Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Российский химико-технологический университет имени Д.И. Менделеева»
Кафедра информационных компьютерных технологий

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 9

Выполнил студент группы.....КС-33..... (Вагенлейтнер Никита Сергеевич)

Ссылка на репозиторий: .....(https://github.com/MUCTR-IKT-CPP/VagenlejtnerNS_33_alg.git)

Приняли: ..... Пысин Максим Дмитриевич
.....Краснов Дмитрий Олегович
.....Лобанов Алексей Владимирович
.....Крашенинников Роман Сергеевич

Дата сдачи: ..... (26.02.2025)

Оглавление

Описание задачи.

В рамках лабораторной работы необходимо реализовать SHA1 алгоритм хеширования.

Для реализованной хеш функции провести следующие тесты:

- Провести сгенерировать 1000 пар строк длинной 128 символов отличающихся друг от друга 1,2,4,8,16 символов и сравнить хеши для пар между собой, проведя поиск одинаковых последовательностей символов в хешах и подсчитав максимальную длину такой последовательности. Результаты для каждого количества отличий нанести на график, где по оси x кол-во отличий, а по оси y максимальная длинна одинаковой последовательности.
- Провести N = 10^i(i от 2 до 6) генерацию хешей для случайно сгенерированных строк длинно 256 символов, и выполнить поиск одинаковых хешей в итоговом наборе данных, результаты привести в таблице где первая колонка это N генераций, а вторая таблица наличие и кол-во одинаковых хешей, если такие были.
- Провести по 1000 генераций хеша для строк длинной n (64, 128, 256, 512, 1024, 2048, 4096, 8192)(строки генерировать случайно для каждой серии), подсчитать среднее время и построить зависимость скорости расчета хеша от размера входных данных

Описание метода/модели.

SHA1 алгоритм хеширования

SHA-1 (Secure Hash Algorithm 1) — это криптографическая хеш-функция, разработанная NSA и опубликованная NIST. Она принимает входные данные произвольной длины и выдаёт 160-битный (20-байтовый) хеш.

Этапы работы SHA-1

Дополнение данных (Padding)

- Сообщение дополняется битом 1, затем 0, пока его длина не станет кратной 512 битам (64 байта).
- В последние 64 бита записывается исходная длина сообщения.
- Инициализация хеш-значений

Используются 5 фиксированных 32-битных констант (в шестнадцатеричном виде):

- H0 = 0x67452301
- H1 = 0xEFCDAB89
- H2 = 0x98BADCFE
- H3 = 0x10325476
- H4 = 0xC3D2E1F0
- Разбиение на блоки по 512 бит и обработка

- Каждый блок разделяется на 16 слов по 32 бита.
- Расширяется до 80 слов по 32 бита с помощью битовых сдвигов и операций XOR.
- Основной цикл SHA-1 (80 раундов)
- В каждом раунде используются:
- 5 переменных (A, B, C, D, E), обновляющихся каждый раунд.
- 4 разные константы (K) и 4 функции (F), работающие на разных этапах.

Функции F изменяются каждые 20 раундов:

- F(B, C, D) = (B AND C) OR (NOT B AND D) (0–19 раунд)
- F(B, C, D) = B XOR C XOR D (20–39 раунд)
- F(B, C, D) = (B AND C) OR (B AND D) OR (C AND D) (40–59 раунд)
- F(B, C, D) = B XOR C XOR D (60–79 раунд)

- Обновление хеш-значений
- После обработки всех 512-битных блоков обновляются H0, H1, H2, H3, H4.
- Формирование итогового хеша
- Объединение значений H0, H1, H2, H3, H4 даёт 160-битный (20-байтовый) хеш.

Выполнение задачи.

Для реализации и выполнения данного алгоритма был задействован язык Python. Были реализованы следующие программные блоки: алгоритмический блок, тестовый блок, аналитический блок полученных результатов работы алгоритма.

