Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Основы алгоритмизации и программирования

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

на тему

АНАЛИЗ ХРАНЕНИЯ И ОБРАБОТКИ БОЛЬШИХ ОБЪЕМОВ ИНФОРМАЦИИ МЕТОДОМ ХЕШИРОВАНИЯ НА ПРИМЕРЕ АЛГОРИТМОВ СДВИГА РАЗРЯДОВ И СРЕДНИХ КВАДРАТОВ

БГУИР КР 6-05-0612-01 115 ПЗ

Студент: гр. 451001 Хренков Д.Д.

Руководитель:

асс. Фадеева Е.Е.

Минск 2025

**Введение**

**Хеш-функция –** функция, преобразующая массив входных данных произвольного размера в выходную битовую строку определённого размера в соответствии с определённым алгоритмом. Преобразование, выполняемое хеш-функцией, называется хешированием. Исходные (входные) данные называются входным массивом, «ключом», «сообщением». Результат преобразования (выходные данные) называется «хешем», «хеш-кодом», «хеш-суммой», «сводкой сообщения», «сверткой».

Хеш-функции применятся при построении ассоциативных массивов, поиске дубликатов в последовательностях данных, построении уникальных идентификаторов и для обнаружения вредоносного ПО. В своей работе я использую хеширование как инструмент, который позволяет размещать данные в структуре так, чтобы вставка, поиск и удаление элемента данных из этой структуры выполнялись за O(1). В качестве такой структуры используется хеш-таблица.

Хеш-таблица (ассоциативный массив) – это структура данных, используемая для быстрой вставки, поиска и удаления пар ключ-значение. Она работает на основе концепции хеширования, когда каждый ключ преобразуется хеш-функцией в отдельный индекс в массиве. Индекс функционирует как место хранения совпадающего значения: он сопоставляет ключи со значениями. Различных реализаций хеш-таблиц огромное множество. Самая простая реализация хеш-таблицы может быть основана на обычном массиве. Связь между самим элементом и его позицией в ассоциативном массиве задаёт хеш-функция.

Хеш-функция должна обладать следующими свойствами:

* детерминированность. Хеш-функция должна всегда возвращать одинаковый хеш-код для одного и того же входного значения. Это обеспечивает воспроизводимость результатов;
* равномерное распределение. Хеш-функция должна равномерно распределять хеш-коды по всему диапазону возможных значений, чтобы минимизировать коллизии;
* минимизация коллизий. Хеш-функция должна минимизировать вероятность коллизий (ситуаций, когда разные входные данные дают одинаковый хеш-код);
* вычислительная эффективность. Хеш-функция должна быть быстрой в вычислении, особенно при работе с большими объемами данных.
* устойчивость к инверсии (однонаправленность). По хеш-коду должно быть практически невозможно восстановить исходные данные. Это свойство важно для безопасности и защиты данных;
* лавинный эффект. Незначительные изменения во входных данных должны приводить к значительным изменениям в хеш-коде. Это свойство повышает криптографическую стойкость;
* фиксированная длина выхода. Хеш-функция должна возвращать хеш-код фиксированной длины, независимо от размера входных данных.

Целью курсовой работы является разработка программного средства для анализа алгоритмов хеширования.

Для достижения поставленной цели необходимо решить ряд задач:

* раскрыть сущность и привести примеры хеширования;
* сравнить рассмотренные алгоритмы хеширования;
* охарактеризовать требования к проектируемому программному средству;
* изложить ход анализа требований к программному средству;
* разработать функциональные требования к программному средству;
* описать функциональность программного средства;
* охарактеризовать спецификацию функциональных требований;
* осуществить проектирование динамических структур данных;
* реализовать алгоритмы хеширования.

1. **Анализ прототипов, литературных источников и формирование требований к проектируемому программному средству**
   1. **Обзор литературы**

**1.1.1** [1] geeksforgeeks [Электронный ресурс]. – Режим доступа: [https://www.geeksforgeeks.org/hash-table-data-structure/](%20https://www.geeksforgeeks.org/hash-table-data-structure/%20ttps://neerc.ifmo.ru/wiki/index.php?title=%D0%A5%D0%B5%D1%88-%D1%82%D0%B0%D0%B1%D0%BB%D0%B8%D1%86%D0%B0)

Здесь подробно описывается логика и принцип работы хэш-таблиц с приведением примеров конкретных реализаций на различных языках программирования.

* 1. **Примеры решения аналогичных задач**

Методы хеширования широко применяются для хранения и обработки больших объемов информации. Рассмотрены примеры их использования:

1. **Базы данных**. В реляционных и NoSQL базах данных хеширование используется для индексации данных, что позволяет ускорять поиск записей. Например, MySQL применяет хеш-индексы для оптимизации выборки строк по ключу.
2. **Кеширование данных**. В распределённых системах, таких как Redis и Memcached, хеширование обеспечивает быстрый доступ к закешированным данным, снижая нагрузку на основное хранилище.
3. **Хеш-таблицы в программировании**. В языках программирования, включая C++, Java, Python, используются стандартные структуры данных (unordered\_map, HashMap, dict), реализованные на основе хеш-таблиц. Это позволяет эффективно хранить и извлекать значения по ключу, обеспечивая быстрый доступ к данным в среднем за O(1).
4. **Компьютерная безопасность**. Хеш-функции (MD5, SHA-1) применяются для проверки целостности данных и хранения паролей в зашифрованном виде. В криптографии они обеспечивают защиту от подделки данных, а также используются в цифровых подписях и сертификатах безопасности.
5. **Распределенные системы**. Хеширование ключей используется для балансировки нагрузки между серверами в кластерах, например, согласованное хеширование в Amazon DynamoDB. Этот метод помогает равномерно распределять запросы между узлами системы.

**1.2.1 «MD-5»**

MD-5 (Message Digest Algorithm 5) – это широко используемая криптографическая хеш-функция, разработанная в 1991 году Рональдом Ривестом. Она принимает на вход данные произвольной длины и возвращает хеш-код фиксированной длины (128 бит). MD-5 был разработан для обеспечения целостности данных и проверки подлинности. Он использует 512-битные блоки данных и выполняет 64 раунда обработки. В настоящее время MD-5 считается устаревшим из-за уязвимостей, которые позволяют находить коллизии.

