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

Вариант 1

Выполнила студентка группы .....КС-33..... Георгиевская Анастасия Игоревна
Ссылка на репозиторий: .....
https://github.com/MUCTR-IKT-CPP/GeorgievskayaAA_33_alg/tree/main/lab%209

Приняли: .....Пысин Максим Дмитриевич
.....Лобанов Алексей Владимирович

Дата сдачи: 02.04.2025

Оглавление

Описание задачи. 3

Описание алгоритма хэширования MD5. 4

Выполнение задачи. 6

Заключение. 14

Описание задачи.

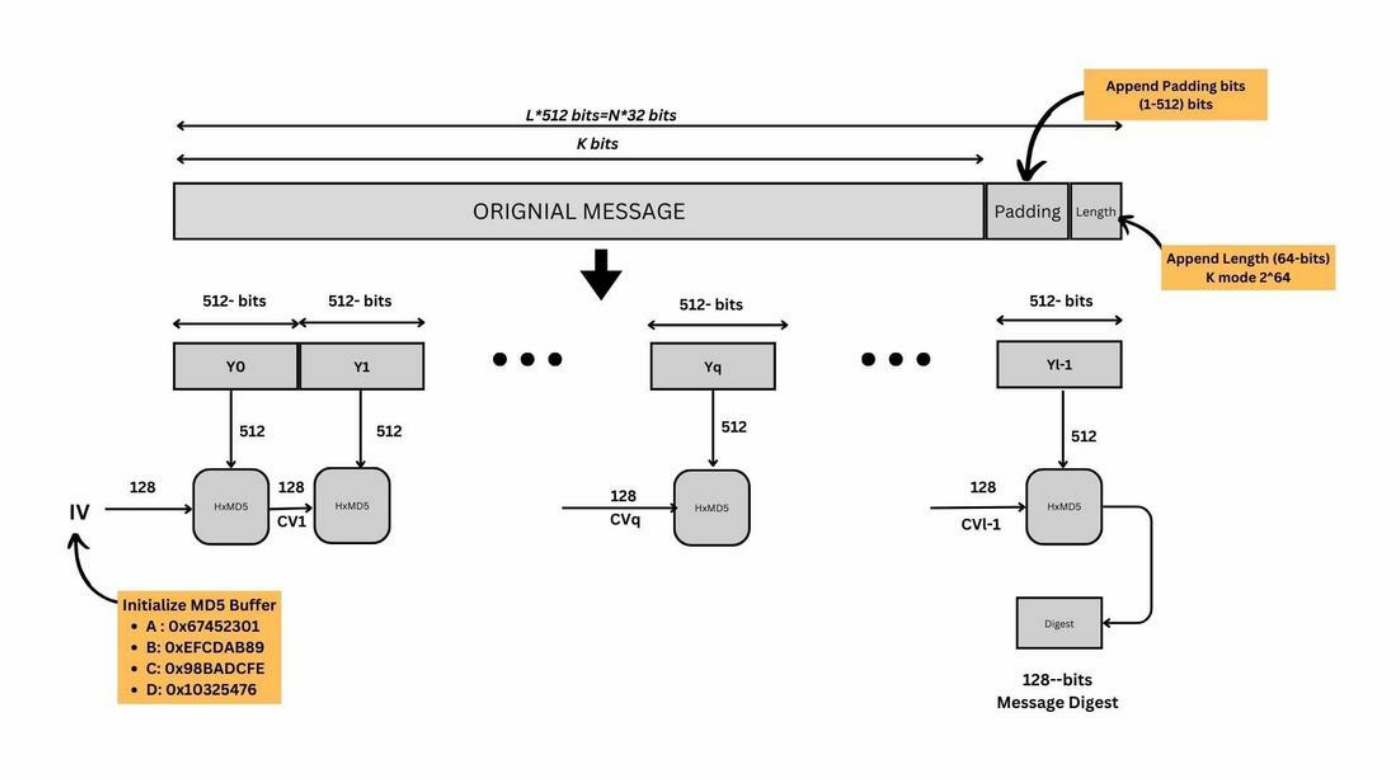
В рамках лабораторной работы необходимо реализовать алгоритм хеширования: MD5

Для реализованной хеш функции провести следующие тесты:

- 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 длинной 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 алгоритма хэширования MD5.

