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-30 Джотян Давид Араикович

Ссылка на репозиторий: https://github.com/MUCTR-IKT-CPP/DzhotyanDA\_30/tree/main/SHA\_256

Приняли: Пысин Максим Дмитриевич

Краснов Дмитрий Олегович

Лобанов Алексей Владимирович

Крашенинников Роман Сергеевич

Дата сдачи: 28.04.2025

Оглавление

[Описание задачи. 2](#_Toc63548272)

[Описание метода/модели. 2](#_Toc63548273)

[Выполнение задачи. 2](#_Toc63548274)

[Заключение. 2](#_Toc63548275)

# Описание задачи.

В рамках лабораторной работы необходимо реализовать алгоритм хеширования SHA-256.  
Для реализованной хэш функции провести следующие тесты:

* Cгенерировать 1000 пар строк длинной 128 символов отличающихся друг от друга 1,2,4,8,16 символов и сравнить хэши для пар между собой, проведя поиск одинаковых последовательностей символов в хэшах 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) генераций хэшей для случайно сгенерированных строк длинно 256 символов, и выполнить поиск одинаковых хэшей в итоговом наборе данных, результаты привести в таблице где первая колонка это N генераций, а вторая таблица наличие и кол-во одинаковых хэшей, если такие были.
* Провести по 1000 генераций хэша 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 хэша от размера входных данных

# Описание метода/модели.

SHA-256 (Secure Hash Algorithm 256) — это алгоритм криптографического хэширования, который превращает любое сообщение в строку из 256 бит (32 байта).

Шаги алгоритма:

1. **Подготовка сообщения**
   1. Преобразовываем строку в двоичный код.
   2. Затем добавляем ‘1’
   3. Дополняем код нулями, пока данные не станут равны 512 бит.
   4. Дописывается 64-битное число (длина оригинального сообщения).
2. **Разбиение на блоки**
   1. Сообщение делится на блоки по 512 бит.
3. **Инициализация переменных (h0,…,h7) и 64 констант К**

h0 = 6a09e667

h1 = bb67ae85

h2 = 3c6ef372

h3 = a54ff53a

h4 = 510e527f

h5 = 9b05688c

h6 = 1f83d9ab

h7 = 5be0cd19

1. **Обработка каждого 512-битного блока**
   1. Создание массива слов.
      1. Первые 16 слов (w[0],…,w[15]) – просто нарезаются напрямую из блока.
      2. Следующие 48 слов (w[16],…,w[63]) рассчитываются по формуле.

w[i] = σ1(w[i-2]) + w[i-7] + σ0(w[i-15]) + w[i-16]

σ0(x) = (x >>> 7) ⊕ (x >>> 18) ⊕ (x >> 3)

σ1(x) = (x >>> 17) ⊕ (x >>> 19) ⊕ (x >> 10)

* 1. Инициализация рабочих переменных

a = h0

b = h1

c = h2

d = h3

e = h4

f = h5

g = h6

h = h7

* 1. Основной цикл на 64 шага.
     1. Вычисляются

S1 = (e >>> 6) ⊕ (e >>> 11) ⊕ (e >>> 25)

ch = (e AND f) ⊕ ((NOT e) AND g)

temp1 = h + S1 + ch + k[i] + w[i]

S0 = (a >>> 2) ⊕ (a >>> 13) ⊕ (a >>> 22)

maj = (a AND b) ⊕ (a AND c) ⊕ (b AND c)

temp2 = S0 + maj

* + 1. Обновляются переменные

h = g

g = f

f = e

e = d + temp1

d = c

c = b

b = a

a = temp1 + temp2

* 1. Обновление хэш-значений (после всех 64 итераций)

h0 = h0 + a

h1 = h1 + b

h2 = h2 + c

h3 = h3 + d

h4 = h4 + e

h5 = h5 + f

h6 = h6 + g

h7 = h7 + h

1. **Формирование финального хэша**

Все h0..h7 объединяются в один 256-битный результат.