Алгоритм работы:

Дополнение данных: входные данные дополняются до длины, кратной 512 битам.

Инициализация буфера: используется 128-битный буфер, инициализированный фиксированными значениями.

Обработка блоков: каждый 512-битный блок обрабатывается в 64 раундах, где применяются нелинейные функции, сложение по модулю и циклические сдвиги.

Формирование хеша: после обработки всех блоков финальный хеш формируется из буфера.

**1.2.2 «SHA-1»**

SHA-1 (Secure Hash Algorithm 1) — это криптографическая хеш-функция, разработанная NSA в 1995 году. Она принимает на вход данные произвольной длины и возвращает хеш-код длиной 160 бит. SHA-1 был разработан для обеспечения более высокой безопасности по сравнению с MD-5. Он использует 512-битные блоки данных и выполняет 80 раундов обработки. Как и MD-5, SHA-1 сейчас считается устаревшим из-за обнаруженных уязвимостей.

Алгоритм работы:

Дополнение данных: входные данные дополняются до длины, кратной 512 битам.

Инициализация буфера: используется 160-битный буфер, инициализированный фиксированными значениями.

Обработка блоков: каждый 512-битный блок обрабатывается в 80 раундах, где применяются нелинейные функции, сложение по модулю и циклические сдвиги.

Формирование хеша: после обработки всех блоков финальный хеш формируется из буфера.

**1.2.3 Сравнение рассмотренных алгоритмов**

Скорость:

MD-5: быстрее, чем SHA-1, благодаря меньшему количеству раундов (64 против 80).

SHA-1: медленнее из-за большего количества раундов, что увеличивает время вычисления.

Безопасность:

MD-5: считается небезопасным из-за уязвимостей, позволяющих находить коллизии за приемлемое время.

SHA-1: более безопасен, чем MD-5, но также уязвим к атакам на коллизии. В 2017 году была продемонстрирована практическая возможность нахождения коллизий.

Размер выходного ключа:

MD-5: 128 бит.

SHA-1: 160 бит.

Примеры использования:

MD-5: использовался для проверки целостности данных, в цифровых подписях и в некоторых системах контроля версий (например, Git). Однако сейчас его использование не рекомендуется.

SHA-1: использовался в SSL/TLS сертификатах, цифровых подписях и других криптографических приложениях. Сейчас заменяется на более безопасные алгоритмы, такие как SHA-256.

* 1. **Требования к проектируемому программному средству**

**1.3.1** Назначение разработки:

Разрабатываемое программное средство предназначено для исследования эффективности методов хеширования при обработке больших объемов данных. В процессе работы выполняется сравнение алгоритмов хеширования по скорости поиска и уровню наполненности области хранения. В курсовой работе рассматриваются следующие алгоритмы хеширования: метод средних квадратов и метод сдвига разрядов.

**1.3.2** Состав выполняемых функций**:**

1. Генерация и размещение записей с уникальными ключами.

Применение двух методов хеширования: метода сдвига разрядов и метода средних квадратов.

1. Определение областей переполнения с использованием метода цепочек.
2. Анализ плотности заполнения основной области.
3. Построение графиков распределения данных и скорости поиска записей.

**1.3.3** Входные данные:

1. Количество пакетов, варьируемое от 20 до 200 000 с различным шагом.
2. Файл с последовательностью записей, состоящих из ключевого строкового поля, строкового информационного поля и числового информационного поля (не менее 1 000 000 элементов).

**1.3.4** Выходные данные:

1. Файлы с результатами хеширования.
2. Статистические данные о наполненности памяти и проценте записей в области переполнения.
3. Графики, иллюстрирующие распределение записей.
   * 1. Условия эксплуатации:

Программное средство должно поддерживать работу на ОС Windows.

* + 1. Язык и среда разработки:

1. **Язык разработки**: Delphi, так как он обеспечивает создание высокопроизводительных и надежных приложений с графическим интерфейсом.
2. **Среда разработки**: Embarcadero RAD Studio, поскольку она предоставляет мощные инструменты для визуального проектирования и отладки приложений на Delphi.
   * 1. Дополнительные требования:

К дополнительным требованиям относится интуитивно понятный пользовательский интерфейс и визуализация результатов анализа.

**2 Анализ требований к программному средству и разработка функциональных требований**

**2.1 Теоретический анализ, математическое обоснование и модели**

**Метод средних квадратов**

1. Ключ возводится в квадрат.

2. Из полученного результата выбираются центральные цифры.

3. Центральные цифры умножаются на константу, приводя их к диапазону допустимых значений адресов.

Этот метод работает хорошо, если ключи равномерно распределены, так как возведение в квадрат усиливает различия между значениями, а выбор центральных цифр помогает уменьшить вероятность кластеризации данных.

Пример работы алгоритма:

Допустим, ключ = 3456

1. Возведение в квадрат:

3456^2 = 11 943 936

2. Выбор центральных цифр (например, трёх): 943

3. Приведение к диапазону 0–N (умножение на коэффициент и деление на N).

Этот метод хорошо подходит для случаев, когда количество пакетов заранее неизвестно или изменяется, так как центральные цифры дают равномерное распределение. Однако если ключи имеют схожие значения, возможны коллизии.

**Метод сдвига разрядов**

Метод сдвига разрядов основывается на разбиении числового представления ключа на части и последовательном сложении этих частей. Алгоритм:

1. Числовой ключ разделяется на две части.

2. Младшая часть складывается со старшей.

3. Операция продолжается до тех пор, пока число цифр результата не станет равным количеству цифр в числе пакетов.

Пример работы алгоритма:

Допустим, ключ = 786543

1. Разбиение: 78 6543

2. Сложение частей:

78 + 6543 = 6621

3. Повторение, если нужно привести к меньшему количеству цифр.

Метод сдвига разрядов удобен для обработки ключей переменной длины и обеспечивает хорошее распределение значений. Однако при некоторых наборах данных возможны частые коллизии, если сумма частей даёт схожие результаты.

**Организация области переполнения: метод цепочек**

**Метод цепочек** – это один из способов разрешения коллизий в хеш-таблицах, при котором элементы, получившие одинаковое значение хеш-функции, объединяются в связанные списки (цепочки). В каждой ячейке таблицы хранится указатель на начало списка элементов с одинаковым хеш-значением.