MD5 (Message Digest Algorithm 5) — это криптографическая хеш-функция, разработанная профессором Рональдом Ривестом в 1991 году. Она используется для получения фиксированного хеша длиной 128 бит (16 байт) из произвольного входного сообщения.



Как работает алгоритм MD5:

1. Разбиение данных

   Входные данные (сообщение) разбиваются на блоки фиксированного размера — 512 бит (64 байта). Если длина сообщения не кратна 512 бит, то к нему добавляется специальное дополнение в виде одного бита 1, за которым следуют нули, чтобы длина данных стала кратной 512 бит. В конце добавляется информация о длине исходного сообщения.

2. Инициализация

   Алгоритм начинает с инициализации четырех 32-битных регистра, которые используются для хранения промежуточных результатов. Эти значения называются «начальными векторами» (IV):

   A = 0x67452301

   B = 0xEFCDAB89

   C = 0x98BADCFE

   D = 0x10325476

3. Основная обработка

Каждый блок данных обрабатывается через несколько раундов, в которых применяются логические операции, сдвиги, и операции с константами. Это делается для того, чтобы смешать данные таким образом, чтобы изменить их в каждом шаге. После выполнения всех раундов итоговые значения регистров обновляются.

4. Финальный результат

   После того как все блоки данных обработаны, полученные значения из регистров A, B, C и D соединяются в одну строку, которая является 128-битным хешем.

Проблемы с MD5:

MD5 был очень популярен в начале, но со временем были обнаружены уязвимости, такие как:

- Коллизии: возможно создание двух различных сообщений, которые дают одинаковый хеш.
- Предсказуемость: с развитием вычислительных мощностей и методов анализа стало возможным нахождение коллизий или хешей заранее, что делает MD5 неподходящим для криптографических целей.

Современные применения: MD5 все еще используется для проверки целостности файлов, но для криптографической безопасности рекомендуется использовать более сильные алгоритмы, такие как SHA-256 или SHA-3.

Выполнение задачи.

