МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«Челябинский государственный университет»**

**(ФГБОУ ВО «ЧелГУ»)**

Институт информационных технологий

Кафедра информационных технологий и экономической информатики

ОТЧЕТ

по лабораторной работе №6

Авторы отчета С.А. Костюк ПрИ-202

подпись инициалы, фамилия группа

Д. Дударов ПрИ-202

подпись инициалы, фамилия группа

Е. Хотенов ПрИ-202

подпись инициалы, фамилия группа

Отчет защищен \_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_

дата оценка

Челябинск 2023 г.

**Цель работы:** реализовать хэш-таблицы использующие различные методы разрешения коллизий, реализовать различные функции для вычисления хэша и произвести сравнительный анализ.

**Задание 1**

1.Реализовать

1. хеш-таблицу (размер таблицы составляет m=1000), использующую метод разрешения коллизий с помощью цепочек,
2. интерфейсные операции работы с таблицей  
   (Поиск элемента, Вставка элемента, Удаление элемента).

Для вычисления хэша использовать:

* метод деления,
* метод умножения,
* свой собственный метод (обосновать выбор своего метода).

2. Произвести:

* генерацию 100000 элементов с различными ключами
* вставить все элементы в хеш-таблицу
* подсчет коэффициента заполнения, длину самой длинной и самой короткой цепочки в ячейках хеш-таблицы
* сравнительный анализ нескольких собственных (3-4) хэш-функций + 2 базовые.
* определить средний коэффициент заполнения для каждой функции
* определить наилучшую и обосновать

**Задание 2**

1.Реализовать

1. хеш-таблицу (размер таблицы составляет m=10000), использующую метод открытой адресации для разрешения коллизий,
2. интерфейсные операции работы с таблицей   
   (Поиск элемента, Вставка элемента, Удаление элемента).

Для вычисления хэша использовать:

* линейное исследование,
* квадратичное исследование,
* двойное хеширование.
* свой собственный метод (обосновать выбор) (2 шт).

2. Произвести:

* генерацию 10000 элементов с различными ключами
* вставить все элементы в хеш-таблицу
* предусмотреть корректную обработку ситуации переполнения таблицы.
* подсчет длины самого длинного кластера в таблице.
* провести сравнительный анализ всех хэш-функций.

определить наилучшую хэш-функцию.

**Задание 1**

**Пункт 1.**

Нами была реализована хэш-таблица, используящая метод разрешения коллизий с помощью цепочек. Хэш-таблица с методом разрешения коллизий с помощью цепочек — это структура данных, которая использует хэш-функцию для отображения ключей на индексы массива, а также списки (цепочки) для хранения элементов, которые отображаются на один и тот же индекс. Этот метод позволяет эффективно разрешать коллизии, которые возникают, когда два или более ключа отображаются на один и тот же индекс.

Код реализации данной хэш-таблицы показан на рисунках 1.1 – 1.2



Рис. 1.1 (Класс хэш функции)

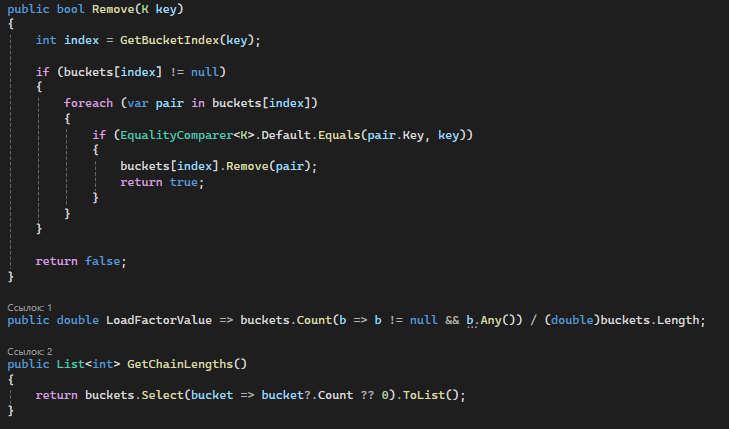


Рис. 1.2 (Класс хэш фукнции продолжение)

Нами были реализованы следующие функции для вычисления хэша:

**Метод деления -** это один из наиболее простых и широко используемых методов для вычисления хэш-значений. Он основан на операции деления и использует остаток от деления ключа на некоторое фиксированное число (обычно размер хэш-таблицы) в качестве хэш-значения.

