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-36 Полковникова Д.Д.

Ссылка на репозиторий: https://github.com/MUCTR-IKT-CPP/Polkovnikova\_CS-36

Приняли: Пысин Максим Дмитриевич

Краснов Дмитрий Олегович

Лобанов Алексей Владимирович

Крашенинников Роман Сергеевич

Дата сдачи: 28.04.2025

Оглавление

[Описание задачи. 2](#_Toc195526846)

[Описание метода/модели 3](#_Toc195526847)

[Выполнение задачи. 6](#_Toc195526848)

[Заключение. 16](#_Toc195526849)

# Описание задачи.

В рамках лабораторной работы необходимо реализовать 1 из ниже приведенных алгоритмов хеширования:

1. [MD5](http://cryptowiki.net/index.php?title=MD5)
2. [SHA1](https://ru.bmstu.wiki/SHA-1_(Secure_Hash_Algorithm_1)#:~:text=Secure%20Hash%20Algorithm%201%20%E2%80%94%20%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC,%D0%B7%D0%BD%D0%B0%D1%87%D0%B5%D0%BD%D0%B8%D0%B5%2C%20%D0%BD%D0%B0%D0%B7%D1%8B%D0%B2%D0%B0%D0%B5%D0%BC%D0%BE%D0%B5%20%D1%82%D0%B0%D0%BA%D0%B6%D0%B5%20%D0%B4%D0%B0%D0%B9%D0%B4%D0%B6%D0%B5%D1%81%D1%82%D0%BE%D0%BC%20%D1%81%D0%BE%D0%BE%D0%B1%D1%89%D0%B5%D0%BD%D0%B8%D1%8F.)
3. [SHA2](https://m.habr.com/ru/company/selectel/blog/530262/)
4. [Стриборг](https://m.habr.com/ru/post/188152/)
5. [RIPEMD-160](https://ru.m.wikipedia.org/wiki/RIPEMD-160)

Доп вариант для тех кто хочет посложнее:

* [Luffa](https://ru.m.wikipedia.org/wiki/Luffa_(%D1%85%D0%B5%D1%88-%D1%84%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F))
* [SHA3](https://ru.m.wikipedia.org/wiki/SHA-3)

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 х 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

**Хеш-табли́ца** — структура данных, реализующая интерфейс ассоциативного массива. При этом по сути это простая таблица в которой есть столбец ключа, и столбец значения. Каждый ключ в такой таблице соответствует какому то значению и новые значения вставляются в такую таблицу всегда сопоставляясь с каким то ключом.

Для такой таблицы значений как правило реализуются следующие операции:

Вставка(k, v) – вставить в таблицу элемент с ключом

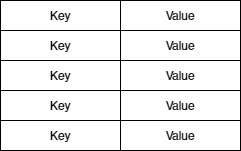
Удаление(k) – удалить из таблицы элемент с ключом

Поиск(k) – найти в таблице элемент с ключом

При этом выделяется 2 вида таких таблиц:

Set – Таблица которая хранит в себе множество значений, каждое из которых само по себе является ключом и значением одновременно.

Map – Таблица которая хранит в себе пары ключ значение в явном виде.



Хеш-таблица хранится в памяти как массив длинною некоторая M. При этом для set в таком массиве хранится напрямую значение добавленные в таблицу которое является и ключом и значением одновременно, в свою очередь для map, каждой ячейке соответствует какой то ключ, а сама ячейка хранит уже значение ассоциирующееся с этим ключом.

Как же определяется позиция записываемого значения?

Для определения позиции значения требуется использовать какую либо функцию, которая превратит наш ключ в номер ячейке в массиве значений. Самый простой вариант о котором можно подумать, это преобразования значения к числу и взятие модуля от этого значения. Т.е.:

i = int(key) % M,

где i – позиция записываемого элемента в массвие,

M – размер массива.

Стоит обратить внимание, что в предыдущей формуле M может быть различного значения и напрямую зависит от размера хранимого значения. Поэтому, если взять M достаточно большим, то количество элементов в массиве будет неподъемным ни для одного компьютера. С другой стороны, если взять M достаточно малым, то мы обнаружим, что функция будет давать одинаковый результат на совершенно разные входные данные:

Положим, что key = 25, а M = 20: тогда i = 5.