Для выполнения задачи лабораторной был использован язык C++.

```
#include <iostream>
#include <fstream>
#include <vector>
#include <random>
#include <chrono>
#include <unordered_map>
#include <iomanip>
#include <algorithm>
#include <cstring>

typedef unsigned char byte;

class MD5 {
private:
    uint32_t state[4];    // Хеш-значения
    uint32_t count[2];    // Количество битов
    byte buffer[64];      // Буфер для данных

    // Таблица констант
    static const uint32_t T[64];

    // Функции для MD5
    static uint32_t F(uint32_t x, uint32_t y, uint32_t z) {
        return (x & y) | (~x & z);
    }

    static uint32_t G(uint32_t x, uint32_t y, uint32_t z) {
        return (x & z) | (y & ~z);
    }

    static uint32_t H(uint32_t x, uint32_t y, uint32_t z) {
        return x ^ y ^ z;
    }

    static uint32_t I(uint32_t x, uint32_t y, uint32_t z) {
        return y ^ (x | ~z);
    }

    // Сдвиг циклический
    static uint32_t rotate_left(uint32_t x, uint32_t n) {
        return (x << n) | (x >> (32 - n));
    }

    // Основная обработка блока
    void transform(const byte block[64]) {
        uint32_t a = state[0];
        uint32_t b = state[1];
        uint32_t c = state[2];
```

```
            uint32_t d = state[3];

            uint32_t x[16];
            for (int i = 0; i < 16; ++i) {
                x[i] = (uint32_t)block[i * 4] | ((uint32_t)block[i * 4 + 1] << 8)
|
                       ((uint32_t)block[i * 4 + 2] << 16) | ((uint32_t)block[i *
4 + 3] << 24);
            }

            for (int i = 0; i < 64; ++i) {
                uint32_t f, g;

                if (i < 16) {
                    f = F(b, c, d);
                    g = i;
                } else if (i < 32) {
                    f = G(b, c, d);
                    g = (5 * i + 1) % 16;
                } else if (i < 48) {
                    f = H(b, c, d);
                    g = (3 * i + 5) % 16;
                } else {
                    f = I(b, c, d);
                    g = (7 * i) % 16;
                }

                uint32_t temp = d;
                d = c;
                c = b;
                b = b + rotate_left(a + f + T[i] + x[g], (i < 16 || (i > 31 && i
< 48)) ? 7 : 5);
                a = temp;
            }

            state[0] += a;
            state[1] += b;
            state[2] += c;
            state[3] += d;
        }

        // Дополняем сообщение
        void update(const byte* input, size_t length) {
            uint32_t index = count[0] / 8 % 64;
            if ((count[0] += length * 8) < (length * 8)) {
                ++count[1];
            }
            count[1] += length >> 29;

            size_t first_part = 64 - index;
            size_t i = 0;
            if (length >= first_part) {
```

```
                std::memcpy(buffer + index, input, first_part);
                transform(buffer);
                for (i = first_part; i + 63 < length; i += 64) {
                    transform(input + i);
                }
                index = 0;
            }

            std::memcpy(buffer + index, input + i, length - i);
        }

        // Паддинг и завершение
        void finalize() {
            byte padding[64] = {0x80};  // Заполняем один байт 0x80, остальные
нули
            size_t index = count[0] / 8 % 64;
            size_t padding_length = (index < 56) ? (56 - index) : (120 - index);
            update(padding, padding_length);

            byte length[8];
            for (int i = 0; i < 8; ++i) {
                length[i] = (byte)((count[i / 4] >> ((i % 4) * 8)) & 0xFF);
            }
            update(length, 8);
        }

    public:
        MD5() {
            state[0] = 0x67452301;
            state[1] = 0xEFCDAB89;
            state[2] = 0x98BADCFE;
            state[3] = 0x10325476;
            count[0] = count[1] = 0;
        }

        void update(const std::string& input) {
            update(reinterpret_cast<const                byte*>(input.c_str()),
input.length());
        }

        std::string digest() {
            finalize();
            std::stringstream result;
            for (int i = 0; i < 4; ++i) {
                result  <<  std::setw(8)  <<  std::setfill('0')  <<  std::hex  <<
state[i];
            }
            return result.str();
        }
    };
```