# Выполнение задачи.

Файл SHA\_256.cs

using System.Text;

namespace lab9\_algo

{

internal class SHA\_256

{

private uint h0 = 0x6a09e667;

private uint h1 = 0xbb67ae85;

private uint h2 = 0x3c6ef372;

private uint h3 = 0xa54ff53a;

private uint h4 = 0x510e527f;

private uint h5 = 0x9b05688c;

private uint h6 = 0x1f83d9ab;

private uint h7 = 0x5be0cd19;

private readonly uint[] K =

[

0x428a2f98, 0x71374491, 0xb5c0fbcf, 0xe9b5dba5, 0x3956c25b, 0x59f111f1, 0x923f82a4, 0xab1c5ed5,

0xd807aa98, 0x12835b01, 0x243185be, 0x550c7dc3, 0x72be5d74, 0x80deb1fe, 0x9bdc06a7, 0xc19bf174,

0xe49b69c1, 0xefbe4786, 0x0fc19dc6, 0x240ca1cc, 0x2de92c6f, 0x4a7484aa, 0x5cb0a9dc, 0x76f988da,

0x983e5152, 0xa831c66d, 0xb00327c8, 0xbf597fc7, 0xc6e00bf3, 0xd5a79147, 0x06ca6351, 0x14292967,

0x27b70a85, 0x2e1b2138, 0x4d2c6dfc, 0x53380d13, 0x650a7354, 0x766a0abb, 0x81c2c92e, 0x92722c85,

0xa2bfe8a1, 0xa81a664b, 0xc24b8b70, 0xc76c51a3, 0xd192e819, 0xd6990624, 0xf40e3585, 0x106aa070,

0x19a4c116, 0x1e376c08, 0x2748774c, 0x34b0bcb5, 0x391c0cb3, 0x4ed8aa4a, 0x5b9cca4f, 0x682e6ff3,

0x748f82ee, 0x78a5636f, 0x84c87814, 0x8cc70208, 0x90befffa, 0xa4506ceb, 0xbef9a3f7, 0xc67178f2

];

internal string GetHash(string input)

{

string binaryString = ToBinary(input, Encoding.ASCII);

int inputBitLength = binaryString.Length; // длина в битах до паддинга

binaryString += "1";

// Дополняем нулями до 448 бит (оставляем место для длины в 64 бита)

while (binaryString.Length % 512 != 448)

{

binaryString += "0";

}

// Теперь добавляем длину исходного сообщения (до паддинга) как 64-битное число

string length64bit = Convert.ToString(inputBitLength, 2).PadLeft(64, '0');

binaryString += length64bit;

// Разбиваем на блоки по 512 бит

uint[] w = new uint[64];

// Заполняем первые 16 слов w[0..15]

for (int i = 0; i < 16; i++)

{

w[i] = Convert.ToUInt32(binaryString.Substring(i \* 32, 32), 2);

}

// Генерация оставшихся слов w[16..63]

for (int i = 16; i < 64; i++)

{

uint s0 = RotateRight(w[i - 15], 7) ^ RotateRight(w[i - 15], 18) ^ (w[i - 15] >> 3);

uint s1 = RotateRight(w[i - 2], 17) ^ RotateRight(w[i - 2], 19) ^ (w[i - 2] >> 10);

w[i] = w[i - 16] + s0 + w[i - 7] + s1;

}

uint a = h0;

uint b = h1;

uint c = h2;

uint d = h3;

uint e = h4;

uint f = h5;

uint g = h6;

uint h = h7;

// Основной цикл SHA-256

for (int i = 0; i < K.Length; i++)

{

uint S1 = RotateRight(e, 6) ^ RotateRight(e, 11) ^ RotateRight(e, 25);

uint ch = (e & f) ^ ((~e) & g);

uint temp1 = h + S1 + ch + K[i] + w[i];

uint S0 = RotateRight(a, 2) ^ RotateRight(a, 13) ^ RotateRight(a, 22);

uint maj = (a & b) ^ (a & c) ^ (b & c);

uint temp2 = S0 + maj;

h = g;

g = f;

f = e;

e = d + temp1;

d = c;

c = b;

b = a;

a = temp1 + temp2;