**Принцип работы метода цепочек**

1. **Хеширование ключа**
2. **Проверка ячейки:**

Если ячейка пуста, элемент просто добавляется.

1. **Разрешение коллизии:**

Если ячейка уже занята, создаётся или расширяется связанный список, и новый элемент добавляется в этот список.

1. **Поиск элемента:**

Выполняется по указанному индексу, а затем осуществляется линейный поиск в списке.

1. **Удаление элемента:**

При удалении элемента из цепочки она корректируется путём удаления узла и перепривязки указателей.

**2.2 Описание функциональности программного средства**

Цель разработки программного средства – анализ алгоритмов хеширования. Исходя из цели функциональность представлена следующими процессами:

1. Генерация и запись тестового набора данных;
2. Поиск записи в хеш-таблице;
3. Сохранение результатов хеширования в файл;
4. Визуализация результатов в виде графиков;
5. Оценка наполненности основной области хранения;
6. Просмотр содержимого каждого пакета;
7. Просмотр результата размещения записей исходных данных;
8. Сравнительный анализ двух методов хеширования.

**2.3 Спецификация функциональных требований**

Функциональные требования определяют основные задачи, которые должно выполнять программное средство, а также его поведение в различных условиях.

1. Генерация тестового набора данных.

Программа должна создавать файл с последовательностью записей, содержащих уникальные символьные ключи (6 символов) и два информационных поля (одно строковое, одно числовое). Число записей должно быть не менее 1 000 000.

2. Обработка области переполнения.

В случае коллизий данные должны записываться с использованием метода цепочек.

3. Анализ плотности заполнения.

Программное средство должно вычислять коэффициент плотности заполнения основной области памяти и процент записей, попавших в область переполнения.

4. Поиск записей по ключу.

Должна быть реализована функция поиска записи по заданному ключу. Поиск должен выполняться в хеш-таблице с учётом метода разрешения коллизий.

5. Визуализация результатов.

Программа должна строить графики, демонстрирующие распределение данных в хеш-таблице, процентное соотношение коллизий и сравнение эффективности двух методов хеширования.

6. Сравнительный анализ методов.

После выполнения хеширования и поиска программа должна предоставлять пользователю статистику, включающую среднее время поиска, уровень заполненности основной области и число записей, размещённых в области переполнения.

1. **Проектирование программного средства**

**3.1** **Проектирование динамических структур данных**

В программном средстве число пакетов изменяется. Хеш-таблица реализована с помощью динамического массива. Базовым типом является тип TDataRecord.

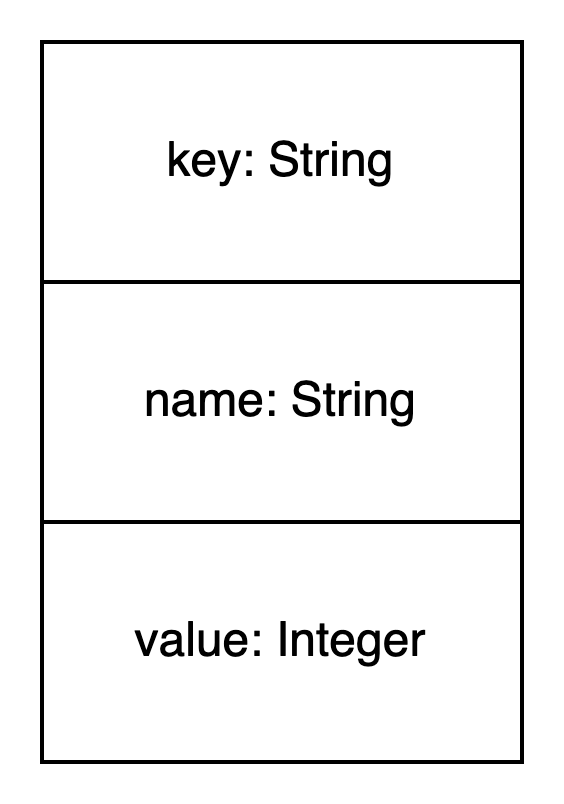


Рисунок 3.1 – Структура записи типа TDataRecord

Для разрешения коллизий я выбрал метод цепочек, в связи с чем область переполнения реализована с помощью однонаправленных связных списков.

**3.2** **Разработка алгоритмов хеширования**

**3.2.1 Метод сдвига разрядов**

Числовое значение ключа делится на две части, младшая часть складывается со старшей. Описанный процесс продолжается до тех пор, пока количество цифр результата не окажется равным количеству цифр числа пакетов N.

Схема алгоритма представлена в Приложении А.

**3.2.2 Метод средних квадратов**

Ключ возводится в квадрат, выбираются центральные цифры. Умножаются на константу.

Схема алгоритма представлена в Приложении А.

**3.2.3 Алгоритм управления областью переполнения**

Для связи основной области памяти с областью переполнения применяются цепочки адресов. Это означает, что каждая ячейка в основной области содержит адрес первой записи в области переполнения, связанной с этой ячейкой.

Основное преимущество такого подхода к организации области переполнения заключается в том, что среднее время вставки и поиска элемента в таблице зависит исключительно от среднего количества коллизий, возникающих при вычислении хеш-функции. Кроме того, хеш-таблицы, использующие метод цепочек, никогда не переполняются в традиционном смысле, так как новые элементы всегда можно добавить в связанные списки. Это делает вставку и поиск элементов относительно простыми, даже при большом количестве данных в таблице. Однако стоит отметить, что производительность некоторых методов хеширования может значительно снижаться, если таблица почти полностью заполнена.

Среди недостатков метода цепочек можно выделить необходимость работы с динамическими структурами данных, что усложняет реализацию. Ещё одним минусом является возможность возникновения очень длинных связанных списков, если количество цепочек относительно невелико, что может негативно сказаться на производительности.

Несмотря на эти недостатки, метод цепочек считается одним из наиболее эффективных способов организации области переполнения, что и обусловило его использование в данной курсовой работе.

**3.2.4 Алгоритм получения случайных ключей**

Псевдослучайная последовательность — последовательность чисел, которая была вычислена по некоторому определённому арифметическому правилу, но имеет все свойства случайной последовательности чисел в рамках решаемой задачи.

