Отчет по сравнению CAVLTree, CHash, BinSearch.

Цель исследования:

Сравнить производительность CAVLTree, CHash, BinSearch при добавлении/сортировке и поиске элементов в структурах данных.

Методология:

Сгенерированы N структур Credentials(login, password), где login и password - длиной от 7 до 20 символов из русского, английского алфавитов верхнего и нижнего регистра и цифр. Каждая пара уникальная. Уникальность проверялась при генерировании (проверка на unordered_set).

Первый замер:

В случае CAVLTree и CHash был при поэлементном добавлении данных в данные структуры. В случае BinSearch при сортировке статического массива структур Credentials с помощью функции MergeSort.

Второй замер:

Поиск N элементов, находящихся в структурах.

Третий замер:

Поиск 2*N элементов.

Четвертый замер:

Для всех замер времени на освобождение ресурсов используемых алгоритмом, т.е. выполнение операции удаления массива, дерева или хеш-таблицы.

Исполняемые файлы лежат в папке
/isamidinova-EDSA/avltree_hash_binsearch_comparison/build/bin/
generate_data - генерирует данные
avltree_test - тестирует CAVLTree
binsearch_test - тестирует BinSearch
hash_test - тестирует CHash
все исполняемые данные по умолчанию работают на 10.000 элементах. Задать количество элементов можно передав число им в командной строке при запуске (в качестве argv[1]). В корневой папке есть скрипт run.sh, который собирает проект в режиме Release, запускает все тесты на 10.000, 100.000, 500.000, 1.000.000 элементах.

Результаты:

Измерения времени записаны в файл timer.txt в формате: по 3 замера для avltree_test, binsearch_test, hash_test на 10.000 элементах, далее также по 3 замера для каждого теста на 100.000 и 1.000.000 элементах. Для воспроизведения эксперимента можно зайти в папку avltree_hash_binsearch_comparison и запустить bash-скрипт run.sh. Он соберет проект, запустит все исполняемые файлы (генерация данных, avltree_test, binsearch_test, hash_test на 3 вариантах) и выполнит python-скрипт по генерации графика.

Анализ:

На момент составления отчета результаты в миллисекундах в режиме Debug:

CAVLTree	10.000	100.000	500.000	1.000.000
add	11	167	1413	3214
search N exist elements	5	90	847	2033
search 2*N random elements	11	231	2319	5355
delete	0	0	2	6

BinarySearch	10.000	100.000	500.000	1.000.000
add	4	61	481	1145
search N exist elements	4	57	395	684
search 2*N random elements	4	57	407	687
delete	0	0	0	0

CHash	10.000	100.000	500.000	1.000.000
add	3	174	8947	40320
search N exist elements	3	194	8991	40271
search 2*N random elements	8	998	39277	167588
delete	0	1	8	16

Ниже графики на основе этих данных:
добавление всех данных
поиск имеющихся элементов
поиск разных элементов (количество поисков в 2 раза больше данных)
освобождение ресурсов (операция clear)
Поэлементное удаление элементов для CAVLTree и CHash:
На момент составления отчета результаты в миллисекундах в режиме Release:

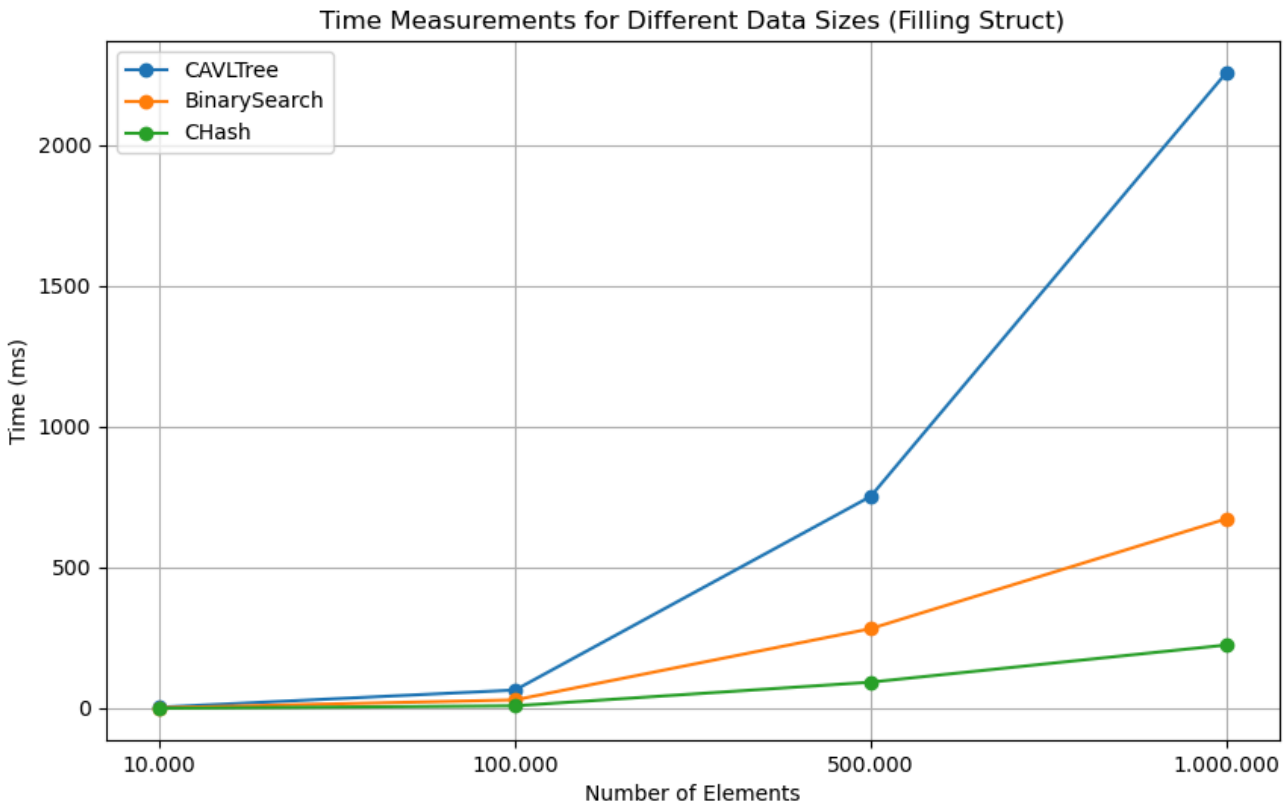
CAVLTree	10.000	100.000	500.000	1.000.000
add	5	92	960	2364
search N exist elements	2	64	706	1656
search 2*N random	5	166	1806	4068

elements				
delete	0	0	1	2

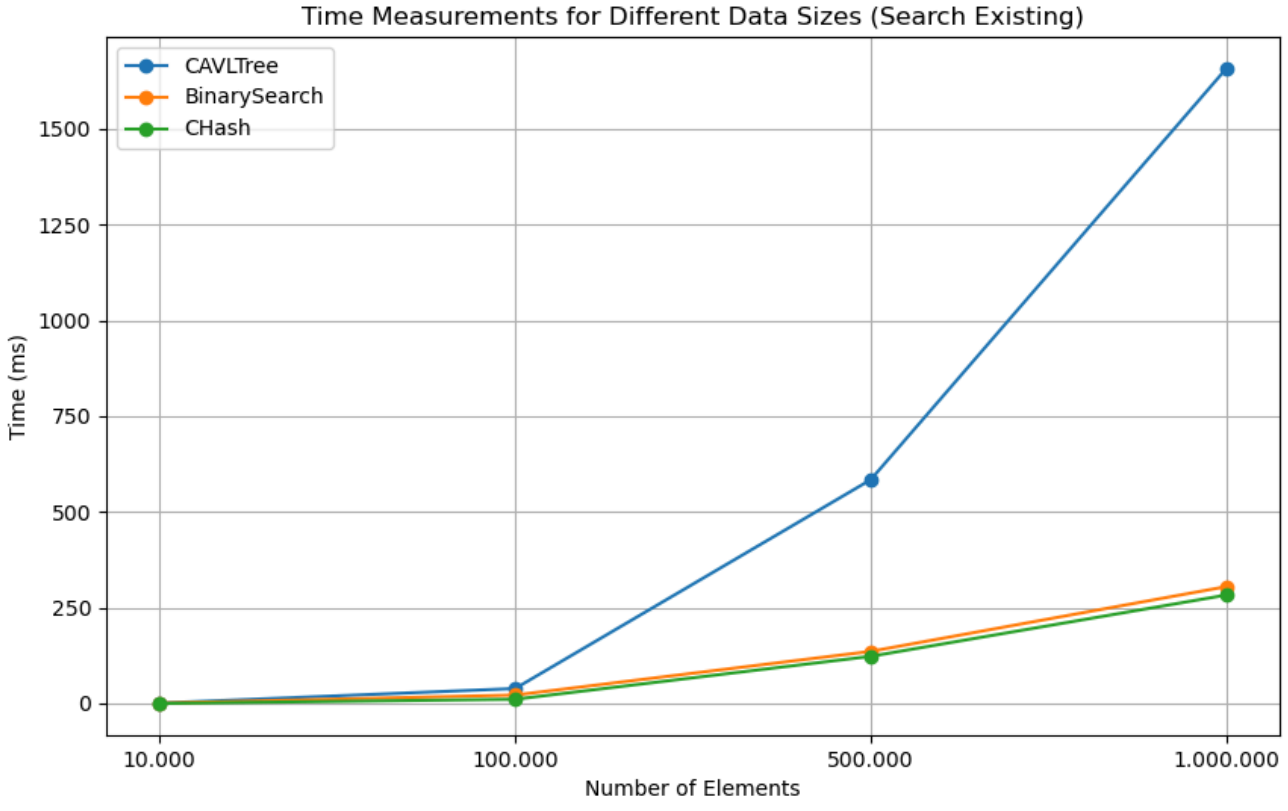
BinarySearch	10.000	100.000	500.000	1.000.000
add	2	44	313	800
search N exist elements	2	28	154	376
search 2*N random elements	2	25	176	350
delete	0	0	0	0