}

h0 += a;

h1 += b;

h2 += c;

h3 += d;

h4 += e;

h5 += f;

h6 += g;

h7 += h;

return $"{h0:x8}{h1:x8}{h2:x8}{h3:x8}{h4:x8}{h5:x8}{h6:x8}{h7:x8}";

}

private static string ToBinary(string text, Encoding encoding)

{

byte[] bytes = encoding.GetBytes(text);

StringBuilder sb = new StringBuilder();

foreach (byte b in bytes)

{

sb.Append(Convert.ToString(b, 2).PadLeft(8, '0'));

}

return sb.ToString();

}

private static string FormatBinaryString(string binary)

{

StringBuilder sb = new StringBuilder();

for (int i = 0; i < binary.Length; i += 8)

{

if (i > 0 && (i % 64 == 0)) sb.AppendLine();

else if (i > 0) sb.Append(" ");

sb.Append(binary.Substring(i, Math.Min(8, binary.Length - i)));

}

return sb.ToString();

}

private static uint RotateRight(uint x, int n)

{

return (x >> n) | (x << (32 - n));

}

}

}

Файл Program.cs

using System.Diagnostics;

namespace lab9\_algo

{

class Program

{

static void Main()

{

Test1();

Test2();

Test3();

}

static void Test1()

{

SHA\_256 sha\_256 = new();

const int N = 1000;

const int STRING\_LENGTH = 128;

int[] differentCharactersCount = [1, 2, 4, 8, 16];

string[] generatedStrings = new string[N];

string[] generatedStringOneSymbolDifference = new string[N];

string[] generatedStringTwoSymbolsDifference = new string[N];

string[] generatedStringFourSymbolsDifference = new string[N];

string[] generatedStringEightSymbolsDifference = new string[N];

string[] generatedStringSixteenSymbolsDifference = new string[N];

string[] hashOriginalStrings = new string[N];

string[] hashOneSymbolDifference = new string[N];

string[] hashTwoSymbolsDifference = new string[N];

string[] hashFourSymbolsDifference = new string[N];

string[] hashEightSymbolsDifference = new string[N];

string[] hashSixteenSymbolsDifference = new string[N];

int maxSubstringLengthOneSymbolDifference = 0;

int maxSubstringLengthTwoSymbolsDifference = 0;

int maxSubstringLengthFourSymbolsDifference = 0;

int maxSubstringLengthEightSymbolsDifference = 0;

int maxSubstringLengthSixteenSymbolsDifference = 0;

// Генерация строк

for (int i = 0; i < N; i++)

{

string generatedStr = GenerateString(STRING\_LENGTH);

generatedStrings[i] = generatedStr;

}

// Замена символов в строках

for (int i = 0; i < N; i++)

{

string originalStr = generatedStrings[i];

foreach (var count in differentCharactersCount)

{

char[] strChars = originalStr.ToCharArray();

int length = strChars.Length;

int step = length / count;

for (int j = 0; j < count; j++)

{

int indexToChange = j \* step;

if (indexToChange >= length)

indexToChange = length - 1;

char symbol = strChars[indexToChange];

if (symbol == 'z')

{

symbol = (char)(symbol - 1);

}

else if (symbol >= 'a' && symbol < 'z')

{

symbol = (char)(symbol + 1);

}

else if (symbol == 'Z')

{

symbol = (char)(symbol - 1);

}

else if (symbol >= 'A' && symbol < 'Z')

{

symbol = (char)(symbol + 1);

}

strChars[indexToChange] = symbol;

}

switch (count)

{

case 1:

generatedStringOneSymbolDifference[i] = new string(strChars);

break;

case 2:

generatedStringTwoSymbolsDifference[i] = new string(strChars);

break;

case 4:

generatedStringFourSymbolsDifference[i] = new string(strChars);

break;

case 8:

generatedStringEightSymbolsDifference[i] = new string(strChars);

break;

case 16:

generatedStringSixteenSymbolsDifference[i] = new string(strChars);

break;

