МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ
по курсовой работе по дисциплине
«Алгоритмы и структуры данных»
Тема: AVL-деревья vs Хэш-таблица(открытая адресация)

Студентка гр.0381 _____ Березовская В. В

Преподаватель _____ Берленко Т. А.

Санкт-Петербург
2021

Задание

На курсовую работу

Студентка Березовская В.В.

Группа 0381

Тема работы: АВЛ – дерево vs Хеш-таблица (открытая адресация)

Исходные данные:

- "Исследование" - реализация требуемых структур данных/алгоритмов; генерация входных данных (вид входных данных определяется студентом); использование входных данных для измерения количественных характеристик структур данных, алгоритмов, действий; сравнение экспериментальных результатов с теоретическими. Вывод промежуточных данных не является строго обязательным, но должна быть возможность убедиться в корректности алгоритмов.

Содержание пояснительной записки:

- Содержание
- Введение
- Выполнение работы
- Список используемых источников.
-

Предполагаемый объем пояснительной записки:

Не менее 15 страниц.

Дата выдачи задания: 18.10.2021

Дата сдачи реферата: 22.12.2021

Дата защиты реферата: 23.12. 21

Студентка гр.0381	_____	Березовская В. В
Преподаватель	_____	Берленко Т. А.

Аннотация

Курсовая работа представляет собой реализацию АВЛ-дерева и Хеш-таблицы с методом открытой адресации в качестве способа разрешения коллизий для исследования этих структур относительно теоритических результатов и друг друга.

Операции в двух структурах данных: поиск, вставка, удаление элемента, а также отображаюся графики зависимости среднего времени операции от количества элементов.

Summary

This course work demonstrates a realisation of two different data structures: an AVL-Tree and a Hash table with open addressing in order to solve the problem of collision during the next theoretical research.

Operations in both data structures are search, insertion and deletion of any element from output data. Moreover, the program draws flow charts, which show the dependency between time and the amount of input data.

Содержание

Введение

Цель работы:

Провести исследование двух структур данных: АВЛ-дерева и Хеш таблицы в качестве метода разрешения коллизий.

Пункты, позволяющие выполнить это задание:

- Изучение материла по данной теме
- Реализация структур данных.
- Измерение результатов работы и проверка на корректность.

1. Теоретические положения

АВЛ-дерево

АВЛ-дерево — сбалансированное по высоте двоичное дерево поиска: для каждой его вершины высота её двух поддеревьев различается не более чем на 1. АВЛ — аббревиатура, образованная первыми буквами создателей (советских учёных) Адельсон-Вельского Георгия Максимовича и Ландиса Евгения Михайловича.

Благодаря основному свойству (высота <=1), скорость работы всех операций с деревом занимает O(log(n)). Это отличает АВЛ дерево от простого бинарного дерева поиска, ведь если не будет балансировки, то у нас может получиться такая ситуация, как линейный список, тогда сложность будет уже O(n).

Балансировка достигается при помощи поворотов – замены ролей потомков и предков. Есть 4 вида поворотов в АВЛ дереве:

1)Левый малый - , когда h(b)-h(L) = 2, h(C)<=h(R)

2)Левый большой - , когда h(b) – h(L) = 2, h(c) > h(R)

3)Правый малый - , когда h(b) – h(R) = 2, h(C)<=h(L)

4)Правый большой - , когда h(b) – h(R) = 2, h(C)>h(L)

Можно с уверенностью сказать, что эти операции уменьшают полную высоту дерева на 1, при этом они не могу увеличить общую высоту.

Хеш-таблица (открытая адресация)

Хеш-таблица — это структура данных, реализующая интерфейс ассоциативного массива, а именно, она позволяет хранить пары (ключ, значение) и выполнять три операции: операцию добавления новой пары, операцию поиска и операцию удаления пары по ключу.

В массиве H хранятся сами пары ключ-значение. Алгоритм вставки элемента проверяет ячейки массива H в некотором порядке до тех пор, пока не будет найдена первая свободная ячейка, в которую и будет записан новый элемент. Этот порядок вычисляется на лету, что позволяет сэкономить на памяти для указателей, требующихся в хеш-таблицах с цепочками.

Последовательность, в которой просматриваются ячейки хеш-таблицы, называется последовательностью проб. В общем случае, она зависит только от ключа элемента, то есть это последовательность h0(x), h1(x), …, hn — 1(x), где x — ключ элемента, а hi(x) — произвольные функции, сопоставляющие каждому ключу ячейку в хеш-таблице. Первый элемент в последовательности, как правило, равен значению некоторой хеш-функции от ключа, а остальные считаются от него. Для успешной работы алгоритмов поиска последовательность проб должна быть такой, чтобы все ячейки хеш-таблицы оказались просмотренными ровно по одному разу.