CHash	10.000	100.000	500.000	1.000.000
add	1	116	6844	29476
search N exist elements	1	159	6091	29683
search 2*N random elements	2	693	27199	124953
delete	0	0	3	6

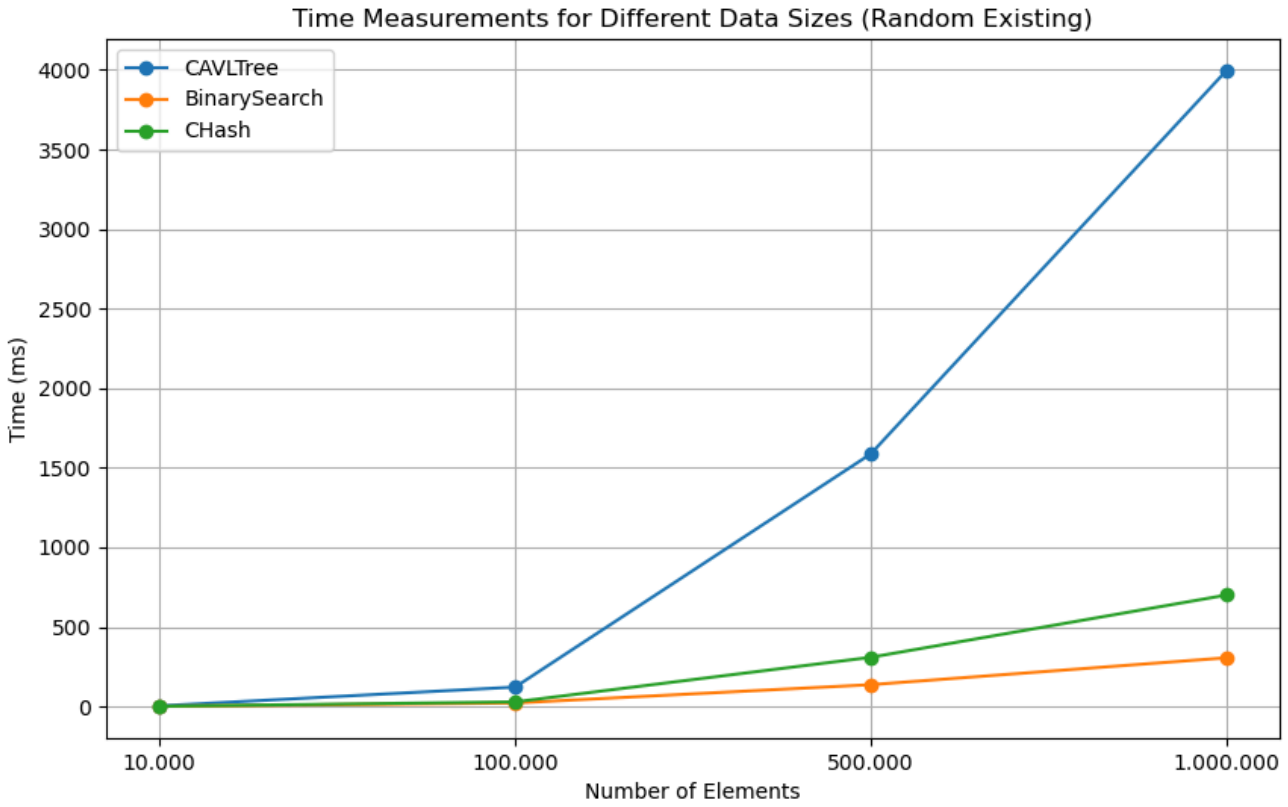
Ниже графики на основе этих данных:
добавление всех данных