Положим, что key = 45, при том же M: тогда i = 5.

Как мы видим, результат одинаковый, а значит мы будем записывать данные в одну и ту же ячейку.

Возможно ли создать какую либо функцию, которая гарантированно вычислит уникальную позицию для любых пар элементов? Нет. Такая ситуация называется коллизией, и в рамках структуры хеш-таблицы, принимается, что могут существовать пересечения значений индекса, поэтому за индексом всегда скрывается дополнительная структура, либо список, либо дерево, либо другая хеш-таблица.

Часто используют функцию: h(k) = ((k\*A) % P) % M), где, P – случайное, простое, большое число, A – случайное, число от 1 до P – 1. Эта функция дает вероятность коллизий в пределах

Стоит обратить внимание, что в предыдущей формуле M может быть различного значения и напрямую зависит от размера хранимого значения. Поэтому, если взять M достаточно большим, то количество элементов в массиве будет неподъемным ни для одного компьютера. С другой стороны, если взять M достаточно малым, то мы обнаружим, что функция будет давать одинаковый результат на совершенно разные входные данные:

Положим, что key = 25, а M = 20: тогда i = 5.

Положим, что key = 45, при том же M: тогда i = 5.

Как мы видим, результат одинаковый, а значит мы будем записывать данные в одну и ту же ячейку.

Возможно ли создать какую либо функцию, которая гарантированно вычислит уникальную позицию для любых пар элементов? Нет. Такая ситуация называется коллизией, и в рамках структуры хеш-таблицы, принимается, что могут существовать пересечения значений индекса, поэтому за индексом всегда скрывается дополнительная структура, либо список, либо дерево, либо другая хеш-таблица.

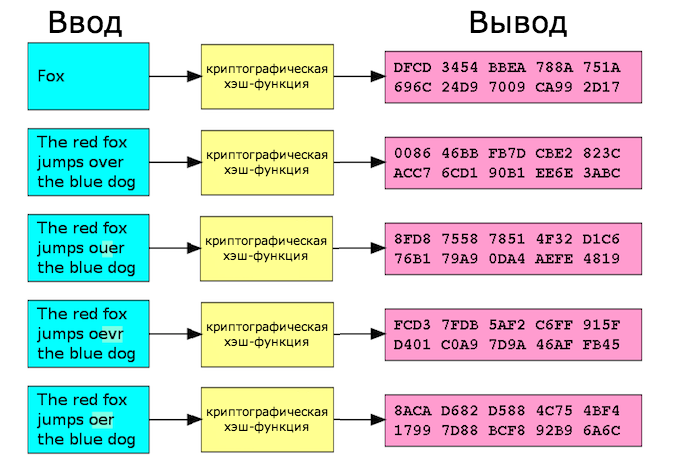
Часто используют функцию: h(k) = ((k\*A) % P) % M), где, P – случайное, простое, большое число, A – случайное, число от 1 до P – 1. Эта функция дает вероятность коллизий в пределах

Хеширование, это применение особого алгоритма к некоторым входным данным произвольного типа, преобразующего их в битовую строку установленной длинны, строка часто выводиться в форме шестнадцатеричного чиста.

Алгоритмом который применяется для хеширования называют хеш-функцией или функцией свертки.

Результат работы алгоритма хеширования называют хешем, хеш-кодом, хеш-суммой.

* при построении ассоциативных массивов;
* при поиске дубликатов в сериях наборов данных;
* при построении уникальных идентификаторов для наборов данных;
* при вычислении контрольных сумм от данных для последующего обнаружения в них ошибок, возникающих при хранении и/или передаче данных;
* при сохранении паролей в системах защиты в виде хеш-кода;
* при выработке электронной подписи;

