**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

Курсовая РАБОТА

**по дисциплине «Алгоритмы и структуры данных»**

Тема: Хеш-таблица (двойное хеширование) vs Хеш-таблица (открытая адресация). Исследование

**Вариант 6**

|  |  |  |
| --- | --- | --- |
| Студентка гр. 1381 |  | Рымарь М.И. |
| Преподаватель |  | Иванов Д.В. |

Санкт-Петербург

2022

**ЗАДАНИЕ**

**на курсовую работу**

|  |  |  |
| --- | --- | --- |
| Студентка Рымарь Мария Игоревна | | |
| Группа 1381  Вариант 6 | | |
| Тема работы: Хеш-таблица (двойное хеширование) vs Хеш-таблица (открытая адресация). Исследование | | |
| Задание:  "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. | | |
|  | | |
| Предполагаемый объем пояснительной записки:  Не менее 15 страниц. | | |
| Дата выдачи задания: 25.10.2022 | | |
| Дата сдачи реферата: 24.12.2022 | | |
| Дата защиты реферата: 24.12.2022 | | |
| Студентка |  | Рымарь М.И. |
| Преподаватель |  | Иванов Д.В. |

**Аннотация**

В ходе выполнения курсовой работы создана программа на языке программирования Python, в которой реализованы таких структуры данных, как Хеш-таблица с открытой адресацией и Хеш-таблица с двойным хешированием. Структуры данных сравниваются по теоретической сложности базовых операций, также сравниваются полученные результаты с экспериментальными значениями.

**Summary**

During the course work, a program was created in the Python programming language, which implements such data structures as a hash table with open addressing and a hash table with double hashing. The data structures are compared according to the theoretical complexity of the basic operations, and the results obtained are also compared with experimental values.**содержание**

|  |  |  |
| --- | --- | --- |
| 1. | Введение | 5 |
| 2. | Теоретические сведения | 6 |
| 3. | Ход выполнения работы | 10 |
| 3.1. | Реализация структур данных | 10 |
| 3.1.1.  3.1.2.  3.2.  3.3. | Хеш-таблица с открытой адресацией  Хеш-таблица с двойным хешированием  Теоретическая оценка сложности базовых операций  Сравнение с экспериментальными значениями | 11  12  12  10 |
| 4. | Заключение | 16 |
| 5. | Список использованных источников | 17 |
| 6. | Приложение А | 18 |
|  |  |  |
|  |  |  |
|  |  |  |

**введение**

Требуется написать программу, которая производит сравнение двух структур данных.

**Цель и задачи работы**

Цель: исследовать такие структуры данных, как Хеш-таблица с открытой адресацией и Хеш-таблица с двойной адресацией.

Задачи:

* Реализовать структуры данных;
* Осуществить генерацию входных данных, вид входных данных определить самостоятельно;
* Произвести теоретическую оценку сложности базовых операций;
* Сравнить значения, полученные в ходе работы программ, с теоретическими.

**2. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

**2.1. Хеш-таблица**

Хеш-таблицей называется структура данных, обеспечивающая очень быструю вставку и поиск. На первый взгляд звучит слишком хорошо, чтобы быть правдой: независимо от количества элементов данных вставка и поиск (а иногда и удаление) выполняются за время, близкое к постоянному: O(1) в O-синтаксисе. На практике это лишь несколько машинных команд.

Для пользователя хеш-таблицы обращение к данным происходит практически мгновенно. Все делается настолько быстро, что компьютерные программы часто используют хеш-таблицы при необходимости сделать выборку из десятков тысяч элементов менее чем за секунду (как, например, в системах проверки орфографии).

Хеш-таблицы по скорости значительно превосходят деревья, которые выполняют операции за с относительно малое время O(logN). Операции с хеш-таблицами не только быстро выполняются, но и относительно просто программируются.

У хеш-таблиц также имеются свои недостатки. Они реализуются на базе массивов, а массивы трудно расширить после создания. У некоторых разновидностей хеш-таблиц быстродействие катастрофически падает при заполнении таблицы, поэтому программист должен довольно точно представлять, сколько элементов данных будет храниться в таблице (или приготовиться к периодическому перемещению данных в другую хеш-таблицу большего размера — процесс занимает довольно много времени).