Алгоритм поиска просматривает ячейки хеш-таблицы в том же самом порядке, что и при вставке, до тех пор, пока не найдется либо элемент с искомым ключом, либо свободная ячейка (что означает отсутствие элемента в хеш-таблице).

Удаление элементов в такой схеме несколько затруднено. Обычно поступают так: заводят булевый флаг для каждой ячейки, помечающий, удален элемент в ней или нет. Тогда удаление элемента состоит в установке этого флага для соответствующей ячейки хеш-таблицы, но при этом необходимо модифицировать процедуру поиска существующего элемента так, чтобы она считала удалённые ячейки занятыми, а процедуру добавления — чтобы она их считала свободными и сбрасывала значение флага при добавлении.

2. Выполнение работы.

1. Хеш-таблица.

Реализация хеш-таблицы представлена в классе HashTable файла hash.py. Как было оговорено ранее, хеш-таблица – это массив, где в каждой ячейке лежит ключ и значение. За этот массив отвечает переменная table, в ней хранится ключ и значение. Так же инцилизируется рамер таблицы. Реализована функция hashing, которая получает короткий ключ путём вычисления по модулю размера таблицы.

Функция добавления элемента принимает на вход ключ и значение, после чего проверяет не переполнена ли таблица. Если количество в массиве больше, чем половина его длины**,** то вызывается метод расширения таблицы вдвое. Затем вычисляется хэш-ключ и проверяется, если по этому значению, клетка пустая, то мы вносим наши данные, иначе линейным опробированием ищется пустая ячейка, в которую мы можем записать данные.

Функция удаления после получения хэш-ключа проверяет совпадает ли изначальный ключ и значение с тем, что записано в массиве по этому адресу. В случае отсутствие коллизии, элемент заменяется на None, иначе индекс увеличивается, пока не найдём.

Функция поиска реализована по такому же принципу.

2. АВЛ-дерево.

АВЛ-дерево – вторая структура данных, которую требовалось реализовать. Она хранится в файле AVL_tree.py. Так как это дерево, у него должен быть лист, ну или вершина Node, в ней хранятся указатели на детей, значение и высота узла. В класс AVLTree, в котором реализованы функции добавления, поиска и удаления элемента.

Функция insert запускает рекурсивные вызовы от левого и правого потомка. Перед тем как непосредственно добавить вершину, обновляется высота. Если

баланс дерева нарушается, то с помощью малых правых и левых поворотов он восстанавливается. Аналогично работает удаление вершины.

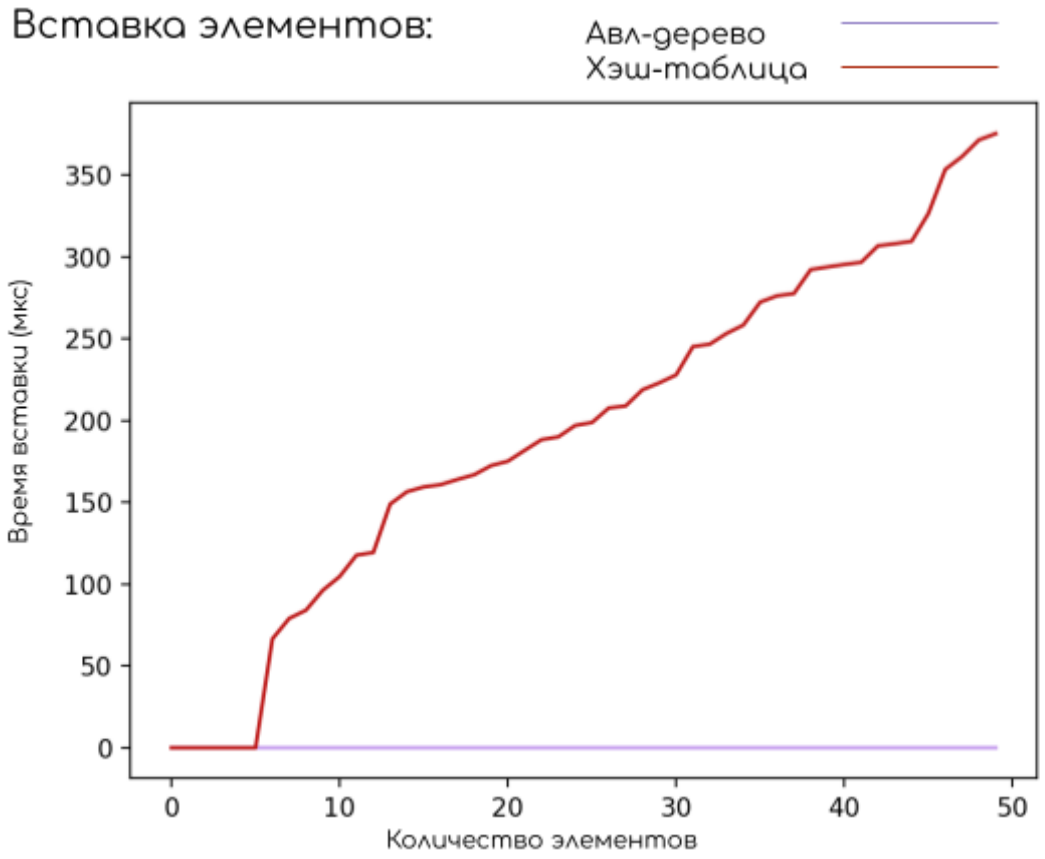
Функция поиска просто проверяет значение и, как в бинарном поиске, спускается по дереву в зависимости велечины искомой вершины. Реализована функция вывода дерева в формате предпорядка.