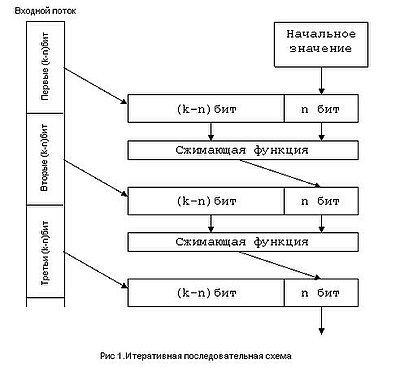
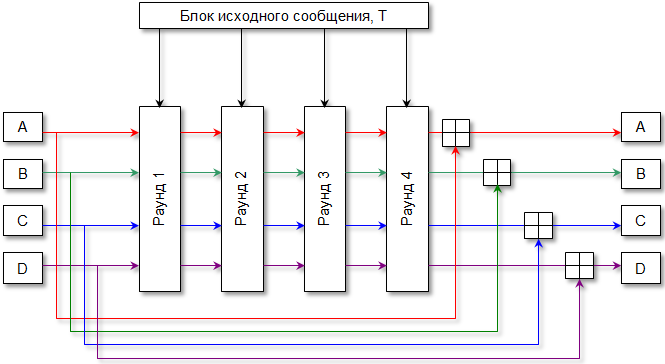


По умолчанию, базовая реализация хэш-функции должна удовлетворять следующим свойствам:

* Детерминированность – т.е. функция должна, в обязательном порядке, выдавать одинаковый вывод на одинаковы ввод.
* Скорость вычисления – функция должна быстро вычисляться, чтобы ее можно было эффективно использовать для итеративных и постоянно возникающих процессов.
* Минимальное количество коллизий. – все хеширующие функции не гарантируют полное отсутствие колизий на бесконечном наборе входных данных.

Коллизии это пересечение значений, выдаваемых хеш-функцией как результат своей работы для двух абсолютно разных вводов.

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Ввод** | **Итоговый хеш** | **Наличие коллизий** |
| 1 | Первый набор данных | F98C469D0CF7E5D466DF06D339990EE138674B6D | Коллизия |
| 2 | Второй набор данных | CF1E415CEEA860D81EC9233D626AE90EBAA823A7 | Нет коллизий |
| 3 | Третий набор данных | F98C469D0CF7E5D466DF06D339990EE138674B6D | Коллизия |

**Универсальным хешированием** называют алгоритм в котором хеш-функция выбирается случайным образом из заранее заданного набора хеш-функций что обеспечивает значительное сокращение коллизий и более равномерное хеширование.

**Идеальной хеш-функцией** называется и считается та хеш функция которая никогда не дает одинаковых результатов на различные входные данные, т.е. никогда не возникает коллизий.

Для обеспечения такого результата хеширование производиться в 2 уровня.

Разумеется есть ограничение: идеальная хешь-функция существует только для конечного набора входных данных, т.е. предполагается что после создания контейнера состав ключей уже не меняется, что будет гарантировать быстроту и однозначность доступа к данным, но вызывать трудоемкую перегенерацию при каждом добавлении.

<https://neerc.ifmo.ru/wiki/index.php?title=%D0%98%D0%B4%D0%B5%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5_%D1%85%D0%B5%D1%88%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5>

Криптографические хеш-функции

Криптографическая хэш-функция — это специальный класс хэш-функций, который имеет различные свойства, необходимые для криптографии.

**Свойство 1: Коллизионная устойчивость**

Это свойство обозначает, что для данной функции еще не было найдено данных, которые создают коллизию, в общем же смысле означает, что для поиска коллизий необходимо потратить огромное количество времени.

**Свойство 2: Устойчивость к поиску первого прообраза**

Это свойство означает, что для данной функции крайне сложно выполнить операцию поиска исходного сообщения имея итоговый хеш, причем это свойство должно соблюдаться вне зависимости от того будет ли поиск выполнятся перебором или каким-либо образом раскручиваться из итогового результата.

**Свойство 3: Устойчивость к поиску второго прообразу**

Это свойство означает, что для данной функции крайне сложно выполнить операцию поиска исходного сообщения имея итоговый хеш и исходное сообщение, дающее коллизию.

**Доказуемо безопасные**

Аналитическое доказательство выполнения приведенных ранее свойств весьма сложная задача, а само словосочетание доказумо безопасная функция означает, что задачи поиска колизии и поиска первого прообраза являются нерешаемыми за полиномиальное время и время требуемое на их решение экспоненциально зависит от размера итогового ключа.