Хотя псевдослучайная последовательность в этом смысле часто, как может показаться, лишена закономерностей, однако любой псевдослучайный генератор с конечным числом внутренних состояний повторится после очень длинной последовательности чисел. Это может быть доказано с помощью принципа Дирихле.

Генератор псевдослучайных чисел — алгоритм, порождающий последовательность чисел, элементы которой почти независимы друг от друга и подчиняются заданному распределению. В качестве такого генератора мною был выбран линейный конгруэнтный метод (Linear Congruential Generator, LCG).

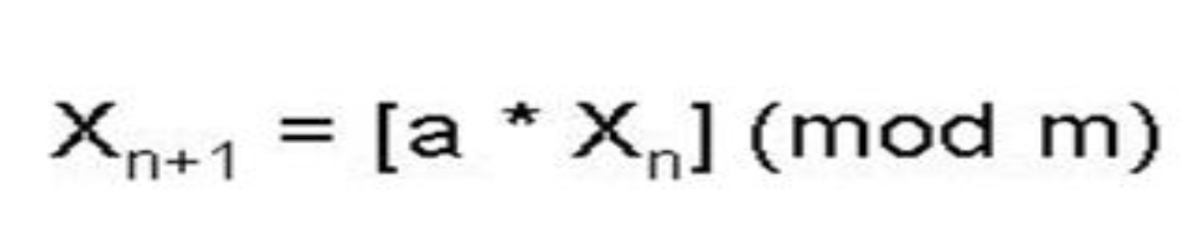


Рисунок 3.2.4 – Формула вычисления n-ого символа ключа

1. **Создание программного средства**

**4.1 Проектирование модулей программного средства**

Разработкам программного средства выполнялась на основе функциональных требований и алгоритмов, представленных в разделе 3.

В программном модуле используется 9 модулей:

1. Модуль ufMain – содержит логику взаимодействия главной формы с пользователем;
2. Модуль ufCharts – содержит в себе логику отрисовки графиков;
3. Модуль uHashTable – содержит в себе класс хеш-таблицы с реализованными методами поиска, вставки, чтения из файла;
4. Модуль uHash – содержит в себе описанные в разделе 3 алгоритмы хеширования;
5. Модуль uLCG – содержит в себе логику для генерации псевдослучайных ключей;
6. Модуль uDataGenerator – содержит в себе логику для генерации псевдослучайных данных для анализа;
7. Модуль uAnalysis – содержит в себе логику анализа;
8. Модуль uFiles – содержит логику чтения из файлов и записи в файлы;
9. Модуль uTypes – содержит в себе типы.

**4.2 Разработка модулей программного средства**

Основные подпрограммы описаны в таблицах х1-хн

uAnalysis

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Название** | **Назначение** | **Сигнатура** | **Параметры** | **Описание параметров** |
| AnalysForTables | Проводит анализ хеш-таблицы: загружает данные, выполняет поиск, возвращает статистику | function AnalysForTables(var HashTable: THashTable; FileName: String; const keys: TOccupiedKeys): TStatistic; | HashTable: THashTable | Хеш-таблица для анализа |
| FileName: String | Имя файла с данными |
| keys: TOccupiedKeys | Массив ключей для поиска |

uDataGenerator

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Название** | **Назначение** | **Сигнатура** | **Параметры** | **Описание параметров** |
| GenerateTestData | Генерирует тестовые данные и записывает в файл, заполняет массив ключей | procedure GenerateTestData (const FileName: string; RecordCount: Integer; var keys: TOccupiedKeys); | FileName: string | Имя файла для сохранения |
| RecordCount: Integer | Количество генерируемых записей |
| keys: TOccupiedKeys | Массив ключей (возвращаемый) |

uFiles

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Название** | **Назначение** | **Сигнатура** | **Параметры** | **Описание параметров** |
| ReadKeysFromFile | Читает ключи из файла данных в массив keys | procedure ReadKeysFromFile (const FileName: String; var keys: TOccupiedKeys); | FileName: String | Имя файла с данными |
| keys: TOccupiedKeys | Массив ключей (возвращаемый) |

uHash

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Название** | **Назначение** | **Сигнатура** | **Параметры** | **Описание параметров** |
| TKeyToNumber | Конвертирует ключ в числовое представление | function TKeyToNumber (const Key: TKey): UInt64; | Key: TKey | Входной ключ |
| SquareHash | Вычисляет хеш методом середины квадрата | function SquareHash(const Key: TKey; TableSize: UInt32): UInt64; | Key: TKey | Ключ для хеширования |
| TableSize: UInt32 | Размер таблицы |
| ShiftHash | Вычисляет хеш методом сдвига | function ShiftHash(const Key: TKey; TableSize: UInt32): UInt64; | Key: TKey | Ключ для хеширования |
| TableSize: UInt32 | Размер таблицы |

uHashTable

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Название** | **Назначение** | **Сигнатура** | **Параметры** | **Описание параметров** |
| THashTable.Create | Инициализирует хеш-таблицу | constructor Create(HashFunc: THashFunc; TableSize: Integer); | HashFunc: THashFunc | Используемая хеш-функция |
| TableSize: Integer | Количество бакетов |
| THashTable.Destroy | Освобождает ресурсы | destructor Destroy; override; | - | - |
| THashTable.Insert | Добавляет запись | procedure Insert (const Rec: TDataRecord); | Rec: TDataRecord | Данные для вставки |
| ТHashTable.Find | Осуществляет поиск записи по ключу | function Find (const Key: TKey; | Key: TKey | Искомый ключ |
| out Data: TSearchData Record): Boolean; | Data: TSearchData Record | Результат поиска |
| THashTable.GetBucket | Возвращает содержимое бакета | function GetBucket(const Index: Integer; out Data: TBucket): Boolean; | Index: Integer | Индекс бакета |
| Data: TBucket | Данные бакета |
| THashTable.Delete | Удаляет запись по ключу | procedure Delete(const Key: TKey); | Key: TKey | Ключ для удаления |
| THashTable.Clear | Очищает таблицу | procedure Clear; | - | - |
| THashTable.Load FromFile | Загружает данные из файла | procedure LoadFromFile(const FileName: string); | FileName: string | Путь к файлу |
| THashTable.SearchFromKeys | Массовый поиск по массиву ключей | function SearchFromKeys (const Keys: TOccupiedKeys): Integer; | keys: TOccupiedKeys | Массив ключей для поиска |
| THashTable.Load Factor | Вычисляет коэффициент заполнения таблицы | function;LoadFactor: Double | - | - |

uLCG

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Название** | **Назначение** | **Сигнатура** | **Параметры** | **Описание параметров** |
| TLCG.Create | Создает генераторпсевдослучайных чисел | constructor Create (A, M: UInt64; Seed: UInt64); | A: UInt64 | Множитель LCG |
| M: UInt64 | Модуль LCG |
| Seed: UInt64 | Начальное состояние |
| TLCG.Next | Генерирует следующее число | function Next: UInt64; | - | - |
| TLCG.NextKey | Генерирует следующий случайный ключ | function NextKey: TKey; | - | - |