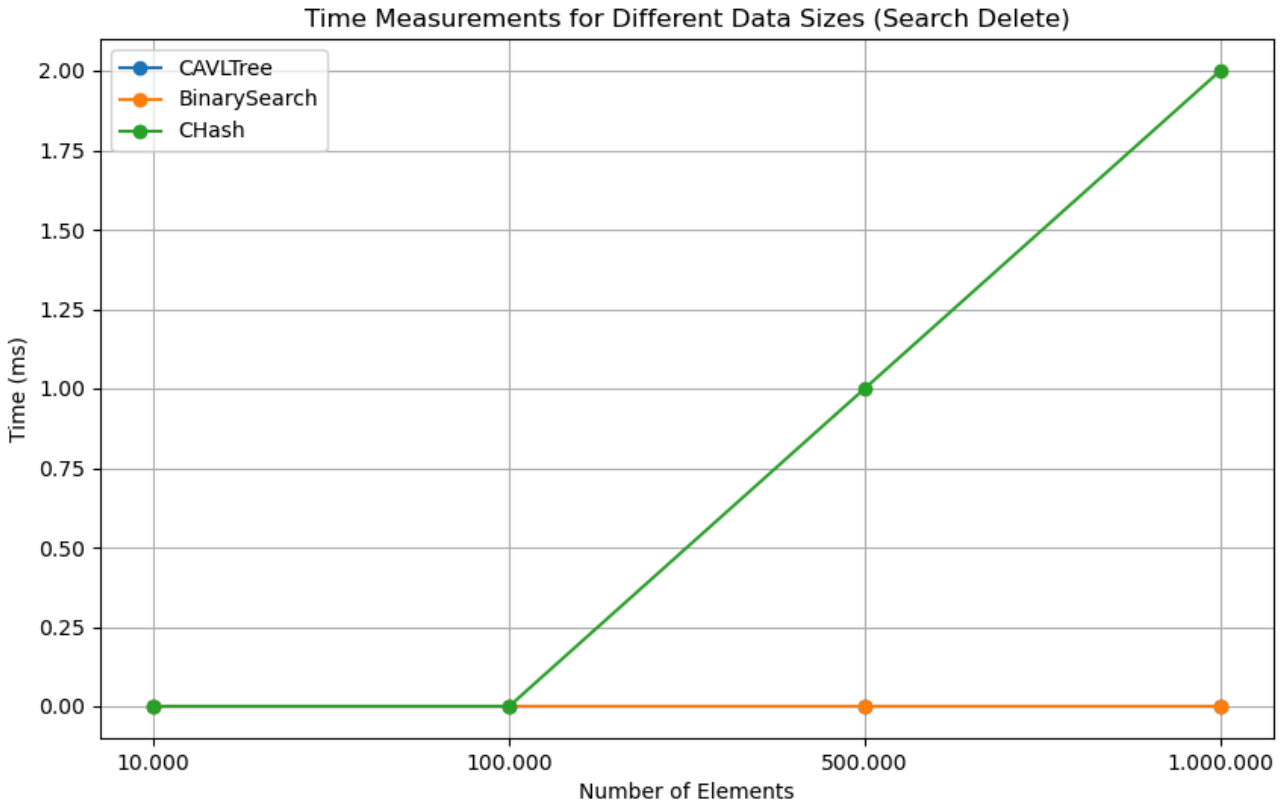
поиск имеющихся элементов



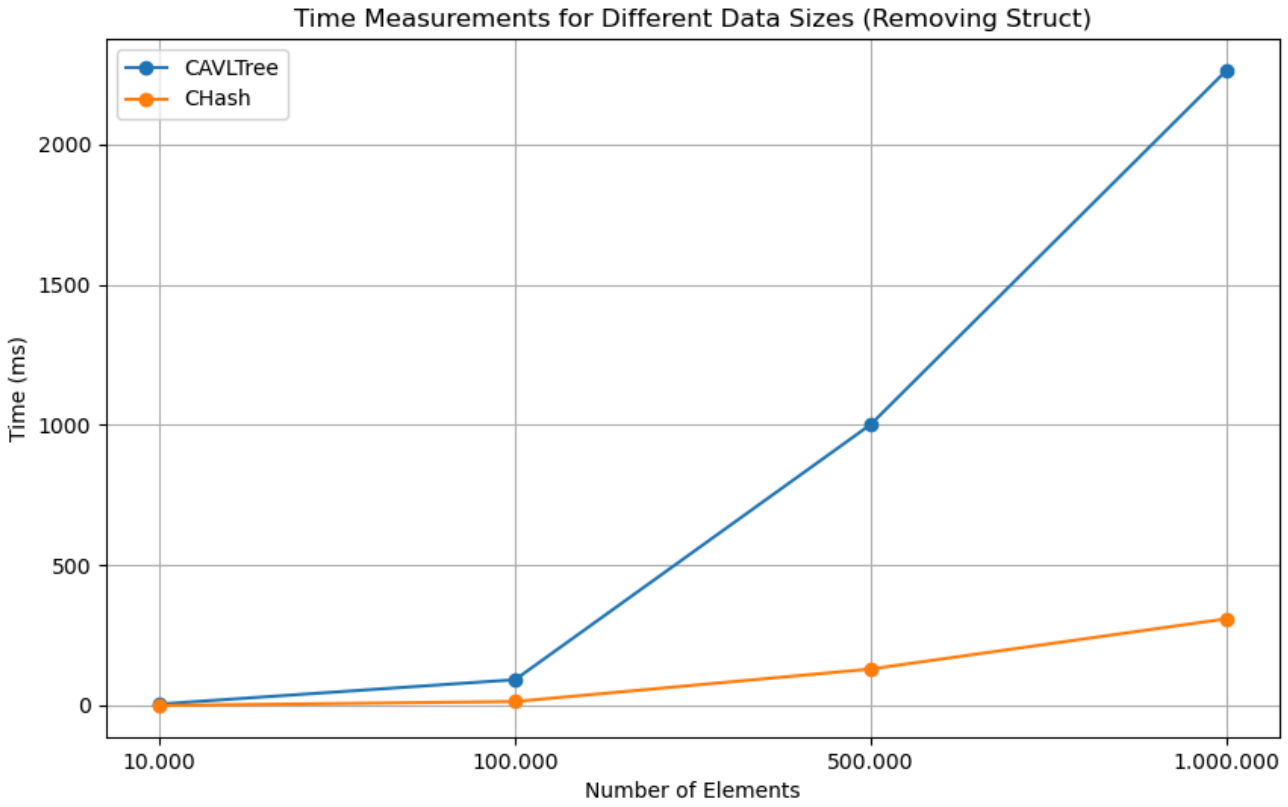
поиск разных элементов (количество поисков в 2 раза больше данных)



освобождение ресурсов (операция clear)



Поэлементное удаление элементов для CAVLTree и CHash:



Вывод:

Я тестировала 2 Хеш функции и большая часть времени в Хеш-таблице уходит на вычисление хеша, так как результаты сильно зависели от способа хеширования. В финальном варианте я оставила стандартную хеш-функцию Языка C++ (но мой вариант функции был быстрее, хоть и менее эффективной. первой функцией была просто сумма всех номеров сроков в строках). Также программа в режиме Release проработала быстрее режима Debug. Результаты показали, что бинарный поиск оказался быстрее, чем другие варианты хранения/поиска для таких нетривиальных структур, а CHash оказался самым неэффективным для данной задачи.