Кроме того, при работе с хеш-таблицами не существует удобного способа перебора элементов в определенном порядке (скажем, от меньших к большим). Если необходима такая возможность, стоит выбрать другую структуру данных.

Но если нет необходимости перебирать элементы в определенном порядке, а размер базы данных можно спрогнозировать заранее, хеш-таблицы не имеют себе равных по скорости и удобству.

**2.2. Хеш-таблица с открытой адресацией**

Если элемент данных не удается разместить в ячейке с индексом, вычисленным посредством хеш-функции, метод открытой адресации ищет в массиве другую ячейку. Существует три основных разновидности открытой адресации, различающихся способом поиска следующей свободной ячейки: линейное пробирование, квадратичное пробирование и двойное хеширование.

Линейное пробирование и квадратичное пробирование будет рассмотрено в этом разделе, двойное хеширование в следующем, так как в ходе реализации курсовой работы требуется реализовать и исследовать хеш-таблицу с открытой адресацией и хеш-таблицу с двойным хешированием двумя отдельными структурами данных.

Алгоритм линейного пробирования последовательно ищет пустые ячейки. Если при попытке вставки элемента выясняется, что ячейка 5421 занята, мы переходим к ячейке 5422, затем к ячейке 5423 и т. д. Индекс последовательно увеличивается до тех пор, пока не будет найдена пустая ячейка. Процедура поиска называется «линейным пробированием», потому что она основана на линейной проверке последовательности ячеек.

При открытой адресации с линейным пробированием возникает проблема группировки. Образовавшиеся группы начинают расширяться. Элементы, хешируемые в пределах группы, добавляются в конец группы, в результате чего группа становится еще больше. Чем больше размер группы, тем быстрее она растет.

Отношение количества элементов в таблице к размеру таблицы называется коэффициентом заполнения. Таблица с 10 000 ячеек, содержащая 6667 элементов, имеет коэффициент заполнения 2/3.

коэффициент\_заполнения = количество\_элементов / размер\_массива;

Группы могут образовываться даже при относительно небольшом коэффициенте заполнения. Одни части хеш-таблицы могут быть заполнены большими группами, другие почти не содержать элементов. Группировка снижает быстродействие таблицы. Квадратичное пробирование пытается избежать образования групп. Его идея заключается в том, чтобы проверять ячейки, находящиеся на больших расстояниях (вместо ячеек, находящихся вблизи от исходной позиции хеширования).

**2.3. Хеш-таблица с двойным хешированием**

Для устранения как первичной, так и вторичной группировки применяется алгоритм двойного хеширования. Вторичная группировка возникает из-за того, что алгоритм, генерирующий последовательность смещений для квадратичного пробирования, всегда генерирует одни и те же смещения: 1, 4, 9, 16 и т. д.

В идеале последовательность проб должна генерироваться в зависимости от ключа (вместо использования набора одинаковых смещений для всех ключей). В этом случае числа с разными ключами, хешируемые в один индекс, будут использовать разные последовательности смещений.

Задача решается повторным хешированием ключа с другой хеш-функцией и использованием результата в качестве смещения. Для заданного ключа размер смещения остается постоянным при пробировании, но для разных ключей используются разные размеры. Практический опыт показал, что вторичная хеш-функция должна обладать некоторыми характеристиками:

- Она не должна совпадать с первичной хеш-функцией.

- Ее результат никогда не должен быть равен 0 (в противном случае смещения не будет, все пробы будут приходиться на одну ячейку, а алгоритм войдет в бесконечный цикл).

Эксперты обнаружили, что для решения этой задачи хорошо подходят функции вида смещение = константа - (ключ % константа); где константа — простое число, меньшее размера массива. Пример функции: stepSize = 5 - (key % 5);

Разные ключи могут хешироваться в один индекс, но для них (вероятно) будут сгенерированы разные смещения. При такой хеш-функции размеры смещений лежат в диапазоне от 1 до 5. На рисунке 1 под буквой а представлен успешный поиск элемента, под буквой б представлен безуспешный поиск элемента.