**4.3 Проектирование основных структур данных**

Таблица 4.3.1 – Основные структуры данных

|  |  |  |
| --- | --- | --- |
| **Название** | **Назначение** | **Структура** |
| TKey | Тип строкового ключа записи | string[KEY\_LENGTH]; |
| TName | Тип строкового значения записи | string[NAME\_LENGTH]; |
| TOccupiedKeys = array of TKey; | Динамический массив всех занятых ключей | array of TKey; |
| TDataRecord | Записи данных хеш-таблицы | key: TKey  ключевое поле |
| name: TName;  строковое поле |
| value: integer;  числовое поле |
| TSearchDataRecord | Запись данных хеш-таблицы для возврата после поиска | Bucket: UInt32; |
| Name: TName; |
| Value: Integer; |
| THashTable | Класс хеш-таблицы | FBuckets: array of TBucket;  Массив «бакетов» хеш-таблицы |
| FHashFunc: THashFunc;  Хеш-функция данной хеш-таблицы |
| FTableSize: UInt32;  Количество «бакетов» в хеш-таблице |
| FBucketSize: UInt32;  Размер «бакета» |
| FCount: UInt32;  Количество записей в хеш-таблице |
| FMain: UInt32;  Количество записей в главной области |
| FCollisions: UInt32;  Количество записей в области переполнения |
| FInsertStart: TDateTime;  Время начала вставки |
| FInsertTime: Double;  Время вставки |
| FSearchStart: TDateTime;  Время начала поиска |
| FSearchTime: Double;  Время поиска |
| THashFunc | Унифицированный тип хеш-функции | function(const Key: TKey; TableSize: UInt32): UInt64; |
| TBucket | Элемент хеш-таблицы | data: array of TDataRecord; динамический массив, состоящий из записей |
| len: UInt32;  Количество записей в основной области |
| Overflow: PChainNode;  Указатель на первый элемент цепочки переполнения |
| TChainNode | Запись элемента в цепочке переполнения | Data: TDataRecord;  Запись данных |
| Next: PChainNode;  Указатель на следующий элемент цепочки переполнения |
| TStatistic | Запись информации о статистике заполнения главной области и области переполнения, а также времени поиска | TableSize: UInt32;  Количество «бакетов» |
| BucketSize: UInt32;  Размер «бакета» |
| Main: UInt32;  Количество элементов в главной области |
| Collisions: UInt32;  Количество элементов в области переполнения |
| SearchTime: Double;  Время поиска |
| TStatistics | Массив, который хранит записи статистик после проведения анализа для конкретного метода хеширования | array [1..37] of TStatistic; |

1. **Создание программного средства**

**5.1 Тестирование и проверка работоспособности программного средства**

Тестирование – это процедура, выполняемая испытательной лабораторией для верификации соответствия программного продукта:

* Функциональным возможностям, зафиксированным в тактико-техническом задании и документации разработчика;
* Ожидаемому поведению (посредством сравнения фактических и прогнозируемых результатов на специально сформированном наборе тестов).

Ключевая задача процесса – идентификация дефектов, несоответствий и недостающих требований, возникших на стадии проектирования.

В рамках исследования было реализовано функциональное тестирование приложения, результаты которого систематизированы в таблице 5.1.