}

}

}

// Вычисление хешей

for (int i = 0; i < N; i++)

{

hashOriginalStrings[i] = sha\_256.GetHash(generatedStrings[i]);

hashOneSymbolDifference[i] = sha\_256.GetHash(generatedStringOneSymbolDifference[i]);

hashTwoSymbolsDifference[i] = sha\_256.GetHash(generatedStringTwoSymbolsDifference[i]);

hashFourSymbolsDifference[i] = sha\_256.GetHash(generatedStringFourSymbolsDifference[i]);

hashEightSymbolsDifference[i] = sha\_256.GetHash(generatedStringEightSymbolsDifference[i]);

hashSixteenSymbolsDifference[i] = sha\_256.GetHash(generatedStringSixteenSymbolsDifference[i]);

}

// Нахождение максимальных длин одинаковых подстрок

for (int i = 0; i < N; i++)

{

string originalHash = hashOriginalStrings[i];

string oneSymbolDifferenceHash = hashOneSymbolDifference[i];

string twoSymbolsDifferenceHash = hashTwoSymbolsDifference[i];

string fourSymbolsDifferenceHash = hashFourSymbolsDifference[i];

string eightSymbolsDifferenceHash = hashEightSymbolsDifference[i];

string sixteenSymbolsDifferenceHash = hashSixteenSymbolsDifference[i];

int length = originalHash.Length;

int lengthOne = 0;

int lengthTwo = 0;

int lengthFour = 0;

int lengthEight = 0;

int lengthSixteen = 0;

for (int j = 0; j < length; j++)

{

// 1 отличие в строках

if (originalHash[j] == oneSymbolDifferenceHash[j])

{

lengthOne++;

}

else

{

if (lengthOne > maxSubstringLengthOneSymbolDifference)

{

maxSubstringLengthOneSymbolDifference = lengthOne;

}

lengthOne = 0;

}

if (lengthOne > maxSubstringLengthOneSymbolDifference)

{

maxSubstringLengthOneSymbolDifference = lengthOne;

}

// 2 отличия в строках

if (originalHash[j] == twoSymbolsDifferenceHash[j])

{

lengthTwo++;

}

else

{

if (lengthTwo > maxSubstringLengthTwoSymbolsDifference)

{

maxSubstringLengthTwoSymbolsDifference = lengthTwo;

}

lengthTwo = 0;

}

if (lengthTwo > maxSubstringLengthTwoSymbolsDifference)

{

maxSubstringLengthTwoSymbolsDifference = lengthTwo;

}

// 4 отличия в строках

if (originalHash[j] == fourSymbolsDifferenceHash[j])

{

lengthFour++;

}

else

{

if (lengthFour > maxSubstringLengthFourSymbolsDifference)

{

maxSubstringLengthFourSymbolsDifference = lengthFour;

}

lengthFour = 0;

}

if (lengthFour > maxSubstringLengthFourSymbolsDifference)

{

maxSubstringLengthFourSymbolsDifference = lengthFour;

}

// 8 отличий в строках

if (originalHash[j] == eightSymbolsDifferenceHash[j])

{

lengthEight++;

}

else

{

if (lengthEight > maxSubstringLengthEightSymbolsDifference)

{

maxSubstringLengthEightSymbolsDifference = lengthEight;

}

lengthEight = 0;

}

if (lengthEight > maxSubstringLengthEightSymbolsDifference)

{

maxSubstringLengthEightSymbolsDifference = lengthEight;

}

// 16 отличий в строках

if (originalHash[j] == sixteenSymbolsDifferenceHash[j])

{

lengthSixteen++;

}

else

{

if (lengthSixteen > maxSubstringLengthSixteenSymbolsDifference)

{

maxSubstringLengthSixteenSymbolsDifference = lengthSixteen;

}

lengthSixteen = 0;

}

if (lengthSixteen > maxSubstringLengthSixteenSymbolsDifference)