Реализация данного метода представлена на рисунке 1.3

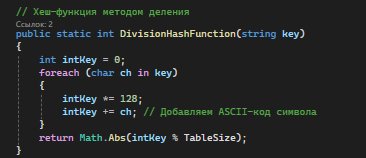


Рис. 1.3 (Метод деления)

**Метод умножения** - это еще один популярный метод для вычисления хэш-значений, который особенно эффективен для ключей, представленных в виде целых чисел. Этот метод основан на умножении ключа на константу.

Реализация данного метода показана на рисунке 1.4

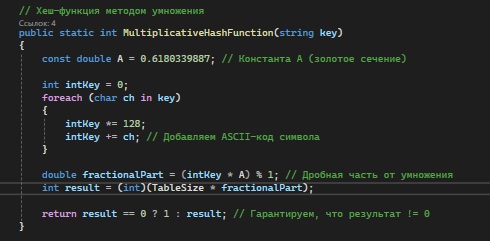


Рис. 1.4 (Метод умножения)

**Метод Adler32** - это алгоритм хэширования, разработанный Марком Адлером (Mark Adler). Алгоритм Adler-32 работает путем вычисления двух сумм (A и B) для каждого байта входных данных и их комбинирования для получения конечного 32-битного хэш-значения.

Реализация данного метода показана на рисунке 1.5

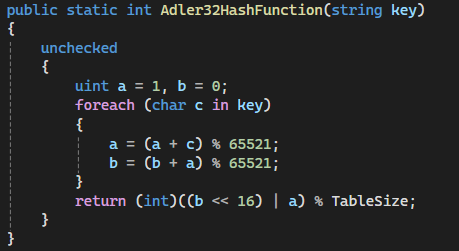


Рис. 1.5 (Метод Adler32)

**Базовая хеш-функция, использующая стандартный GetHashCode –** это метод в котором хэш-код вычисленный с помощью стандартного метода C# делится на размер таблицы (константу)

Реализация этого метода показано на рисунке 1.6

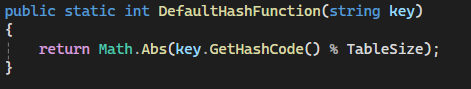


Рис. 1.7 (Метод DefaultHashCode)

**Полиномиальная хеш-функция** — это метод хэширования, который использует полином для вычисления хэш-значения.

Реализация данного метода показана на рисунке 1.8

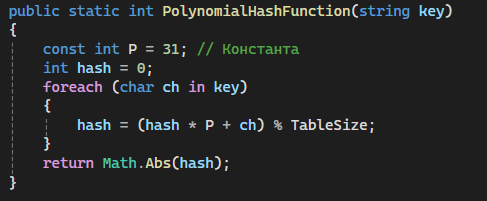


Рис. 1.8 (Полиномиальная хэш функция)

**Хеш-функция на основе XOR (XOR Hash Function)**

Описание:

Хеш-функция на основе побитового XOR применяет побитовую операцию XOR для каждого символа в строке. Это простая и эффективная функция для создания псевдослучайного распределения хешей.

Причина выбора:

1.XOR-операция эффективно перемешивает биты данных, минимизируя коллизии для небольших строк.

2.Хорошо подходит для строковых ключей.