Таблица 5.1 – Прохождение тестов программы

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Специфи-ка тестирова-ния** | **Номер теста** | **Последовательно-сть действий** | **Ожидаемый результат** | **Получен-ный результат** |
| Запуск программы | 1 | - | Появление и загрузка главного окна | Тест пройден |
| Перемещение окна | 2 | - | Окно передвигается без задержки | Тест пройден |
| Увеличение высоты окна | 3 | - | Масштабируемость поддерживается | Тест пройден |
| Уменьшение высоты окна | 4 | - | Масштабируемость поддерживается | Тест пройден |
| Преодоление минимальной высоты окна | 5 | - | Невозможность сделать высоту меньше минимальной | Тест пройден |
| Увеличение ширины окна | 6 | - | Масштабируемость поддерживается | Тест пройден |
| Уменьшение ширины окна | 7 | - | Масштабируемость поддерживается | Тест пройден |
| Преодоление минимальной ширины окна | 8 | - | Невозможность сделать ширину меньше минимальной | Тест пройден |
| Открытие файла типа .TDataRecord | 9 | 1) Нажатие на кнопку «File» в верхнем меню программы;  2) Выбор подменю «Select file»;  3) Выбор файла в диалоговом окне. | Успешная загрузка файла, отображение страницы анализа | Тест пройден |
| Открытие файла любого другого типа | 10 | 1) Нажатие на кнопку «File» в верхнем меню программы;  2) Выбор подменю «Select file»;  3) Выбор файла в диалоговом окне. | Невозможность загрузить файл | Тест пройден |
| Генерация файла для анализа | 11 | 1) Нажатие на кнопку «File» в верхнем меню программы;  2) Выбор подменю «Geenrate file»;  3) Выбор директории и названия файла в диалоговом окне. | Успешная генерация и загрузка файла, отображение страницы анализа | Тест пройден |
| Получение графиков без проведения анализа | 12 | 1) Нажатие на кнопку «Get charts». | Невозможность нажать на кнопку | Тест пройден |
| Запуск анализа | 13 | 1) Нажатие на кнопку «Start analysis». | Успешное начало анализа, отображение соответствующего сообщения в информационной строке, появление прогресс-бара | Тест пройден |
| Получение графиков после проведения анализа | 14 | 1) Нажатие на кнопку «Get charts». | Появление нового окна с графиками, отражающие результаты анализа | Тест пройден |
| Переход на страницу «Laborato-ry» | 15 | 1) Нажатие на кнопку «Laboratory». | Успешный переход на новую страницу | Тест пройден |
| Попытка создать хеш-таблицу без выбора параметров | 16 | 1) Нажатие на кнопку «Create table». | Невозможность создать хеш-таблицу и появление модального окна с предупреждением | Тест пройден |
| Попытка создать хеш-таблицу с указанием параметров | 17 | 1) Выбор метода хеширования;  2) Выбор количества пакетов;  3) Нажатие на кнопку «Create table». | Успешное создание хеш-таблицы и появление модального окна после создания | Тест пройден |
| Попытка поиска записи по ключу без создания хеш-таблицы | 18 | 1) Попытка ввода в текстовое поле ввода ключа. | Невозможность ввода без созданной хеш-таблицы | Тест пройден |
| Попытка получения случайного ключа из массива всех ключей в хеш-таблице без создания хеш-таблицы | 19 | 1) Попытка нажатия на кнопку «Get random key». | Невозможность нажатия на кнопку без созданной хеш-таблицы | Тест пройден |
| Попытка поиска записи по существующему ключу после создания хеш-таблицы | 20 | 1) Ввод существующего ключа в текстовое поле ввода ключа;  2) Нажатие на кнопку «Find record». | Успешный поиск записи в хеш-таблице, вывод результата поиска в таблицу результата поиска записи | Тест пройден |
| Попытка поиска записи по несуществующему ключу после создания хеш-таблицы | 21 | 1) Ввод несуществующего ключа в текстовое поле ввода ключа;  2) Нажатие на кнопку «Find record». | Вывод модального окна с предупреждением о том, что запись не найдена | Тест пройден |
| Попытка получения случайного ключа из массива всех ключей в хеш-таблице после создания хеш-таблицы | 22 | 1) Нажатие на кнопку «Get random key». | Заполнение поля ввода ключа значением из массива ключей | Тест пройден |
| Попытка поиска пакета без создания хеш-таблицы | 23 | 1) Попытка ввода номера пакета. | Невозможность ввода номера пакета без создания хеш-таблицы | Тест пройден |
| Попытка поиска несуществующего пакета после создания хеш-таблицы | 24 | 1) Ввод несуществующего номера пакета.  2) Нажатие на кнопку «Get bucket». | Появление модального окна с предупреждением о том, что номер пакета не валиден. | Тест пройден |
| Попытка поиска существующего пакета после создания хеш-таблицы | 24 | 1) Ввод существующего номера пакета.  2) Нажатие на кнопку «Get bucket». | Заполнение таблицы результата поиска пакета данными из пакета | Тест пройден |
| Перемещение разделителя на странице «Laborato-ry» | 25 | 1) Перемещение разделителя страницы в право или в лево. | Адаптация интерфейса под потребность пользователя | Тест пройден |

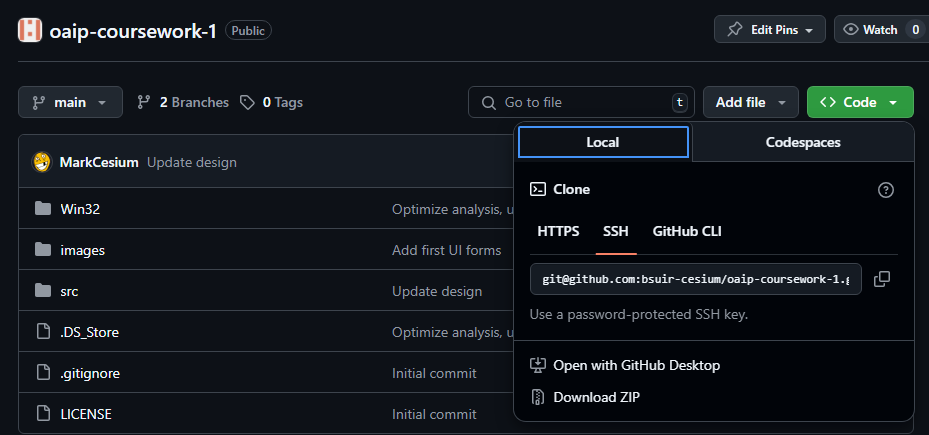
1. **Руководство по установке и использованию**

**6.1 Руководство по установке**

Перед установкой необходимо убедиться, что система соответствует системным требованиям, а именно:

* Операционная система: Windows 10/11 (32/64-bit)
* 200 МБ свободного места на диске
* 4 ГБ оперативной памяти