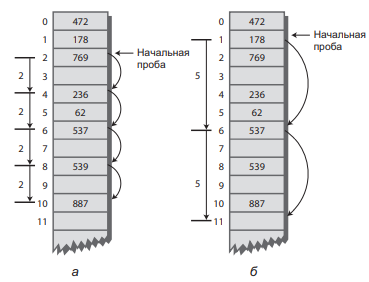
****

Рисунок 1 – Поиск элемента с помощью двойного хеширования

**3. ХОД ВЫПОЛНЕНИЯ РАБОТЫ**

**3.1 Реализация структур данных**

**3.1.1. Хеш-таблица с открытой адресацией**

Для реализации хеш-таблицы с открытой адресацией создан класс OpenHash на основе метода открытой адресации. Во время создания объекта класса требуется указать размер таблицы prime. В функции хеширования hash\_function происходит деление на простое число в случае, если введённый размер таблицы не является простым. Происходит приведение числа, равного размеру таблицы, к простому числу.

В классе реализованы методы вставки, поиска и удаления.

- insert\_data(key, data, i) - метод вставки значения в таблицу. Функция вычисляет хеш значение от ключа key и вставляет пару key, data в ячейку с индексом, равным (hash\_function(key) + i) % num. Этот индекс находится с помощью линейного пробирования. Если по данному хешу уже есть данные с другим ключом, то происходит поиск свободной ячейки, следующая по индексу. Функция вызывается рекурсивно. Если коэффициент заполнения больше 2/3, то происходит расширение таблицы, меняется местоположения всех ключей и данных.

- search\_elem(key) – метод поиска значения по соответствующему ключу key. Функция вычисляет хеш значение от ключа key, проверяет, есть ли элемент под индексом hash\_function(key), проверяет совпадение ключа и возвращает его. В случае несовпадения функция вызывается рекурсивно, i увеличивается на 1. Если элемента нет, то функция сообщает об этом.

- remove\_data(key) – метод удаления данных с ключом key. Функция выполняет аналогичные действия методу search\_elem. Однако вместо того, чтобы возвращать элемент, заменяет его на строку delete и ничего не возвращает.

Пример визуализации таблицы см. в Приложении А (открытая адресация).

**3.1.2. Хеш-таблица с двойным хешированием**

Для реализации хеш-таблицы с двойным хешированием для разрешения коллизии создан класс DoubleHash. Во время создания объекта класса требуется указать размер таблицы prime. В основной функции хеширования hash\_function происходит деление на простое число в случае, если введённый размер таблицы не является простым. Происходит приведение числа, равного размеру таблицы, к простому числу. Дополнительная хеш функция off\_hash\_function умножает полученное число на некоторую константу, заданную при инициализации, которая является смещением (константа меньше единицы и больше нуля), после чего полученное число умножается на размер таблицы, который приводится к простому числу, если не является им.

В классе реализованы методы вставки, поиска и удаления.

- insert\_data(key, data, i) - метод вставки значения в таблицу. Функция вычисляет хеш значение от ключа key и вставляет пару key, data в ячейку с индексом, равным (hash\_function(key) + i \* off\_hash\_function(key)) % num. Это является двойным хешированием. Если по данному хешу уже есть данные с другим ключом, то происходит поиск свободной ячейки, следующая по индексу. Функция вызывается рекурсивно. Если коэффициент заполнения больше 2/3, то происходит расширение таблицы, меняется местоположения всех ключей и данных.

- search\_elem(key) – метод поиска значения по соответствующему ключу key. Функция вычисляет хеш значение от ключа key, проверяет, есть ли элемент под индексом hash\_function(key), проверяет совпадение ключа и возвращает его. В случае несовпадения функция вызывается рекурсивно, i увеличивается на 1. Если элемента нет, то функция сообщает об этом.

- remove\_data(key) – метод удаления данных с ключом key. Функция выполняет аналогичные действия методу search\_elem. Однако вместо того, чтобы возвращать элемент, заменяет его на строку delete и ничего не возвращает.