3. **Файл запуска** – main.cpp. В нем создаются вызываются методы двух структур и стоятся графики зависимости времени, от количества элементов.

3. Анализ результатов

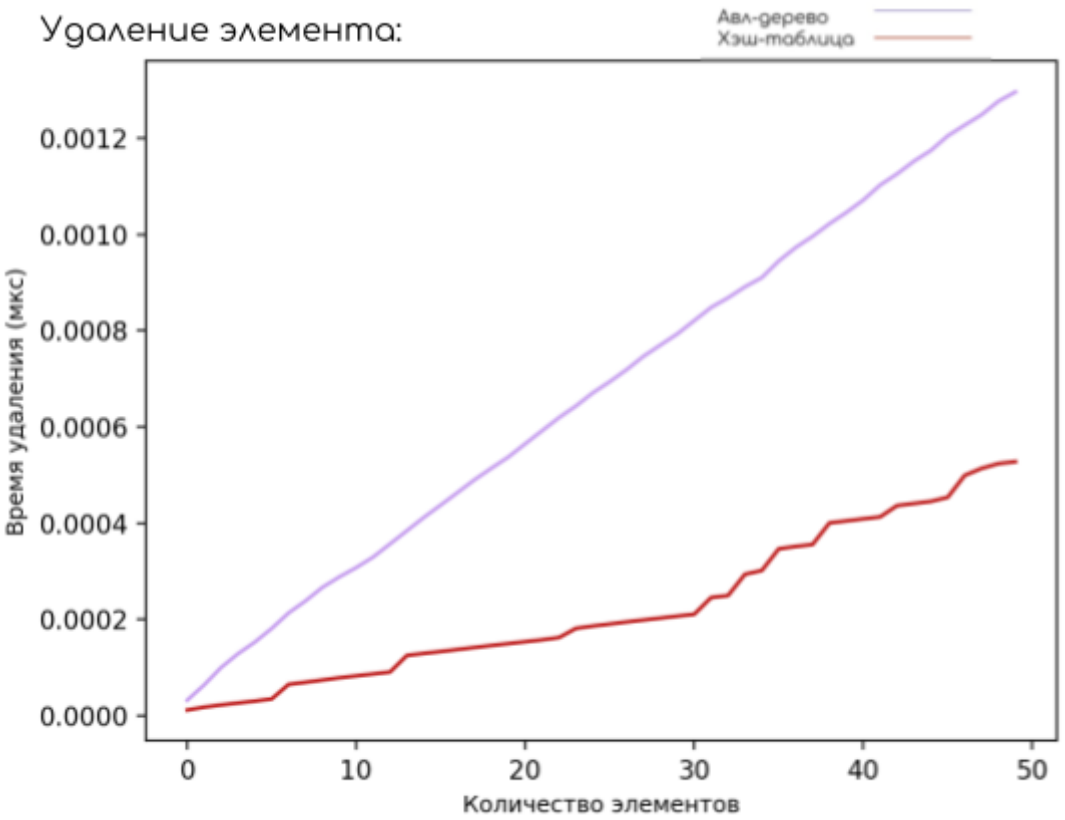
1) Вставка элементов. В результате добавление сотни рандомных чисел в обе структуры данных выяснилось, что АВЛ-дерево ведет себя стабильнее, чем хеш таблица. Красным показан график хэш-таблицы, как раз в худшем случае O(n), фиолетовым график дерева, в среднем и худшем случае случае O(log n).