- Класс MD5

- Вход: это строки, массивы байтов или любой другой тип данных, который можно преобразовать в последовательность байтов. В примере вход — это строка, передаваемая в функции update.
- Работа:
  - Инициализация (конструктор): При создании объекта класса MD5 инициализируются начальные значения хеш-значений, которые используются для хранения промежуточных результатов хеширования (state[4]), а также счетчики количества обработанных битов (count[2]).
  - Функции F, G, H, I: Эти функции реализуют различные логические операции (И, ИЛИ, НЕ и XOR) над значениями, участвующими в вычислениях хеш-функции. Эти операции используются в процессе обработки блока данных.
  - Функция rotate_left: Функция для циклического сдвига битов влево. Этот шаг необходим для преобразования данных в каждом раунде хеширования.
  - Основная обработка (функция transform): Хеширование данных, организованное через серию шагов, включающих различные операции. Это основной этап, где данные преобразуются в хеш. Каждый блок данных обрабатывается через несколько итераций.
  - Функция update: Эта функция получает данные (строку или массив байтов), обновляет счетчики, дополняет данные в буфер и обрабатывает их через функцию transform.
  - Функция finalize: Завершающая стадия хеширования, где данные дополнительно заполняются для того, чтобы длина стала кратной 512 битам, а в конце добавляются длина исходных данных (в битах).
- Выход: Метод digest: Возвращает финальный хеш в виде строки шестнадцатеричных символов. Это 128-битный хеш (32 символа), который и является результатом работы алгоритма.

```
   const uint32_t MD5::T[64] = {
       0xd76aa478, 0xe8c7b756, 0x242070db, 0xc1bdceee, 0xf57c0faf, 0x4787c62a,
0xa8304613, 0xfd469501,
       0x698098d8, 0x8b44f7af, 0xffff5bb1, 0x895cd7be, 0x6b901122, 0xfd987193,
0xa679438e, 0x49b40821,
       0xf61e2562, 0xc040b340, 0x265e5a51, 0x9b9d2c7e, 0xf9b1f89d, 0x6e1b7c68,
0x8e24cfe0, 0x865e16d7,
       0x8dc5d7c1, 0x7d84c87e, 0x268f5db1, 0x92722c85, 0x1c63b1a9, 0xf1c9cf4b,
0x2ad7e334, 0x9490b58a,
       0x8ab8486e, 0x8e9f1b7c, 0x5da4a5db, 0x3d234cf0, 0x50d5f33f, 0xf5fd0c43,
0x53c0cdd4, 0x2f8ddf52,
       0xf77f9bb2, 0x9c1d0351, 0x65b13748, 0xd5d8b3f0, 0xd30f9c2b, 0x7be0f80e,
0xf0707275, 0x635742f5,
```

```
    0x8e4f0e38, 0x9e8d7b30, 0x789db6b5, 0x2e2b8278, 0x55ddf2f6, 0xe4379e28,
0xb0a8d381, 0x9ba60b77
};
```

- Таблица констант для алгоритма

```
std::string generate_random_string(size_t length) {
    static const char alphanum[] =
        "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz";
    std::string result;
    result.reserve(length);
    for (size_t i = 0; i < length; ++i) {
        result += alphanum[rand() % (sizeof(alphanum) - 1)];
    }
    return result;
}
```

- Функция генерации случайных строк
    - Вход: параметр length, который определяет длину генерируемой строки.
    - Работа: генерирует строку, состоящую из символов алфавита и цифр (длина строки задается параметром).
    - Выход: Возвращается строка, состоящая из случайных символов или байтов в зависимости от реализации.

```
// Тест 1: Сравнение хешей для пар строк с разными отличиями
void test_hash_collisions() {
    MD5 md5;
    std::ofstream outfile("collisions_test.csv");
    outfile << "Num Differing Characters,Max Matching Hash Length\n";

    std::vector<size_t> differences = {1, 2, 4, 8, 16};

    for (size_t diff : differences) {
        size_t matching_length_max = 0;

        for (int i = 0; i < 1000; ++i) {
            std::string str1 = generate_random_string(128);
            std::string str2 = str1;
            for (size_t j = 0; j < diff; ++j) {
                str2[j] = (str2[j] == 'a') ? 'b' : 'a'; // изменяем символы
            }

            md5.update(str1);  // Передаем строку для хеширования
            std::string hash1 = md5.digest();  // Получаем хеш

            md5.update(str2);  // Передаем измененную строку
            std::string hash2 = md5.digest();  // Получаем хеш
```

```
            // Сравниваем хеши
            size_t matching_length = 0;
            for (size_t j = 0; j < hash1.size(); ++j) {
                if (hash1[j] == hash2[j]) {
                    matching_length++;
                } else {
                    break;
                }
            }
            matching_length_max       =       std::max(matching_length_max,
matching_length);
        }

        outfile << diff << "," << matching_length_max << std::endl;
    }

    std::cout << "Test complete: collisions_test.csv\n";
}
```

