Санкт-Петербургский Национальный Исследовательский Университет

Информационных Технологий, Механики и Оптики

Факультет инфокоммуникационных технологий

**Домашнее задание №2**

**Вариант №1**

Выполнил:

Бацанова Е. А.

Проверил

Мусаев А.А.

Санкт-Петербург,

2024

**ОГЛАВЛЕНИЕ**

[ВВЕДЕНИЕ 3](#_fmjc2tazxxvi)

[Задание 1 4](#_ffln2tea40r9)

[ЗАКЛЮЧЕНИЕ 7](#_9xethxpssxtq)

[СПИСОК ЛИТЕРАТУРЫ 8](#_1kxd0z7sto9x)

[ПРИЛОЖЕНИЕ 9](#_1pjf9s5jjz9l)

# **ВВЕДЕНИЕ**

Хэширование – метод адресации данных для быстрого поиска по ключевым выражениям.

Целью данной работы является изучение алгоритмов хэширования текстовой информации, их реализация и тестирование на языке питон. Основными задачами работы являются:

1. реализация метода деления для хэширования строки;
2. реализация метода СRC-32 для хэширования строки;
3. тестирование программы в формате ввода пользователем строки для хэширования.

# **Задание 1**

**Задание:** Пользователь вводит текст. Программа осуществляет хэширование по указанному методу.

**Решение:**

Взят вариант 1:

* Алгоритм 1 – Деление

Реализуем хэширование методом деления с помощью функции *division\_hash(data, size)* (рис. 1). Для этого необходимо выбрать размер таблицы хэширования (в данном случае *11*). После этого преобразуем текст в ключ – целочисленное значение, например, путем сложения значений кодов символов текста, используя метод *ord()*. Функция возвращает целое число, которое и будет являться хэшем – остаток от деления полученной суммы на размер таблицы хэширования.

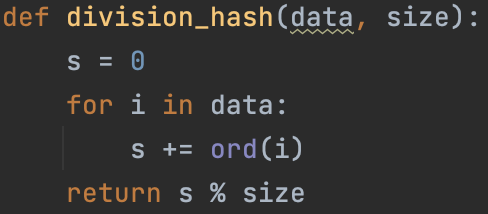


Рисунок 1 – Реализация хэширования методом деления

* Алгоритм 2 – CRC-32.

Далее реализуем алгоритм хэширования CRC-32. Всего это можно сделать тремя способами: с помощью встроенных методов python, стандартного расчета без использования библиотек и табличного (быстрого) метода расчета, который также не использует встроенные библиотеки для расчета хэша методом CRC-32.

Рассмотрим все эти методы и в конце сравним полученные значения хэша.

* Использование встроенных библиотек (рис. 2)

В python существует модуль *zlib* для вычисления CRC-32 для переданных данных. Импортируем ее и создадим функцию *crc(data)*, которая будет принимать строку данных *data* в качестве входного аргумента, затем кодировать ее в байтовый формат с использованием метода *encode()* и применять к ней метод *crc32* из модуля *zlib* для вычисления хэша.

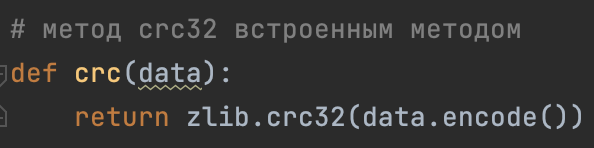


Рисунок 2 – Реализация хэширования CRC-32 с помощью встроенных библиотек python

* Расчет стандартным методом (рис. 3)

Реализуем первый способ самостоятельного расчета хэша. Для этого создадим функцию *crc32(data)*, которая также будет принимать на вход один аргумент *data*, представляющий строку данных, для которой необходимо рассчитать хэш, а затем преобразует ее в байтовую строку text. После этого инициализируются переменные crc (начальное значение CRC) и poly (порождающий полином для CRC-32, стандартно берется полином *0xEDB88320*). Далее происходит итерация по каждому байту в строке данных. Для каждого байта выполняется операция *XOR* с текущим значением *crc*. После этого во вложенном цикле происходит проверка младшего бита – если он равен 1, то выполняется сдвиг вправо на 1 бит и применяется *XOR* с порождающим полиномом. В противном случае выполняется только сдвиг вправо на 1 бит. После обработки всех байтов входных данных выполняется окончательное смещение CRC на 32 бита (используя *XOR* с 0xFFFFFFFF) и возвращается полученное значение хэша.