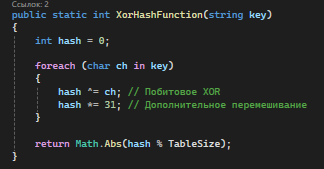


Рис. 1.9 (XOR Hash Function)

**Хеш-функция: "Shift-XOR-Multiply"**

Описание идеи:

1. Побитовый сдвиг влево и вправо помогает эффективно перемешивать биты значений символов.

2. Операция XOR с текущим символом строки равномерно распределяет ключи.

3. Умножение с простым множителем усиливает хеширование и делает коллизии менее вероятными.

Реализация данного метода показана на рисунке 1.10

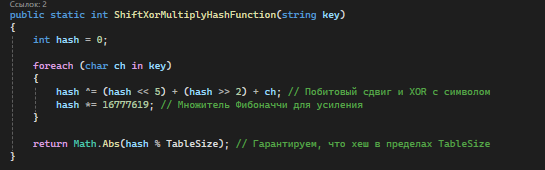


Рис. 1.10 (Shift-XOR-Multiply)

**Пункт 2**

Нами была создана и заполнена хэш-таблица размером m = 100000. Также был проведен сравнительный анализ, результаты которого показаны на рисунке 1.10

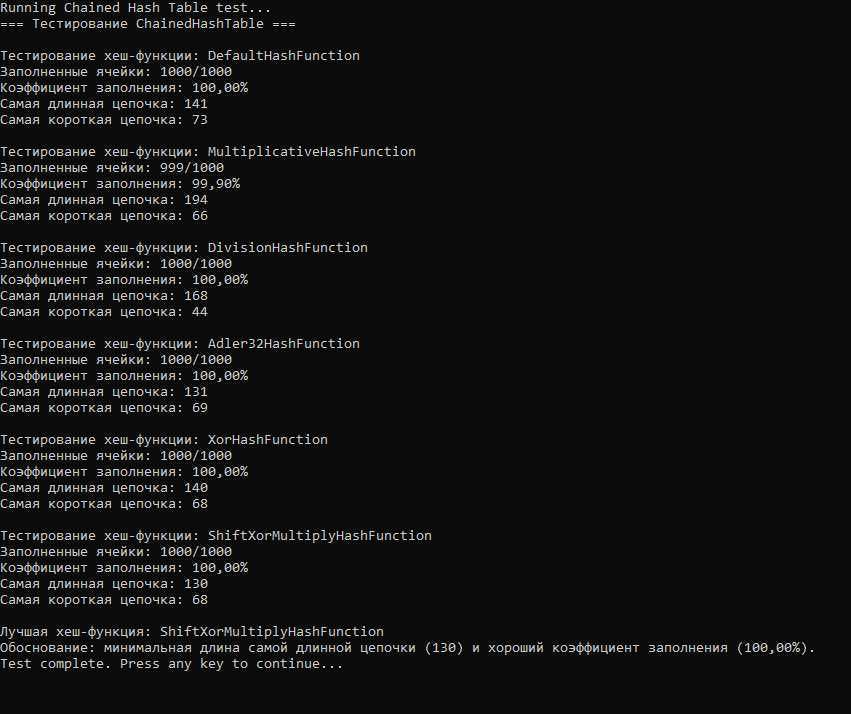


Рис. 1.10 (Сравнительный анализ)

**Задание 2**

**Пункт 1**

Нами была реализована хэш-таблица использующая метод открытой адресации для разрешения коллизий. Хэш-таблица с методом открытой адресации для разрешения коллизий — это структура данных, которая использует хэш-функцию для отображения ключей на индексы массива и разрешает коллизии путем поиска альтернативных индексов в том же массиве, когда происходит столкновение.