У доказательного подхода есть определенные недостатки:

* Алгоритмы которые на данный момент являются доказуемо безопасными вычислительно сложны, что не позволяет использовать их в потоковых задачах.
* Трудоемкость разработки и обоснования доказуемо безопасной функции огромна.
* Само доказательство строиться на сведении решения задач поиска коллизий, первого и второго прообразов к задачам которые имеют требуемую сложность в среднем и худшем случаях, что потенциально делает эти функции уязвимыми.

Примеры таких функций:

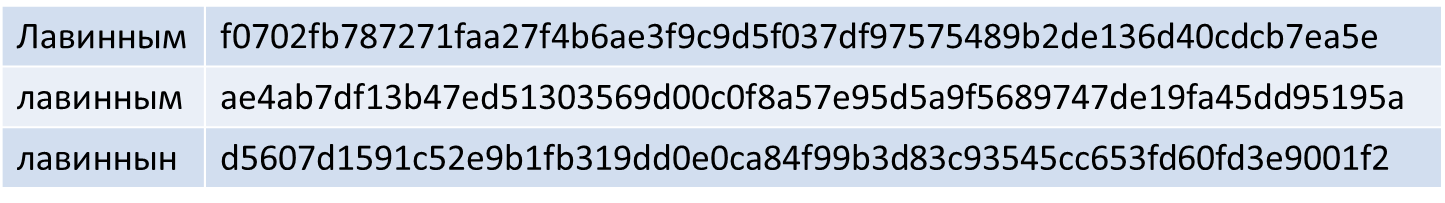
* VSH – основана на сложности задачи нахождения нетривиальных квадратных корней по модулю составного числа
* ECOH – основана на идее эллиптических кривых, задаче о сумме подмножеств и суммировании полиномов
* FSB – показано что взломать функцию так же сложно как решить задачу известную как регулярное синдромное декодирование.
* Knapsack-based hash functions – это функции которые основаны на задаче о рюкзаке.

Идеальной можно назвать ту функцию, которая отвечает следующим свойствам:

* Детерминированность;
* Высокая скорость вычисления значения хеш-функции для любого заданного сообщения;
* Невозможность сгенерировать сообщение из его хеш-значения, за исключением попыток создания всех возможных сообщений;
* Наличие лавинного эффекта;
* Невозможность найти два разных сообщения с одинаковыми хеш-значениями.
* Высокая энтропия информации на выходе.

**Лавинный эффект**

Лавинным эффектом называют ситуацию, в которой при генерации хеша для двух слабо отличающихся друг от друга изначальных набора данных, результат будет отличаться колоссально.



**Соль**

**Соль** – это дополнительные данные, которые вносятся в хеш-функцию и генерируются случайным образом. Соль делиться на статическую – одинаковая на каждый хеш и динамическую, разная для каждого хеша. Сама соль, в случае если она динамическая, как правило записывается рядом с итоговым хешом и нужна для повторного хеширования данных при поиске. Наличие соли защищает от атаки связанной с перебором по словарю возможных исходных значений (например, перебор по словарю популярных паролей), т.е. позволяет скрыть факт использования одинаковых входных значения.

**Проблемы соли**

**Малая длина соли и низкая энтропия**

Если соль имеет малую длину, злоумышленнику будет легко создать радужную таблицу, состоящую из всех возможных солей определённой длины, добавляемых к каждому вероятному паролю.

**Повторное использование соли для разных прообразов**

Хотя использование статической соли для одинаковых прообразов сделает некоторые существующие радужные таблицы бесполезными, следует заметить, что если соль статично вписана в исходный код популярного продукта, то она может быть рано или поздно извлечена, после чего на основе этой соли можно создать новую радужную таблицу.

**Атака дней рождения**

Суть атаки заключается в поиске коллизий хеш функции на основе парадокса дней рождений, суть парадокса состоит в том, что в группе состоящей из 23 и более человек, вероятность совпадения дня рождения(число, месяц) хотя бы у 2х людей превышает 50%. В итоге, идея заключается в том что бы брать набор правильных и не правильных данных с найденными коллизиями, и после выполнения операции хеширования выдавать за правильный документ, неправильный, так как их хеши совпадают

**Атака грубой силой**

Суть атаки заключается в переборе входных значений и сравнения результата хеширования с известным.