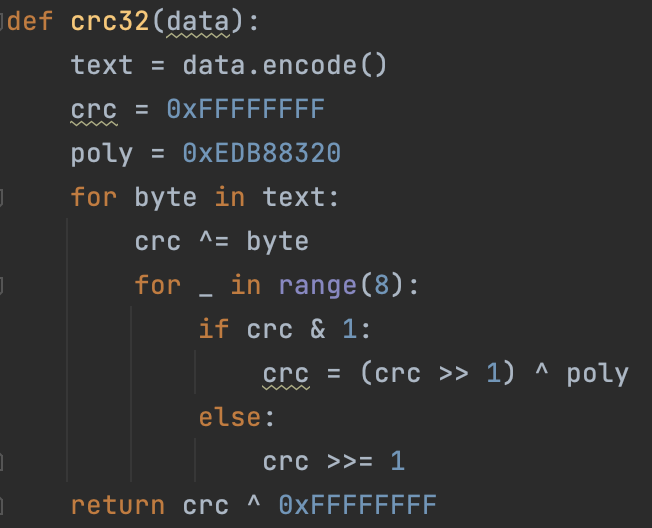


Рисунок 3 – Реализация хэширования CRC-32 стандартным методом

* Расчет табличным методом (рис. 4)

Реализуем второй способ самостоятельного расчета хэша. В данном случае таблица масок CRC-32 рассчитывается и заполняется единожды, а затем просто используется для расчета контрольной суммы (хэша). Данный способ считается более быстрым и удобным при расчете хэша для множества строк, но по сравнению со стандартным методом занимает больше памяти.

Для начала сгенерируем таблицу значений CRC-32 для всех возможных байтов (256 значений) с помощью функции *generate\_crc\_32\_table()*. Внутри цикла производим операции стандартное вычисление хэша CRC-32. Саму таблицу сохраняем в переменную *crc32\_table*.

Далее реализуем функцию, непосредственно расчитывающую хэш строки *crc32\_table\_method(data)*. Это функция снова же принимает строку данных, кодирует ее в байты, и считает CRC-32 значения для каждого байта, используя ранее сгенерированную таблицу. Результат находится в *crc* и возвращается после применения финального *XOR* оператора с *0xFFFFFFFF*.