Пример визуализации таблицы см. в Приложении А (двойное хеширование).

**3.2. Теоретическая оценка сложности базовых операций**

Теоретическая сложность базовых операций в хеш таблице представлена в таблице 1.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Лучший случай | Средний случай | Худший случай |
| Поиск | О(1) | О(а) | О(n) |
| Удаление | О(1) | О(а) | О(n) |
| Вставка | О(1) | О(а) | О(n)\* |

Таблица 1 – Теоретическая сложность базовых операций

\* - О(n) в случае расширения таблицы или при вставке с одним и тем же индексом при достижении или превышении коэффициента заполняемости таблицы. Придётся расширить таблицу, что займёт О(n).

а – количество повторений получения одного и того же хеша для различных ключей, то есть то, сколько раз приходилось разрешать коллизию.

Можно сделать вывод о сложности базовых операций для хеш таблиц: все они в среднем выполняются за О(1), однако при этом не гарантируется, что время выполнения отдельной операции мало.

**3.3. Сравнение с экспериментальными значениями**

Хеш-таблица с открытой адресацией

Худший случай представлен на рисунке 2.

Вставка: Чтобы проверить сложность вставки, вставляем n элементов и засекаем время каждой вставки. Ключами являются простые числа, взятые от текущего числа элементов. Зависимость действительно похожа на O(n).

Поиск: Для демонстрации худшего случая ищем элементы по одному и тому же хешу. По графику похоже на O(n).

Удаление: Удаляем элементы, выбранные в поиске, что является худшим случаем. Результат похож на O(n).

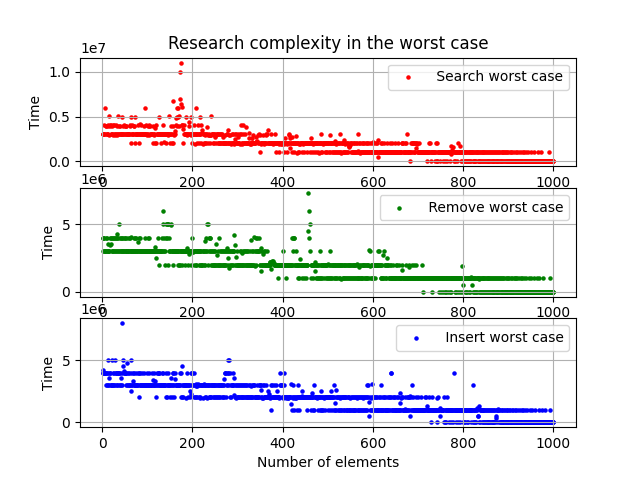


Рисунок 2 – Худший случай для открытой адресации

Средний случай представлен на рисунке 3.

Вставка: Заполнение таблицы случайными числами. Засекаем время вставки каждого.

Поиск: Ищем элементы в таблице с ключами, начиная от нулевого и дальше в цикле. По графику видно, что время зависит от количества повторений.

Удаление: Происходит аналогично поиску.

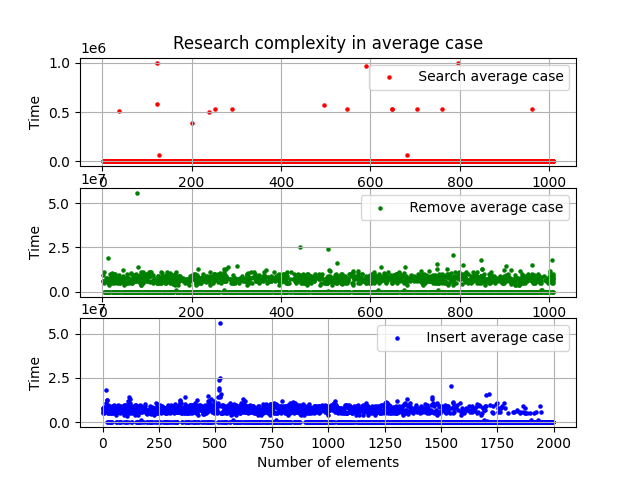


Рисунок 3 – Средний случай для открытой адресации

Хеш-таблица с двойным хешированием

Худший случай представлен на рисунке 4.