{

maxSubstringLengthSixteenSymbolsDifference = lengthSixteen;

}

}

}

Console.WriteLine($"Максимальная длина одинаковых подстрок для 1 отличия: {maxSubstringLengthOneSymbolDifference}");

Console.WriteLine($"Максимальная длина одинаковых подстрок для 2 отличий: {maxSubstringLengthTwoSymbolsDifference}");

Console.WriteLine($"Максимальная длина одинаковых подстрок для 4 отличий: {maxSubstringLengthFourSymbolsDifference}");

Console.WriteLine($"Максимальная длина одинаковых подстрок для 8 отличий: {maxSubstringLengthEightSymbolsDifference}");

Console.WriteLine($"Максимальная длина одинаковых подстрок для 16 отличий: {maxSubstringLengthSixteenSymbolsDifference}");

Console.WriteLine("-------------------------------------------------------");

}

static void Test2()

{

const int STRING\_LENGTH = 256;

SHA\_256 sha\_256 = new();

Dictionary<int, int> generationsHashCollisions = [];

for (int i = 2; i <= 6; i++)

{

int N = (int)Math.Pow(10, i);

HashSet<string> hash = [];

for (int j = 0; j < N; j++)

{

string generatedString = GenerateString(STRING\_LENGTH);

string generatedHash = sha\_256.GetHash(generatedString);

hash.Add(generatedHash);

}

int collisions = N - hash.Count;

generationsHashCollisions.Add(N, collisions);

}

Console.WriteLine($"{"Количество генераций",25} | {"Количество совпадений",25}");

Console.WriteLine(new string('-', 55));

foreach (var kvp in generationsHashCollisions)

{

Console.WriteLine($"{kvp.Key,25} | {kvp.Value,25}");

}

}

static void Test3()

{

SHA\_256 sha\_256 = new();

Stopwatch stopwatch = new();

const int N = 1000;

int[] lengths = [64, 128, 256, 512, 1024, 2048, 4096, 8192];

double[] avgStopwatch = new double[lengths.Length];

for(int j = 0; j < lengths.Length; j++)

{

int length = lengths[j];

for (int i = 0; i < N; i++)

{

string generatedString = GenerateString(length);

stopwatch.Start();

sha\_256.GetHash(generatedString);

stopwatch.Stop();

avgStopwatch[j] += stopwatch.Elapsed.TotalMilliseconds;

}

avgStopwatch[j] /= N;

}

Console.WriteLine($"{"Длина строки",25} | {"Время (мс)",25}");

Console.WriteLine(new string('-', 55));

for (int j = 0; j < lengths.Length; j++)

{

Console.WriteLine($"{lengths[j],25} | {avgStopwatch[j],25}");

}

}

static string GenerateString(int length)

{

Random random = new();

string str = "";

for (int j = 0; j < length; j++)

{

int isUpperCase = random.Next(0, 2);

if (isUpperCase == 0)

{

str += (char)random.Next(97, 123); // a-z

}

else

{

str += (char)random.Next(65, 91); // A-Z

}

}

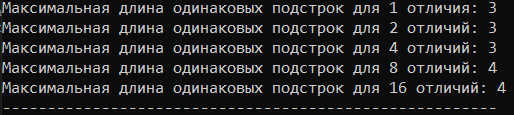
return str;

}

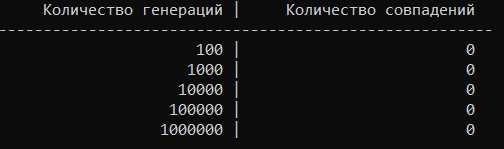
}

}

**Тест 1**



**Тест 2**



**Тест 3**



# Заключение.

1. Вне зависимости от количества измененных символов в оригинальной строке, хэш-значение будет сильно отличаться в обоих случаях.
2. Количество совпадений хэш-значений равно нулю при разных строках и разных количествах генераций.
3. Чем длиннее строка, тем больше нужно времени функции на расчет хэш-значения.