Код реализации данной хэш-таблицы показан на рисунках 2.1 – 2.6

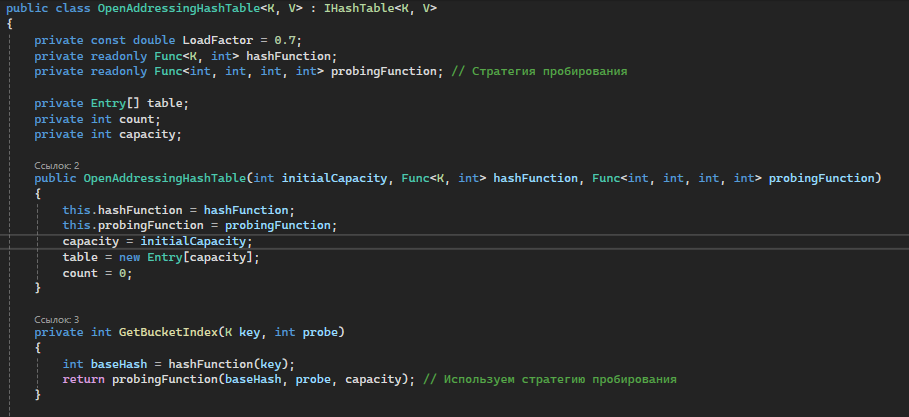


Рис. 2.1 (Класс OpenAddressingHashTable)

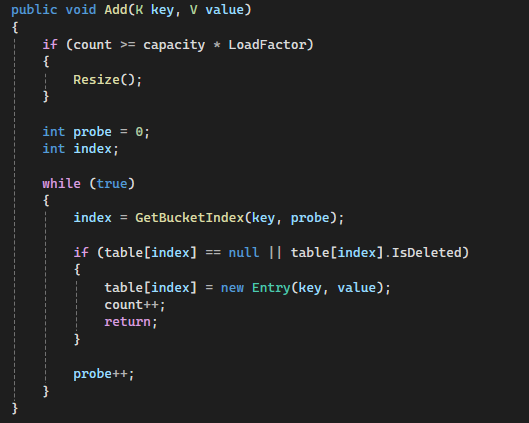


Рис. 2.2 (Класс OpenAddressingHashTable продолжение)

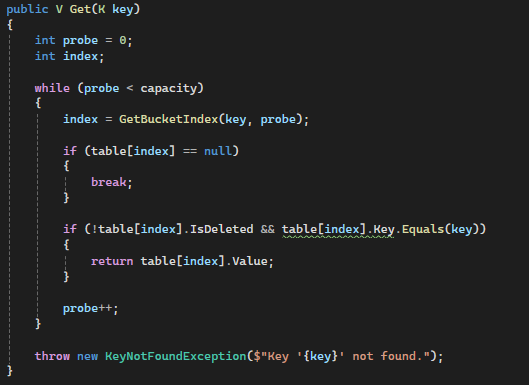


Рис. 2.3 (Класс OpenAddressingHashTable продолжение)

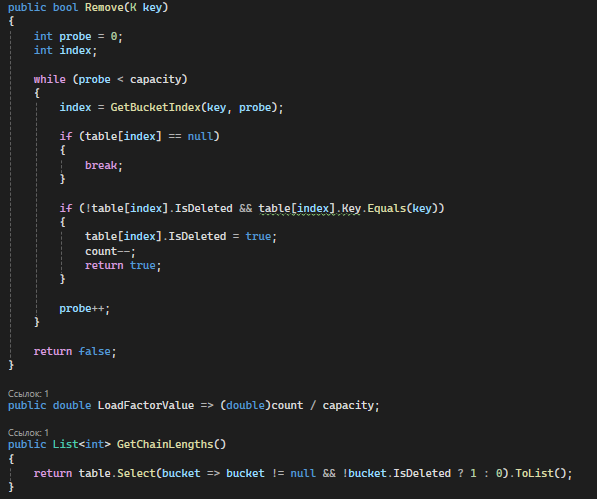


Рис. 2.4 (Класс OpenAddressingHashTable продолжение)