**Атака встреча посередине**

Суть атаки как и в случае с атакой дней рождения, заключается в подборе коллизий, однако подбор осуществляется разбивая поддельное сообщение на 2 части и хешируя их по отдельности некими алгоритмами hash1 и hash2, причем для второй части применяется не сам алгоритм hash2, а обратная ему функция, две половины хеша складываются и сравниваются с искомым хешом, если совпадение найдено, то по сути мы создали поддельное сообщение.

**Феш-функции**

Примеры часто используемых хеш функций:

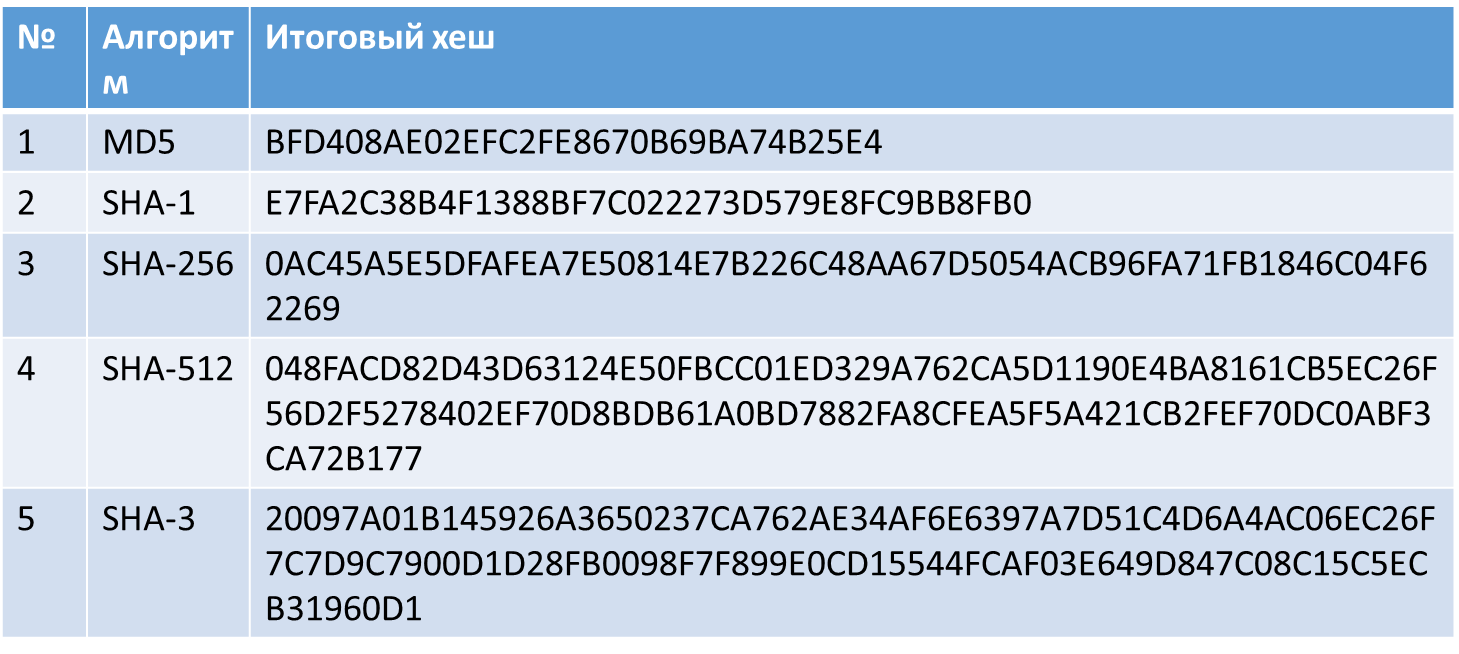
MD5 - Message Digest #5

SHA-1 - Secure Hash Algorithm 1 (1995) использует 160 бит для хранения результата

SHA-256 - Secure Hash Algorithm 2 использует 256 бит для хранения результата

SHA-512 - Secure Hash Algorithm 2 использует 512 бит для хранения результата

SHA-3 - Secure Hash Algorithm 3 новый вид алгоритма из семейства SHA



**Решение проблемы коллизий**

Используя хеш-функцию можно, составить таблицу содержащуюю пары ключ значение, где ключом будет является хеш, а значением будет является некоторое хранимое по этому ключу значение, таким образом будет получена хеш-таблица.