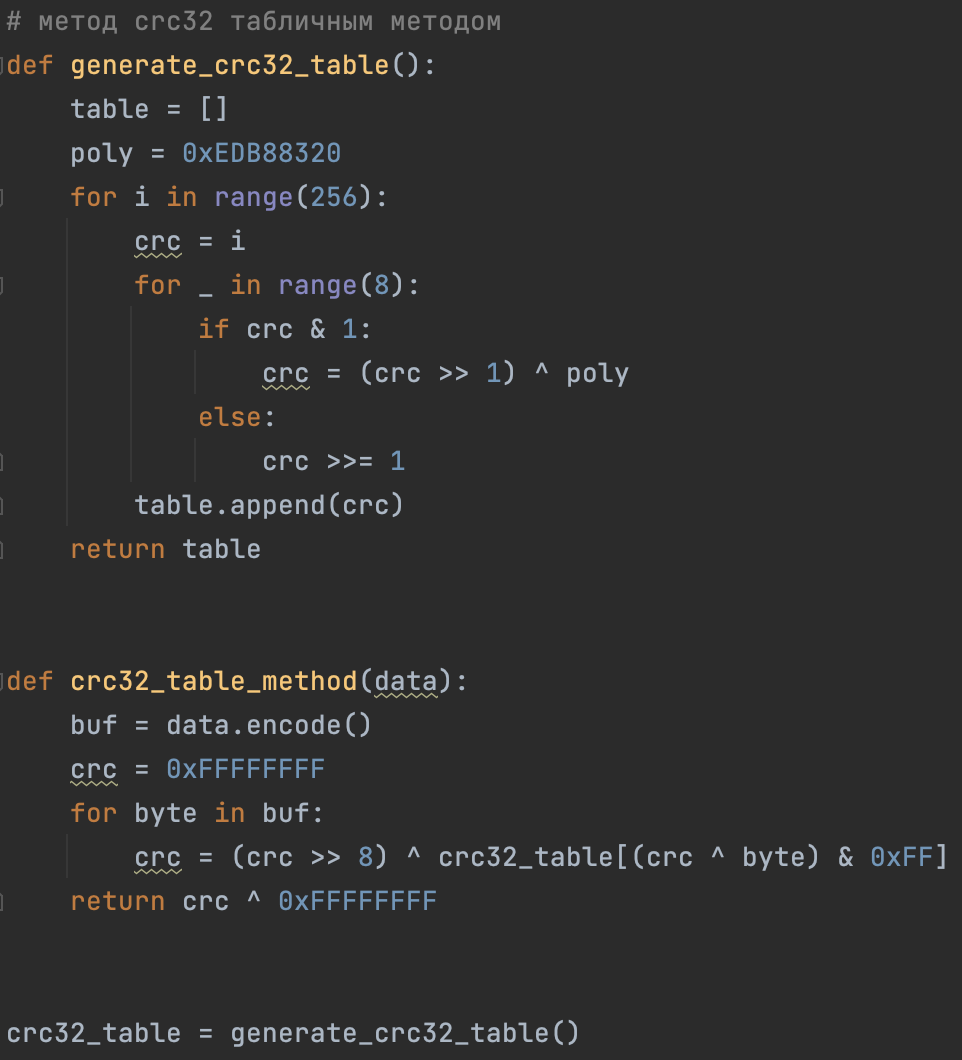


Рисунок 4 – Реализация хэширования CRC-32 табличным методом

Проверим работу программы. Видим, что хэш CRC-32, рассчитанный всеми тремя способами, совпал.

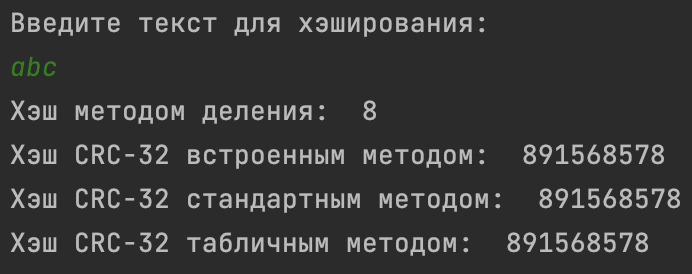


Рисунок 5 – Вывод программы расчета хэша методом деления и CRC-32

# **ЗАКЛЮЧЕНИЕ**

В результате выполнения работы были реализованы алгоритмы хэширования текстовой информации методом деления и CRC-32.

Сначала был реализован метод деления. Данный метод является простым в реализации, но способен создавать большое количества коллизий, из-за чего на входной набор строк необходимо накладывать ограничения.

После этого тремя методами было реализо хэширование CRC-32: с помощью встроенного модуля *zlib*, стандартным и табличным методом. Вывод для всех трех вариантов оказался одинаков, что говорит о том, что самостоятельный расчет хэша был реализован верно.

В итоге мы получили два метода, которые способны успешно хэшировать входные данные. Таким образом, можно сделать вывод, что алгоритмы хэширования методом деления и CRC-32 можно успешно применять для обработки текстовой информации в различных прикладных задачах.

# 

# **СПИСОК ЛИТЕРАТУРЫ**

1. Викиучебник. [Реализации алгоритмов/Циклический избыточный код](https://ru.m.wikibooks.org/wiki/%D0%A0%D0%B5%D0%B0%D0%BB%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D0%B8_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%D0%BE%D0%B2/%D0%A6%D0%B8%D0%BA%D0%BB%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B8%D0%B7%D0%B1%D1%8B%D1%82%D0%BE%D1%87%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4). [Электронный ресурс] – (Дата последнего обращения 19.05.2024);
2. Wikipedia. [Хеш-функция](https://ru.wikipedia.org/wiki/%D0%A5%D0%B5%D1%88-%D1%84%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F). [Электронный ресурс] – (Дата последнего обращения 19.05.2024).

# **ПРИЛОЖЕНИЕ**

Для удобства все файлы выгружены на GitHub: <https://github.com/kathykkKk/Algorithms-and-Data-Structures-ICT.git>