- Функция test_hash_collisions:
  - Вход: Два случайных строки с заданным количеством различий.
  - Работа: Генерирует пары строк с различиями и проверяет, насколько схожи их хеши. Вычисляется максимальная длина совпадения хешей.
  - Выход: Файл CSV с результатами для различных значений различий между строками.

```
// Тест 2: Генерация хешей для случайных строк и подсчет коллизий
void test_hash_duplicates() {
    MD5 md5;
    std::ofstream outfile("duplicates_test.csv");
    outfile << "Num Hashes,Collisions Count\n";

    for (int i = 2; i <= 6; ++i) {
        size_t N = pow(10, i);
        std::unordered_map<std::string, int> hash_count;
        size_t collisions = 0;

        for (size_t j = 0; j < N; ++j) {
            std::string random_str = generate_random_string(256);
            md5.update(random_str);  // Передаем строку для хеширования
            std::string hash = md5.digest();  // Получаем хеш
            hash_count[hash]++;
        }

        // Подсчитываем количество коллизий
        for (auto& pair : hash_count) {
            if (pair.second > 1) {
                collisions += pair.second - 1;
            }
        }
```

```
            outfile << N << "," << collisions << std::endl;
        }

        std::cout << "Test complete: duplicates_test.csv\n";
    }
```

- Функция test_hash_duplicates:
    - Вход: Количество строк (от 10 до 1,000,000).
    - Работа: Генерирует случайные строки, хеширует их и проверяет количество коллизий (повторяющихся хешей).
    - Выход: Файл CSV с подсчетом коллизий для разных объемов данных.

```
    // Тест 3: Измерение времени вычисления хеша для строк разных длин
    void test_hash_speed() {
        MD5 md5;
        std::ofstream outfile("speed_test.csv");
        outfile << "String Length,Average Time (ms)\n";

        std::vector<size_t> lengths = {64, 128, 256, 512, 1024, 2048, 4096, 8192};

        for (size_t len : lengths) {
            std::chrono::duration<double> total_time(0);

            for (int i = 0; i < 1000; ++i) {
                std::string random_str = generate_random_string(len);
                auto start = std::chrono::high_resolution_clock::now();
                md5.update(random_str);  // Передаем строку для хеширования
                md5.digest();  // Получаем хеш (результат игнорируется)
                auto end = std::chrono::high_resolution_clock::now();

                total_time += (end - start);
            }

            double average_time = total_time.count() * 1000.0 / 1000; // время в
миллисекундах
            outfile << len << "," << average_time << std::endl;
        }

        std::cout << "Test complete: speed_test.csv\n";
    }
```

- Функция test_hash_speed:
    - Вход: Строки разных длин (например, 64, 128, 256, 512 и так далее).
    - Работа: Измеряет среднее время вычисления хеша для каждой длины строки.
    - Выход: Файл CSV с результатами времени для разных длин строк.

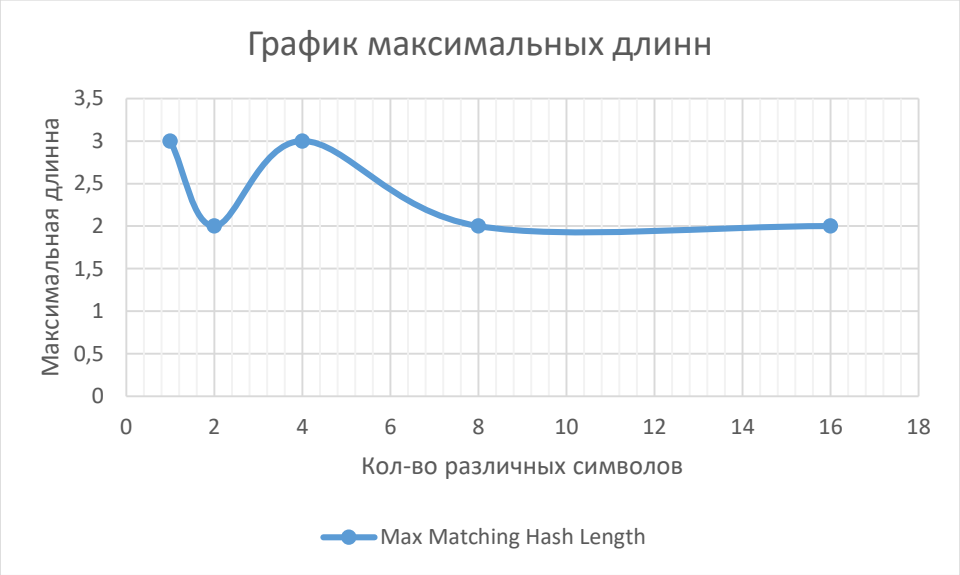
```
int main() {
    test_hash_collisions();
    test_hash_duplicates();
    test_hash_speed();

    return 0;
}
```