В итоге получается, если в теории АВЛ дерево медленнее, чем хеш-таблица во всех случаях, так в лучшем случае у таблицы будет костантное время, то на практике оказывается, что их скорость добавления элементов примерно одинаковая, за исключением, времени перевыделения памяти у хеш-таблицы.

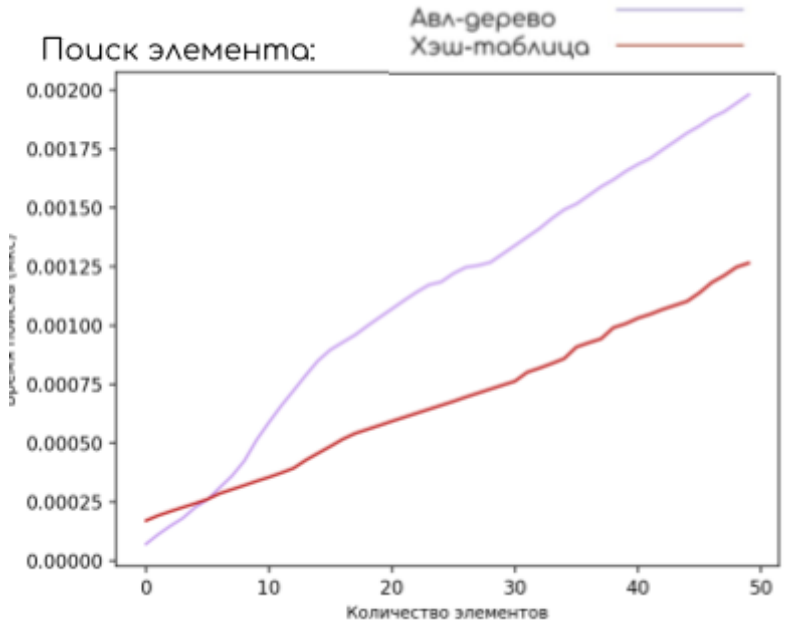


2) Удаление элементов. На все тех же сотне рандомных числах выяснилось, что удаление, все же, быстрее проходит у хеш-таблицы, чем у АВЛ дерева. Скорее всего подобное отставание может быть вызвано большим перебалансировок

дерева после каждого удаления элемента.



В теории сказано, что АВЛ дерево имеет стабильную скорость, что и выясняется из графика, а вот у хеш-таблицы, наоборот, слишком большой разброс от константного времени до O(n). Это и выясняется из графика, что АВЛ дерево на дистанции быстрее работает.

3) 

Поиск. Здесь хеш-таблица работает лучше, чем АВЛ дерево. Это наглядно видно из графика:

Исходя из теоретических данных, АВЛ дерево должно быть медленнее, чем

хеш-таблица, но только не в худшем случае. На практике это свойство так же соблюдается.

4) Память. Последний пункт сравнения этих двух структур. В теории говорится, что у них память O(n), однако на практике, как мы можем убедиться из графика, хеш-таблица (красная линия) занимает больше памяти, чем АВЛ дерево(фиолетовая линия).

Заключение

Была написана программа для исследования двух различных структур данных, таких как АВЛ-дерево и хеш-таблица.После исследования можно сделать вывод, что АВЛ дерево на большой дистанции более оптимальный вариант, нежели хеш-таблица, однако для малого количества данных таблица будет быстрее и эффективнее, чем АВЛ дерево.

Список использованной литературы:

1) https://ru.wikipedia.org/wiki/АВЛ-дерево

2) https://www.cs.usfca.edu/~galles/visualization/AVLtree.html

3) https://www.notion.so/9ff1a5c4c16642bca244570373f12edb#8992f86211ba46b795e734aac779164f

4) https://intuit.ru/studies/courses/648/504/lecture/11469?page=2

5) https://www.cs.usfca.edu/~galles/visualization/Hash.html

6) Алгоритмы: построение и анализ (Кормен Т., Лезерсон Ч., Ривест Р., Штайн К.)

7) Алгоритмы и структуры данных (Вирт Н.)

8) https://ru.wikipedia.org/wiki/Хеш-таблица

9) https://neerc.ifmo.ru/wiki/index.php?title=Разрешение_коллизий