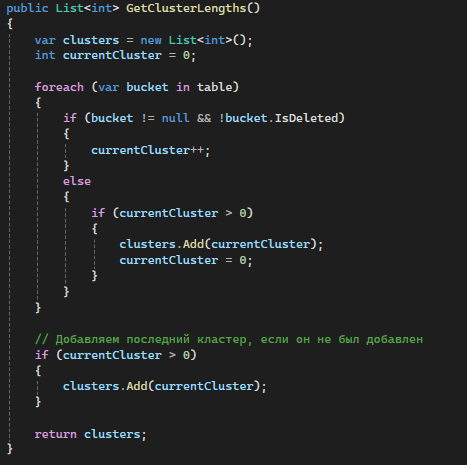


Рис. 2.5 (Класс OpenAddressingHashTable продолжение)

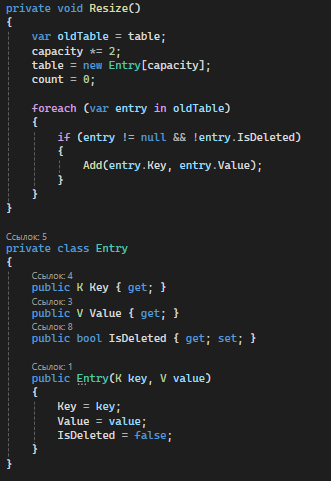


Рис. 2.6 (Класс OpenAddressingHashTable продолжение)

Для вычисления хэша мы использовали следующие стратегии пробирования:

**Линейное исследование -** это метод разрешения коллизий, при котором, если индекс, вычисленный хэш-функцией, уже занят, проверяются последующие индексы один за другим до нахождения свободного места

Реализация этой стратегии показана на рисунке 2.7

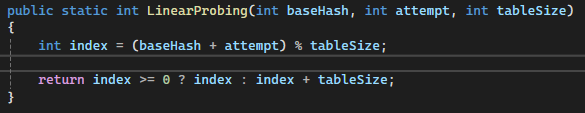


Рис. 2.7 (Линейное пробирование)

**Квадратичное исследование** - это метод разрешения коллизий, при котором, если индекс, вычисленный хэш-функцией, уже занят, проверяются индексы, отстоящие на квадратичное расстояние от исходного индекса

Реализация этой стратегии показана на рисунке 2.8

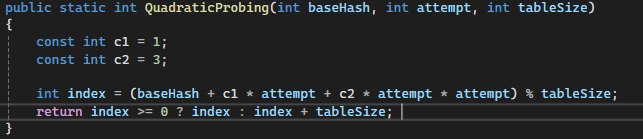


Рис. 2.8 (Квадратичное пробирование)

**Двойное исследование** - метод борьбы с коллизиями, возникающими при открытой адресации, основанный на использовании двух хеш-функций для построения различных последовательностей исследования хеш-таблицы.

Реализация этой стратегии показана на рисунке 2.9

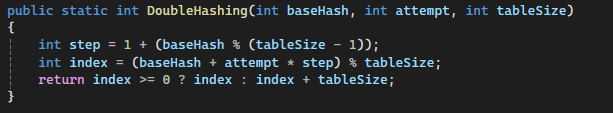


Рис. 2.9 (Двойное исследование)

**Экспоненциальное пробирование** использует степенное увеличение смещения на каждой итерации.

**Преимущества:**

Экспоненциальный рост шага помогает быстро "убежать" из кластеров, особенно при большом числе коллизий.

**Недостатки:**

Некоторые индексы могут никогда не быть достигнуты в таблице, если её размер не является степенью двойки.

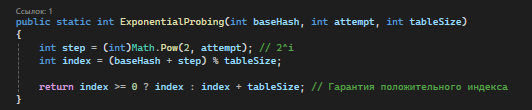


Рис. 2.9 (Экспоненциальное пробирование)

**Кубическое пробирование** увеличивает смещение на каждой итерации в кубическом порядке.

**Преимущества:**

Шаги кубического роста позволяют избежать первичного и вторичного сгущения, как в линейном и квадратичном пробировании.