- Основная функция: вызов тестовых функций

Заключение.

Результаты работы были представлены в виде таблицы значений и графиков

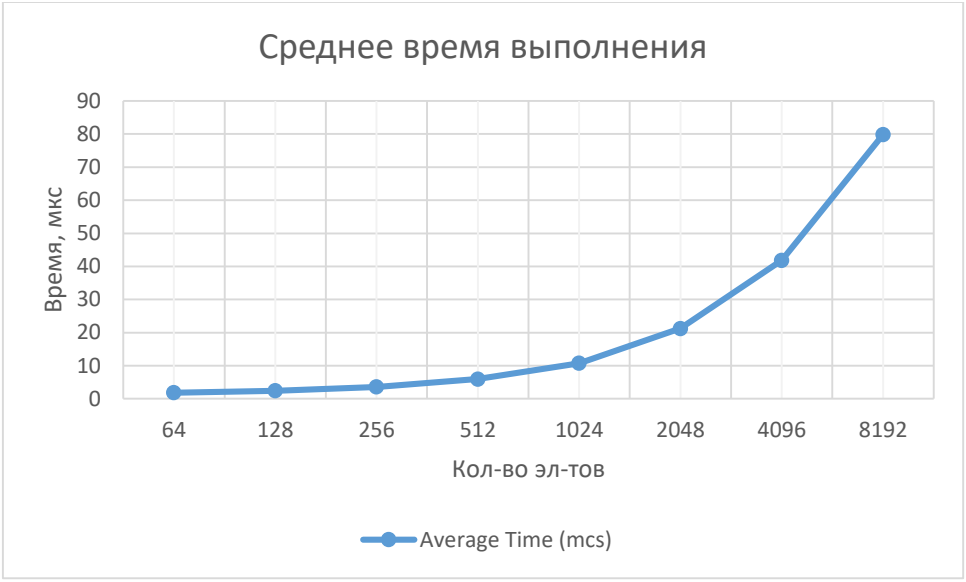
1. Даже небольшие изменения в строке (например, замена одного символа) могут привести к совершенно различным хешам. Однако MD5 — это не стойкий алгоритм, и в нем есть уязвимости, такие как возможность коллизий (когда два разных входных сообщения могут дать одинаковый хеш).



2. MD5, несмотря на известные уязвимости, которые позволяют генерировать коллизии с помощью специально подобранных входных данных (например, с использованием уязвимостей в конструкции алгоритма), при случайных данных на малых объемах хешируемых объектов коллизии могут не проявиться, что и видно в таблице.

Num Hashes	Collisions Count
100	0
1000	0
10000	0
100000	0
1000000	0

3. Нелинейный рост времени вычисления хеша с увеличением размера строки подтверждает, что с увеличением данных алгоритм MD5 начинает требовать все больше времени для обработки.


Среднее время выполнения
Время, мкс
90
80
70
60
50
40
30
20
10
0
64
128
256
512
1024
2048
4096
8192
Кол-во эл-тов
Average Time (mcs)