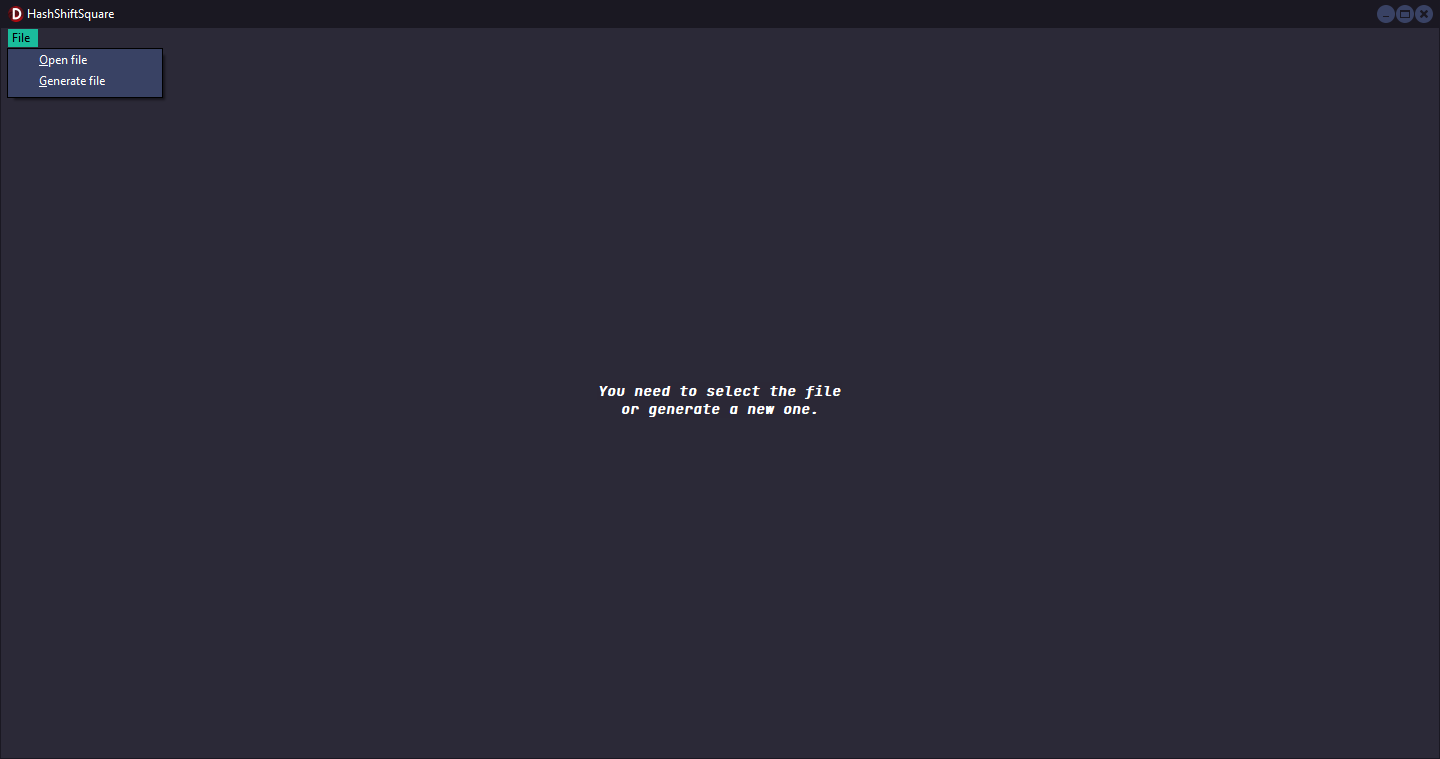
Перейдите по ссылке <https://github.com/bsuir-cesium/oaip-coursework-1>. На открывшейся странице нажмите на кнопку «Code», а затем на «Download ZIP» (вид страницы представлен на рисунке НОМЕР\_РИСУНКА)

****

Затем распакуйте архив в любое удобное для вас место, перейдите в папку Win32/Release. Здесь будет находиться исполняемый файл HashShiftSquare.exe

**6.2 Руководство по использованию**

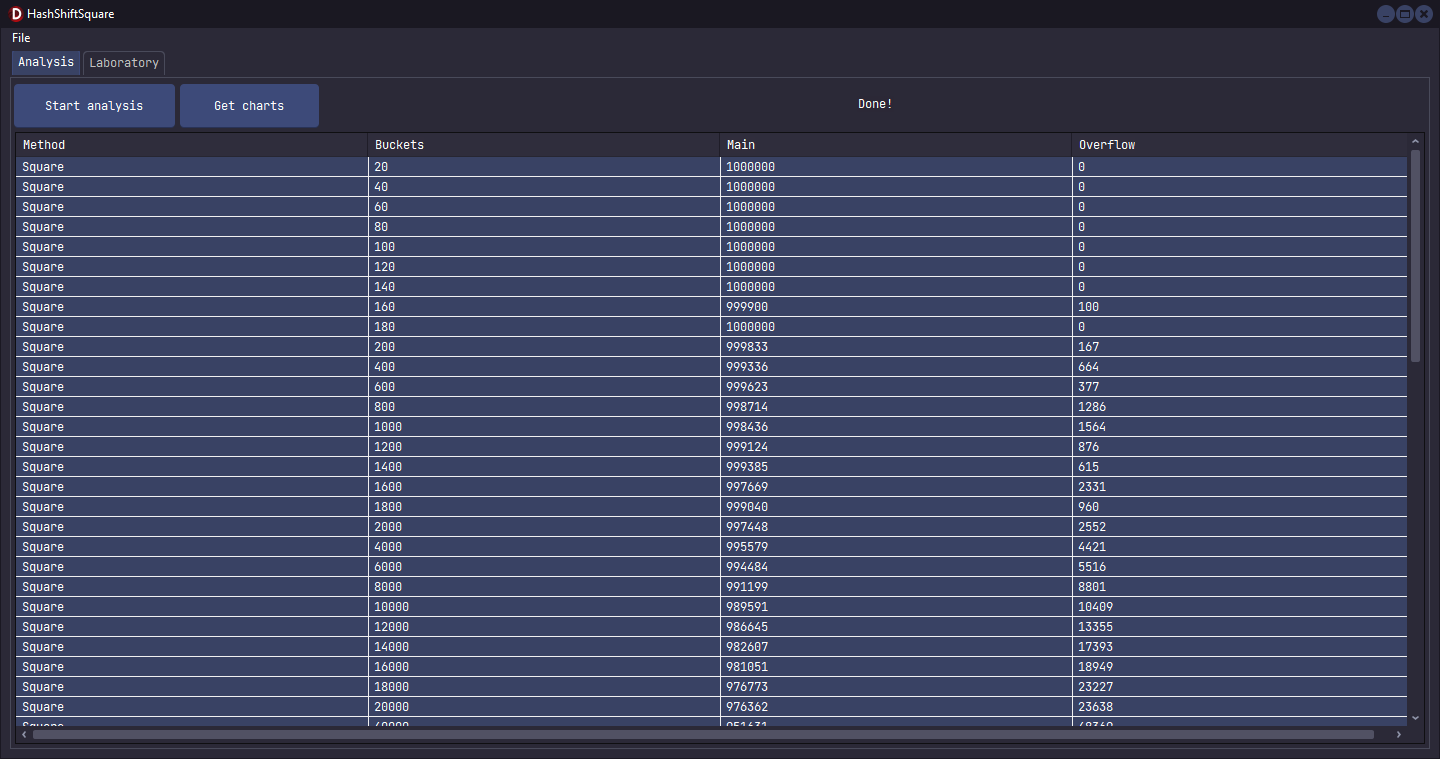
После запуска программы перед вами откроется стартовое окно программы. По середине окна будет выведен текст, который попросит вас либо выбрать существующий файл, либо сгенерировать новый. Для того, чтобы выбрать файл или сгенерировать новый, вам необходимо нажать на кнопку «File» в левом верхнем углу программы и выбрать в подменю необходимый вам вариант. Стартовое окно программы изображено на рисунке НОМЕР РИСУНКА.