Вставка: Чтобы проверить сложность вставки, вставляем n элементов и засекаем время каждой вставки. Ключами являются простые числа, взятые от текущего числа элементов и изменяем его в зависимости от константы, при помощи которой получали вторую хеш-функцию. Зависимость действительно похожа на O(n).

Поиск: Для демонстрации худшего случая ищем элементы по одному и тому же хешу. По графику похоже на O(n).

Удаление: Удаляем элементы, выбранные в поиске, что является худшим случаем. Результат похож на O(n).

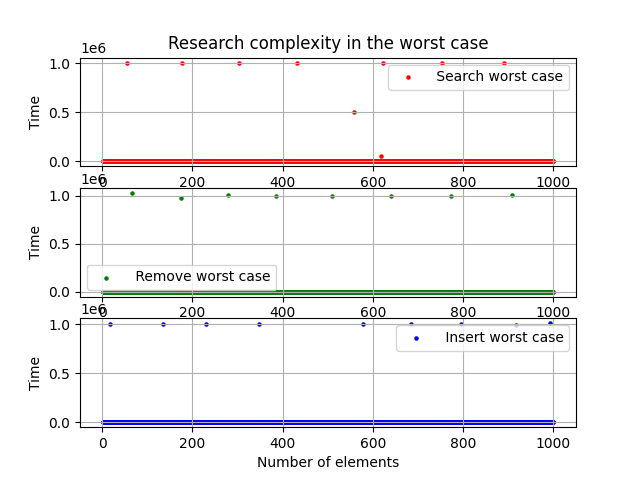


Рисунок 4 – Худший случай для двойного

Средний случай представлен на рисунке 5.

Вставка: Заполнение таблицы случайными числами. Засекаем время вставки каждого.

Поиск: Ищем элементы в таблице с ключами, начиная от нулевого и дальше в цикле. По графику видно, что время зависит от количества повторений.

Удаление: Происходит аналогично поиску.

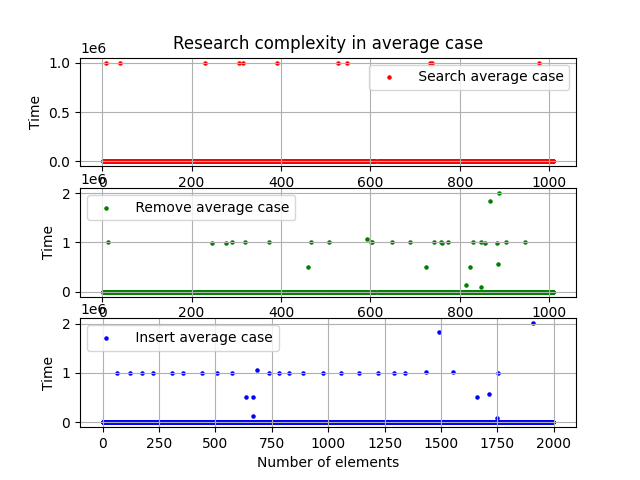


Рисунок 5 – Средний случай для открытой адресации

**заключение**

В ходе выполнения курсовой работы изучены, реализованы и исследованы такие хеш-таблицы, как хеш-таблица с двойным хешированием и хеш-таблица с открытой адресацией. Полученные экспериментальные оценки соответствуют теоретическим.

Появление коллизий при двойном хеширование происходит значительно реже, чем при открытой адресации с решением коллизий линейным методом. Если нет задачи построение оптимальной по памяти и времени работы хэш-таблицы, тогда стоит реализовывать решение коллизий с помощью двойного хеширования.

**список использованных источников**

1. Дасгупта С. Алгоритмы / С. Дасгупта, Х. Пападимитриу, У. Вазирани: Пер. с англ. под ред. А. Шеня. –– М.: МЦНМО, 2014. –– 320 с.

2. Кормен Т. Алгоритмы. Вводный курс / Т. Х. Кормен.: Пер. с англ. - М.: ООО "И.Д. Вильямс", 2014. - 208 с.