В таких хеш-таблицах используются простые хеш-функции, которые гораздо чаще чем криптографические хеш функции будут давать эффект коллизий.

Выделяют 2 метода для решения проблемы коллизий:

* Метод цепочек – составляется таблица итоговых хешей хранимых ключей, далее, в случае если возникает коллизия ключей, в эту таблицу записывается не один ключ, а список ключей, и в дальнейшем, при поиске данных в такой таблицы, мы сначала быстро перемещаемся по ней используя поиск по хешу, а затем ищем уже внутри списка ключей, размер которого предполагается несоизмеримом меньшим чем список хешей.
* Метод открытой адресации – составляется таблица итоговых хешей, которая хранит пары хеш – ключ, и при поиске по такой таблице, мы сначала находим нужный хеш после чего сверяемся с ключом, если он не совпадает ищем следующий подходящий хеш, и так до тех пор пока не будет найден нужный ключ, или не кончаться варианты.

В обоих случая предполагается, что если искомое значение не было найдено, то создается новая пара и записывается в таблицу.

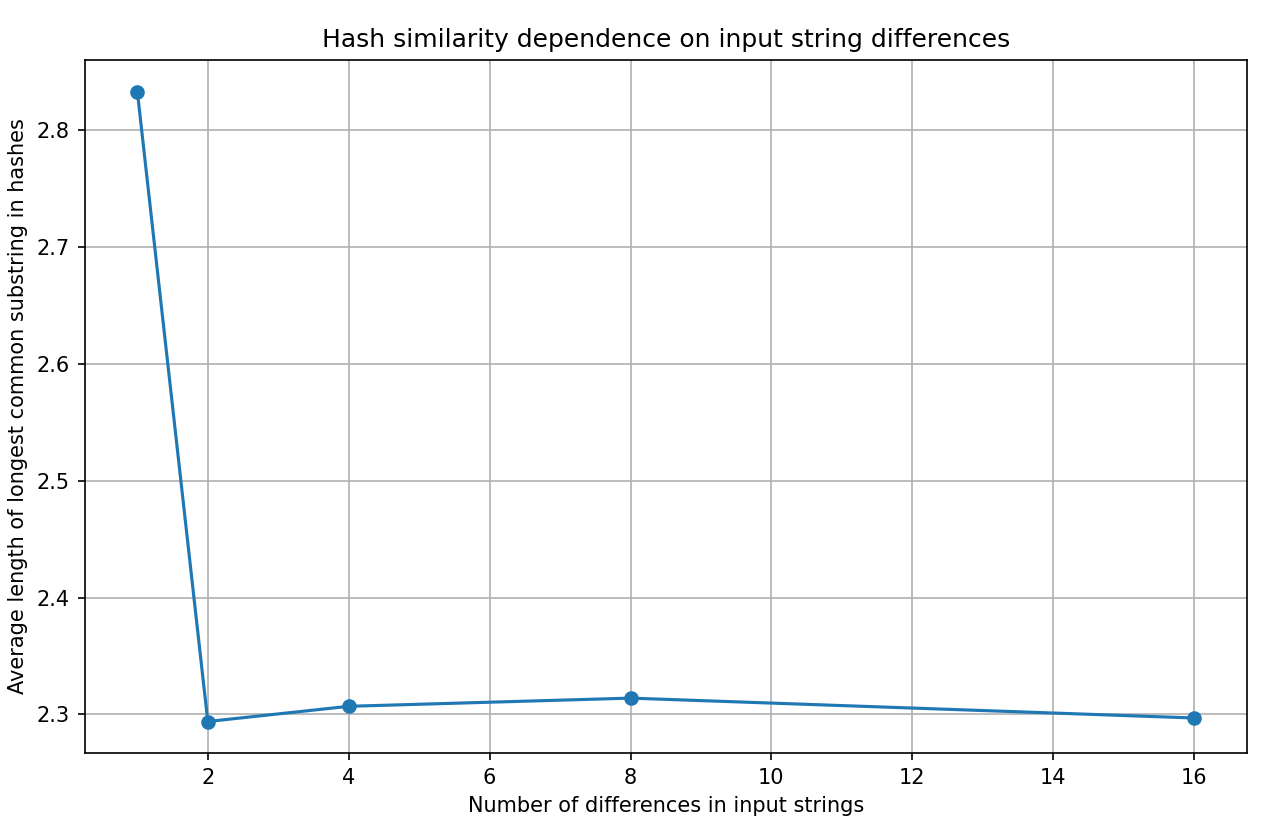
# Выполнение задачи.