****

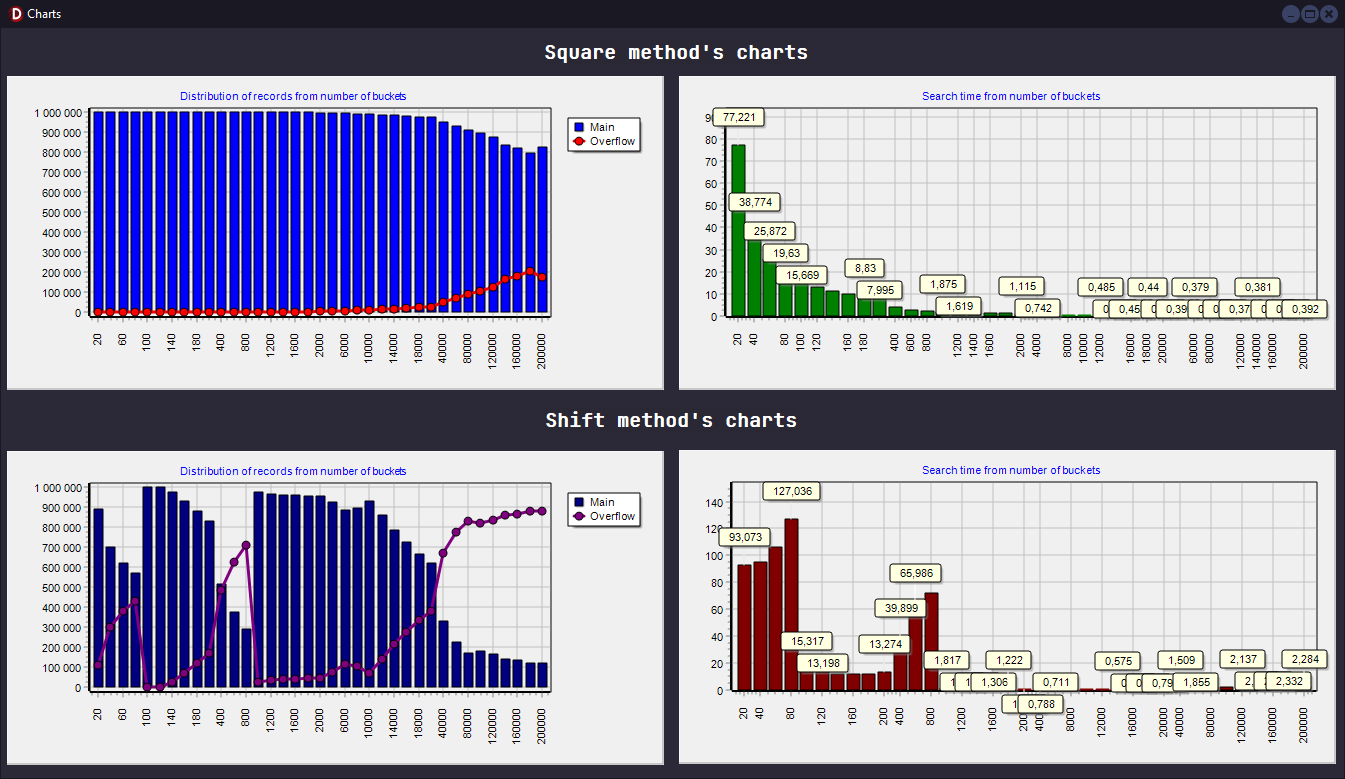
После того, как вы выберите файл или сгенерируете новый, программа будет готова к работе, и вы увидите страницу анализа. На ней расположена главная таблица с результатами анализа, кнопки для старта анализа и перехода к графикам, а также информационная строка. Страница анализа изображена на рисунке НОМЕР РИСУНКА.

****

При нажатии на кнопку «Start analysis» вы запустите анализ. Это займёт время. По ходу продвижения анализа главная таблица будет заполняться данными, а строка выше таблицы и правее кнопок будет отображать текущий процент выполнения анализа. После прохождения анализа вы сможете пользоваться приложением дальше, просмотреть полную таблицу данных, перейти на страницу лаборатории или посмотреть графики по результатам анализа. Вид страницы анализа после проведения анализа представлен на рисунке НОМЕР РИСУНКА

****

При нажатии на кнопку «Get charts» открывается окно с графиками. В нём по два графика на каждый из метод хеширования: график зависимости наполненности главной области и области переполнения от количества пакетов, график зависимости скорости поиска от количества пакетов. Окно с графиками представлено на рисунке НОМЕР РИСУНКА



При переходе на страницу лаборатории (нажатие на вкладку «Laboratory») перед вами появится страница лаборатории. В ней мы можем сами создать хеш-таблицу с выбранным методом хеширования, производить поиск ключей, просматривать содержимое каждого пакета. Также для удобства использования между левой и правой частями предусмотрел разделитель, передвигая который вы можете настроить рабочую область под ваши визуальные требования. Страница лаборатории показана на рисунке НОМЕР РИСУНКА



# **ЗАКЛЮЧЕНИЕ**

В процессе выполнения курсового проекта была проанализирована предметная область, рассмотрены существующие аналоги, проведено их сравнение и выявлены их преимущества и недостатки. В качестве языка разработки использовался Delphi.

На этапе проектирования были разработаны блок-схемы алгоритмов. В соответствии с поставленной целью было разработано программное средство для анализа хранения и обработки больших объемов информации методом хеширования на примере алгоритмов деления и складывания.

Для каждого из методов хеширования строятся соответствующие графики, которые облегчают процесс анализа и сравнения алгоритмов. Были проанализированы полученные гистограммы и предложено оптимальное количество пакетов для каждого из алгоритмов.

Проведено тестирование работоспособности разработанной программной части. Поставленная цель была выполнена в полном объеме, работоспособность подтверждена тестированием программного средства.

Разработанное программное средство может быть использовано для получения и закрепления знаний в области алгоритмов хеширования, а также при изучении дисциплин, связанных с изучением и анализом алгоритмов.

**Приложение А**

Схема алгоритма

