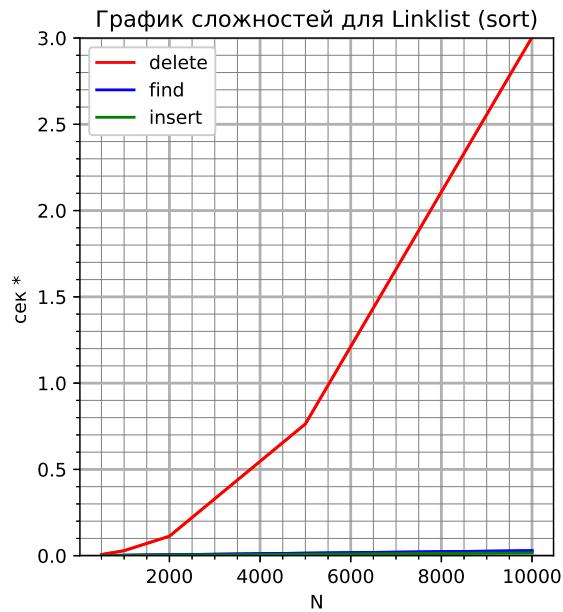
2. Хеш-таблица: Стабильность и скорость





- **Чувствительность к порядку:** Хеш-таблица показала идентичные результаты как на shuffled, так и на sorted данных (около **0.165с – 0.167с** для 10000 вставок). Это объясняется тем, что хеш-функция распределяет ключи по бакетам независимо от их исходного порядка, предотвращая деградацию структуры.
- **Превосходство:** На больших объемах хеш-таблица оказалась самой быстрой структурой для поиска и удаления ($\approx 0.001\text{с}$ при $N = 10000$), что подтверждает теоретическую среднюю сложность $O(1)$.
- **Замечание:** Так как реализация использует списки для разрешения коллизий со вставкой в конец, при заполнении таблицы наблюдается рост времени вставки, стремящийся к квадратичному, однако абсолютные значения остаются на порядки ниже, чем у выродившегося BST.

3. Связный список: Линейная зависимость



- **Поиск и удаление:** Связный список показал худшие результаты среди всех структур на случайных данных. Время поиска при 10000 элементах (**0.029с**) значительно медленнее, чем у BST на перемешанных данных (**0.0002с**). Это подтверждает линейную сложность $O(n)$.

- **Вставка:** Вставка (вероятно, в конец или с сохранением порядка) дает $O(n^2)$ при заполнении (**2.83с** – **3.00с** на 10000 эл.). Характер роста времени при переходе от $N = 5000$ (**0.71с**) к $N = 10000$ подтверждает квадратичную зависимость.

Вывод: выбор структуры данных

1. **Хеш-таблица** — наиболее универсальный выбор. Она обеспечивает стабильное $O(1)$ для поиска и не зависит от порядка входящих данных.
2. **BST** — крайне эффективен ($O(\log n)$) при случайном распределении данных, но без механизмов самобалансировки критически уязвим к отсортированным входным последовательностям, замедляясь до уровня списка.
3. **Связный список** — продемонстрировал самую низкую производительность на операциях поиска и массовой вставки. Его использование оправдано только в специфических сценариях (например, реализация стека), где работа ведется исключительно с головой списка за $O(1)$.

Сводная таблица результатов

Структура	Режим	Опер.	N=500	N=1000	N=5000	N=10000
LinkList	Shuffled	Insert	0.0066	0.0292	0.7089	2.8358
	Shuffled	Find	0.0012	0.0026	0.0147	0.0289
	Sorted	Insert	0.0065	0.0290	0.7637	3.0042
HashTable	Shuffled	Insert	0.0007	0.0022	0.0468	0.1670
	Shuffled	Find	0.0001	0.0002	0.0008	0.0014
	Sorted	Insert	0.0007	0.0022	0.0448	0.1646
BinTree	Shuffled	Insert	0.0009	0.0021	0.0145	0.0309
	Shuffled	Find	0.0001	0.0001	0.0002	0.0002
	Sorted	Insert	0.0298	0.1239	3.3052	13.2706

Таблица 1: Сравнение времени выполнения операций (в секундах) в зависимости от объема данных N