Блок кода отвечающий за реализацию алгоритма хеширования:

```
    def sha1_hash(string):
        sha1 = hashlib.sha1()
        sha1.update(string.encode('utf-8'))
        return sha1.hexdigest()
    # Генерация строки с отличиями
    def generate_modified_string(base_str, num_changes):
        base_list = list(base_str)
        for _ in range(num_changes):
            index = random.randint(0, len(base_str) - 1)
            base_list[index] =
random.choice('abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRST
UVWXYZ0123456789')
        return ''.join(base_list)
    # Проверка одинаковых последовательностей в хешах
    def find_longest_common_sequence(hash1, hash2):
        max_len = 0
        for i in range(len(hash1)):
            for j in range(i + 1, len(hash1) + 1):
                if hash1[i:j] in hash2:
                    max_len = max(max_len, j - i)
        return max_len
```

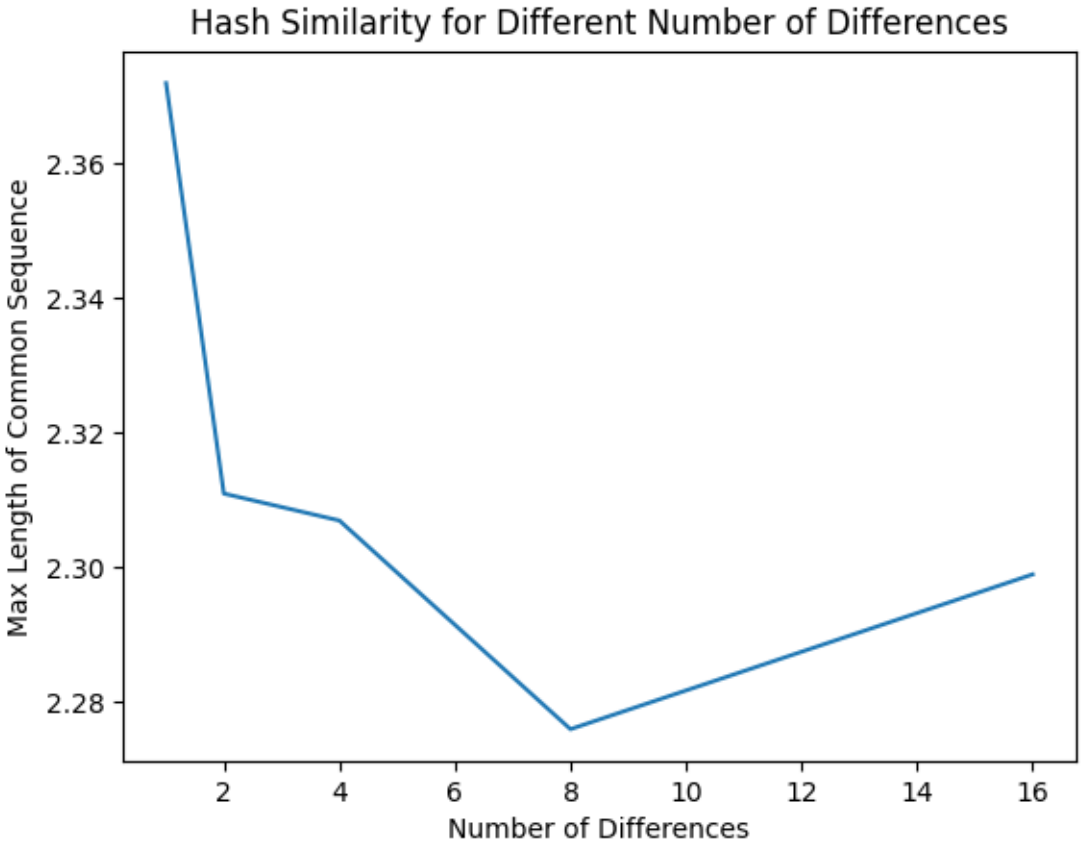
Блоки кода отвечающие за тестирование:

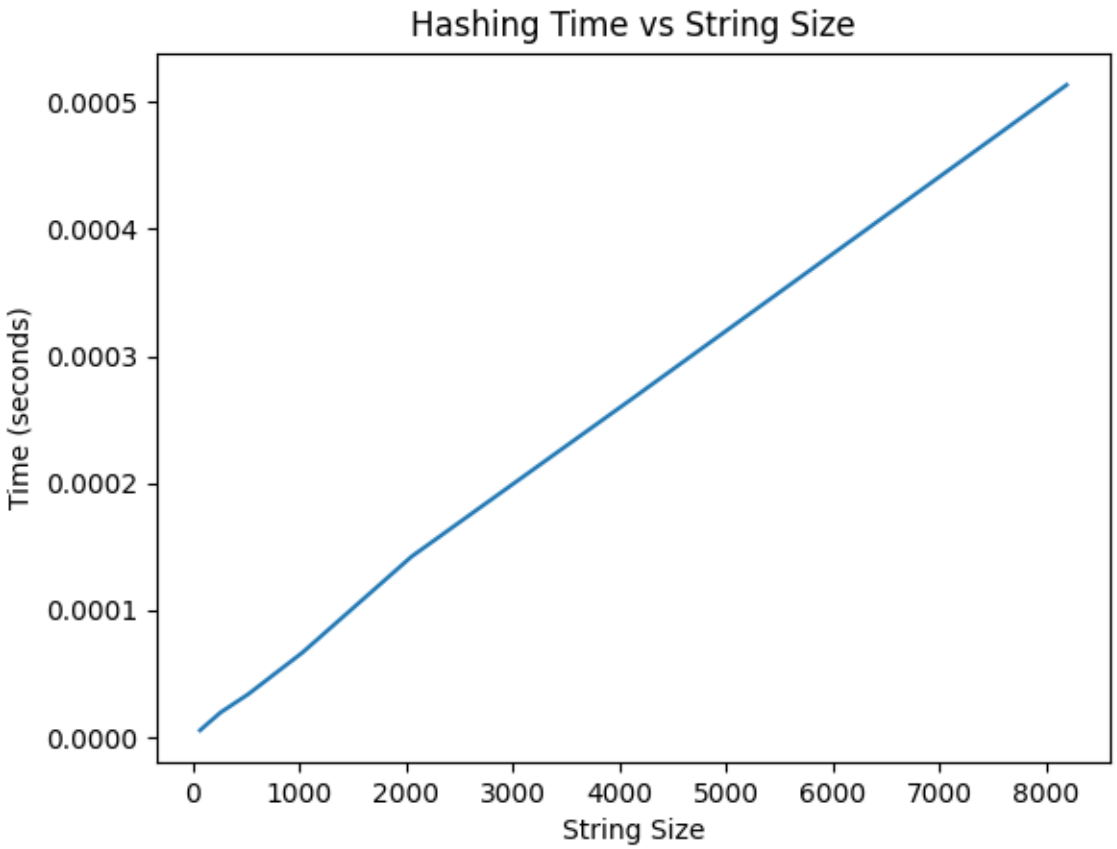
```
    # Тестирование с разными количеством отличий в строках
    def test_hash_similarity():
        base_string = 'a' * 128
        differences = [1, 2, 4, 8, 16]
        results = []
        for diff in differences:
            max_lengths = []
            for _ in range(1000):  # 1000 пар строк
                string1 = generate_modified_string(base_string, diff)
                string2 = generate_modified_string(base_string, diff)
                hash1 = sha1_hash(string1)
                hash2 = sha1_hash(string2)
                max_len = find_longest_common_sequence(hash1, hash2)
                max_lengths.append(max_len)
            avg_max_length = np.mean(max_lengths)
            results.append((diff, avg_max_length))
        # Построение графика
        diffs, avg_max_lengths = zip(*results)
        plt.plot(diffs, avg_max_lengths, label="Average Max Length")
        plt.xlabel("Number of Differences")
        plt.ylabel("Max Length of Common Sequence")
        plt.title("Hash Similarity for Different Number of Differences")
        plt.show()
    # Тестирование на одинаковые хеши для разных N
    def test_duplicate_hashes():
        results = []
        for i in range(2, 7):  # Для N = 10^i, где i от 2 до 6
            N = 10 ** i
            hashes = set()
            duplicates = 0
            for _ in range(N):
                random_string = ''.join(

random.choices('abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQ
RSTUVWXYZ0123456789', k=256))
                h = sha1_hash(random_string)
                if h in hashes:
                    duplicates += 1
                else:
                    hashes.add(h)
            results.append((N, duplicates))
```

Результаты работы программы выводится в виде графика зависимости и приведены ниже. Так же результаты тестирования в виде строк данных хранятся в output.txt и приведены ниже.

```
N           (Generations)           |           Duplicate           Hashes
100                                  |                                    0
1000                                 |                                    0
10000                                |                                    0
100000                               |                                    0
1000000 | 0
```





Заключение.

Алгоритм SHA-1 обеспечивает быстрое хеширование (O(N)), но уязвим к коллизиям, что делает его небезопасным для криптографии.

- Достоинства: Простая реализация, высокая скорость, подходит для контрольных сумм.
- Недостатки: Устаревший, незащищён от атак на коллизии (атака SHAttered, может выдать одинаковый хеш для двух разных сообщений).