3. Кормен Т. Алгоритмы: построение и анализ, 2-е издание / Т. Х. Кормен, Ч. И. Лейзерсон, Р. Л. Ривест, К. Штайн.: Пер. с англ. – М.: Издательский дом «Вильямс», 2011. – 1296 с.

4. Лафоре Р. Структуры данных и алгоритмы в Java. Классика Computer Science. 2-е издание / Р. Лафоре. - СПб.: Питер, 2013. – 704 с.

5. Бхаргава А. Грокаем алгоритмы. Иллюстрированное пособие для программистов и любопытствующих / А. Бхаргава. – СПб: Питер, 2017. – 288с.

**приложение А**

Визуализация хеш-таблицы с открытой адресацией представлена на рисунке 6.

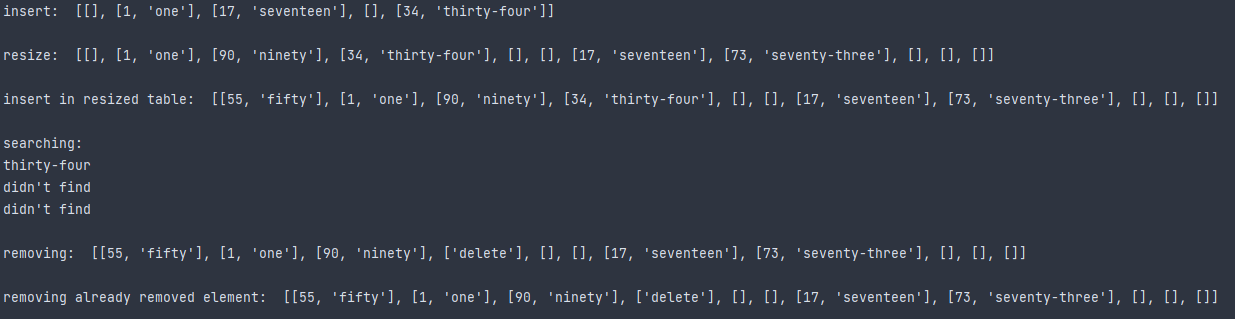


Рисунок 6 – Визуализация открытой адресации

Визуализация хеш-таблицы с двойным хешированием представлена на рисунке 7.

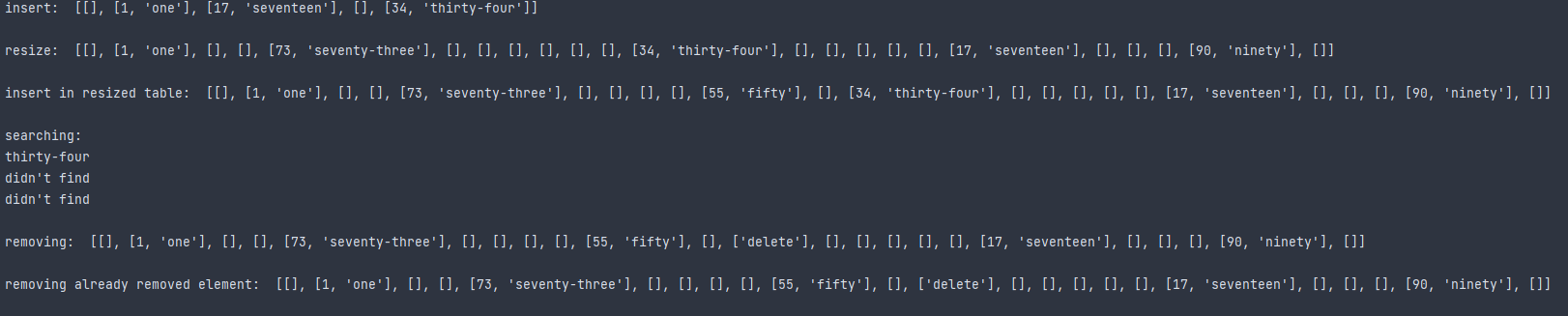


Рисунок 7 – Визуализация двойного хеширования