В данной лабораторной работе получили следующее:

**Тест 1: Сравнение хешей строк с разным количеством отличий**

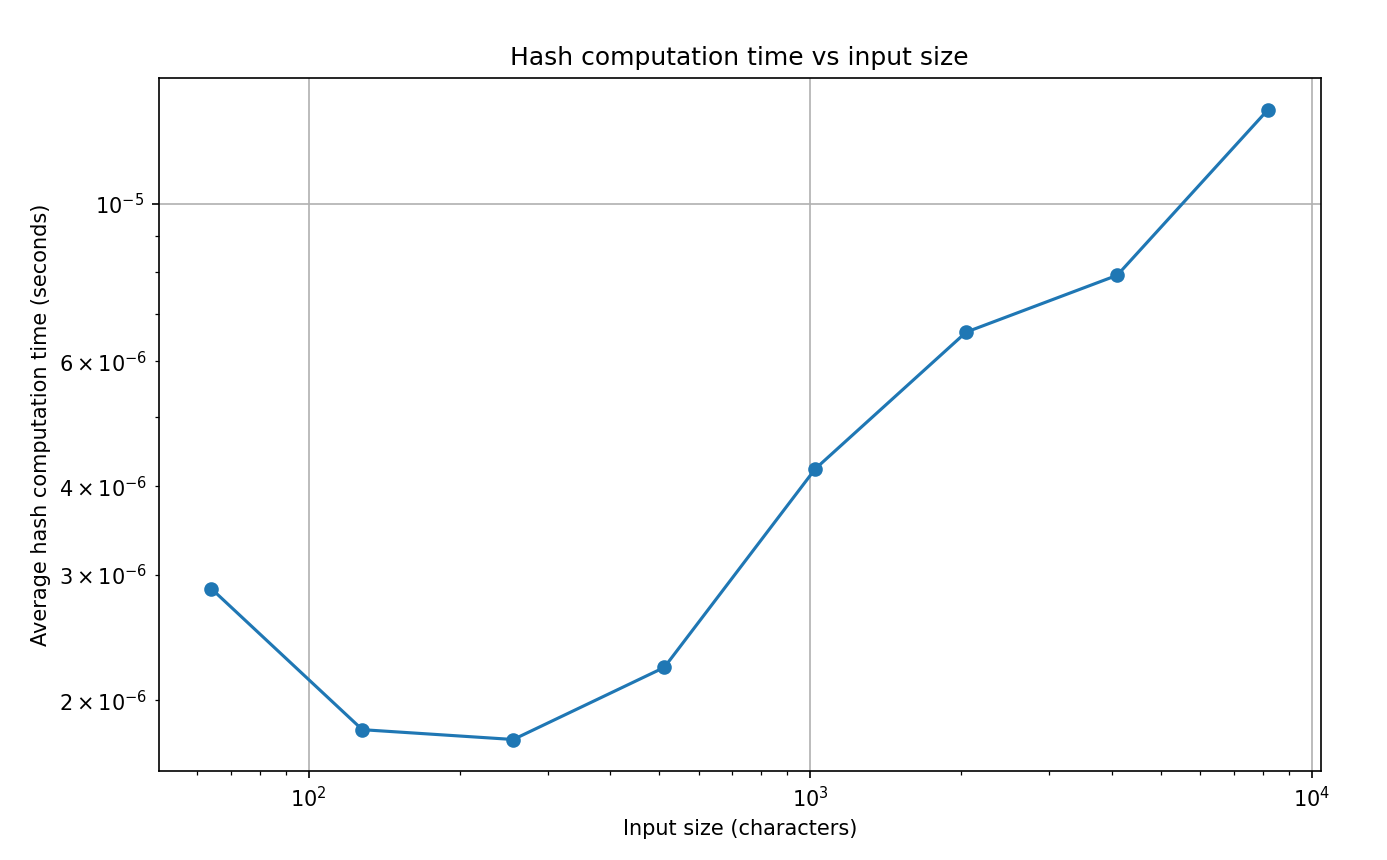
Этот тест показывает, как изменяется схожесть хешей при изменении исходных строк. Ожидается, что даже при небольших изменениях входных данных хеши будут значительно отличаться (лавинный эффект).