Хорошо работает при большом размере таблицы.

**Недостатки:**

Может быть избыточным для небольших таблиц из-за большого разрыва между шагами.

**Пункт 2**

Нами была создана и заполнена таблица размером m = 10000. Также нами был проведен сравнительный анализ, результаты которого показаны на рисунках 2.10 и 2.11

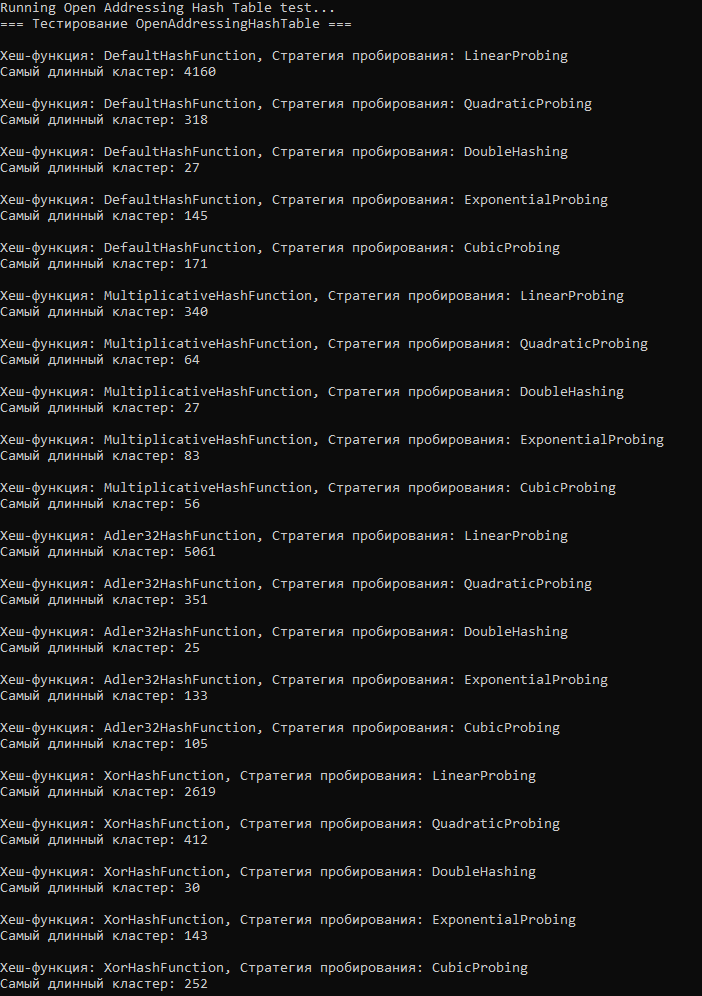


Рис. 2.10 (Результаты сравнительного анализа)

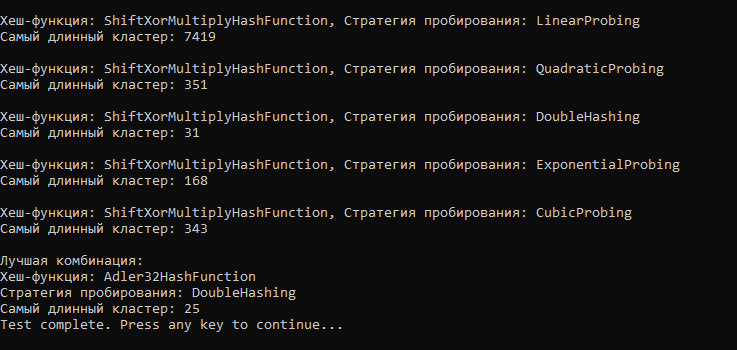


Рис. 2.11 (Результаты сравнительного анализа продолжение)