График покажет среднюю длину наибольшей общей подстроки в хешах для пар строк с 1, 2, 4, 8 и 16 отличиями.

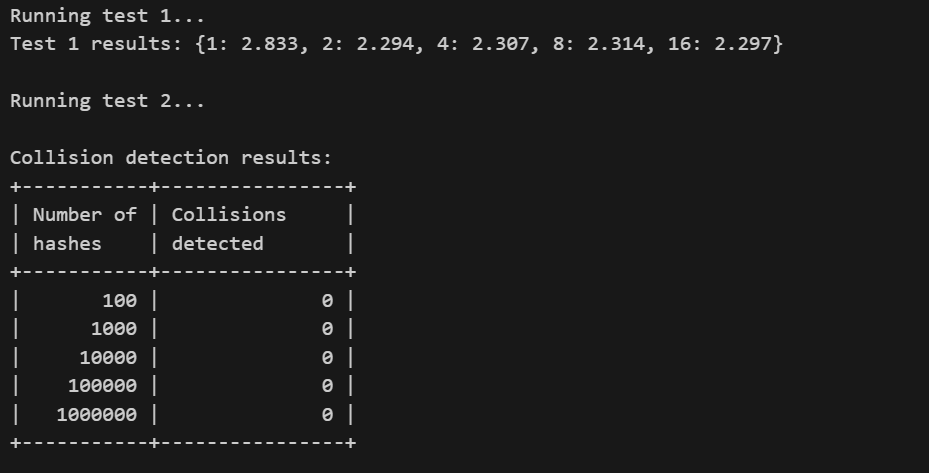
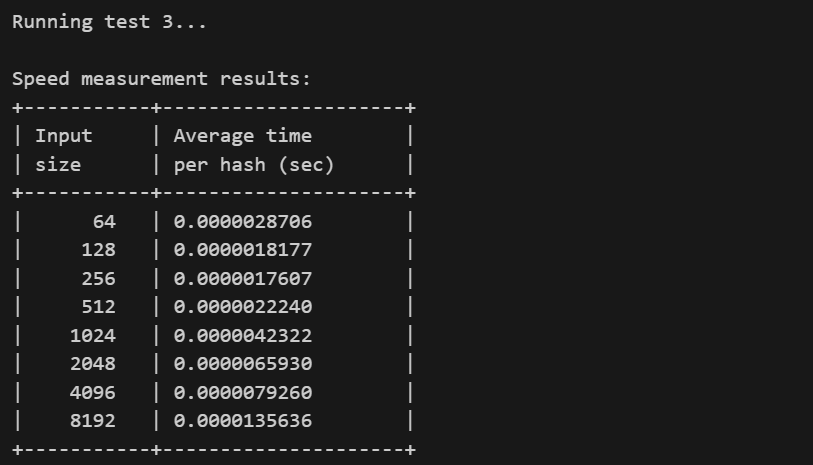
****

**Тест 2: Поиск коллизий при массовой генерации хешей**

Этот тест проверяет вероятность коллизий (случаев, когда разные входные данные дают одинаковый хеш). Для SHA-1 с его 160-битным хешем коллизии должны быть крайне редки.



Результаты будут представлены в таблице вида:

# Заключение.

**Выводы**

1. SHA-1 демонстрирует сильный лавинный эффект - даже при изменении одного символа в исходной строке хеши значительно отличаются.
2. Коллизии в SHA-1 крайне редки, что подтверждается тестами при генерации до 1 миллиона хешей.
3. Время вычисления хеша линейно зависит от размера входных данных, что соответствует ожиданиям для алгоритмов хеширования.