Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Дисциплина «Защита информации и надежность информационных систем»

**Отчёт по лабораторной работе №3**

Элементы теории информации. Информативность данных в различных кодировках

Студент: Жук С.С.

ФИТ 3 курс 2 группа

Преподаватель: Савельева М.Г.

Минск 2025

**Содержание**

[1 Теоретические сведения 3](#_Toc193186807)

[2 Конвертер в формат base64 5](#_Toc193186808)

[3 Энтропия и избыточность алфавита 6](#_Toc193186809)

[4 Операция XOR 8](#_Toc193186810)

[Вывод 11](#_Toc193186811)

# **1 Теоретические сведения**

Энтропия максимальна при равномерном появлении букв на любом месте сообщения. Для характеристики источника сообщений с различным алфавитом представляет интерес сравнение фактической энтропии источника с максимально возможной.

Избыточностью алфавитаназывают уменьшение информационной нагрузки на один символ вследствие разной вероятности и взаимозависимости появления его символов в сообщениях.

Одно и то же сообщение на основе алфавита русского и представленное в кодах ASCII будут характеризоваться различным количеством содержащейся в них информации. Дополнительная избыточность обусловлена переносом сообщения из одной среды в другую или, иначе говоря, кодированием символов исходного алфавита.

Ранее почтовый сервер при передаче письма обнулял старшие биты в каждом байте сообщения. Одним из решений проблемы стала кодировка (а точнее – алгоритм) base64. Алгоритм base64 используется для кодирования бинарных данных.

Кодирование base64 разработано для представления произвольных последовательностей октетов в форме, позволяющей использовать строчные и прописные буквы. Используется 65-символьное подмножество набора символов US-ASCII, обеспечивающее представление одним печатным символом 6 битов данных (дополнительный 65-й символ используется для обозначения функции специальной обработки).

Процесс кодирования:

1. Группа из 24 последовательных битов в форме строки из 4 символов.
2. Обработка выполняется слева направо.
3. 24-битная исходная группа образуется конкатенацией трех 8-битных групп.
4. 24 бита после трактуются как 4 сцепленных группы по 6 битов, каждая группа – символ алфавита base64.
5. Каждая 6-битная группа используется в качестве индекса массива из 64 печатных символов.
6. Символы алфавита, соответствующие индексу, помещаются в выходную строку.

Если кодируются только один или два байта, в результате получаются только первые два или три символа строки, а выходная строка дополняется двумя или одним символами «=». Это предотвращает добавление дополнительных битов к восстановленным данным.

Если на входе доступно менее 24 битов, входная группа дополняется (справа) нулями до формирования целого числа 6-битных групп. Заполнение в конце данных осуществляется как раз с использованием символа «=». Поскольку входная информация base64 всегда включает целое число октетов, возможны лишь перечисленные ниже случаи:

* размер финального блока кодирования на входе кратен 24 битам, кодированный результат будет содержать целое число 4-символьных групп без заполнения символами «=»;
* размер финального блока кодирования на входе составляет 8 битов, выходной блок будет представлять 2 символа, дополненные последовательностью из двух символов заполнения «==»;
* размер финального блока кодирования на входе составляет 16 битов, выходной блок будет представлять 3 символа, дополненные символом заполнения «=».



Операция XOR (вспомним, что эта операция называется также сложением по модулю 2, логическим сложением, исключающим «ИЛИ», строгой дизъюнкций, поразрядным дополнением).

Таблица 3.3 - Таблица истинности операции XOR

|  |  |  |
| --- | --- | --- |
| *а* | *b* | *а*XOR*b* |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

Если а и b имеют длину более 1 бита, к примеру 1 байт, то рассматриваемая операция над ними выполняется побитово. Указанным байтам могут соответствовать символы в определенной кодировке. Положим, символу «*М*» (hex4d) соответствует 8-битный код 01001101 (см. табл. 3.2), а символу «*а*» (hex61) соответствует код 01100001, тогда операция сложения по модулю 2 этих двух бинарных кодов дает 00101100 ((hex2с), или символ «,».

# **2 Конвертер в формат base64**

Для исследования алфавита на латинице был выбран текст на итальянском языке. Электронный текстовый документ на основе алфавита содержится в файле italian.txt, представленном на рисунке 2.1.

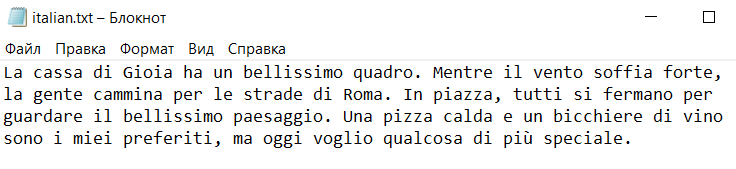


Рисунок 2.1 – Исходный текст итальянского алфавита

Для того, чтобы конвертировать текстовый документ italian.txt, необходимо было разработать функцию, которая представлена в листинге 2.1.

|  |
| --- |
| const encodeDocumentToBase64 = (filename) => {  const data = fs.readFileSync(filename, 'utf-8').toLowerCase();  return *Buffer*.from(data).toString('base64'); }; |

Листинг 2.1 – Функция для конвертации документа в формат base64

В результате получим некоторое количество строк, представленных в формате base64. Результат показан на рисунке 2.2.

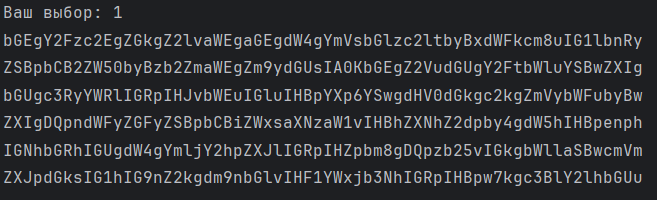


Рисунок 2.2 – Вывод содержимого документа в формате base64

# **3 Энтропия и избыточность алфавита**

Информационной характеристикой алфавита является энтропия. С физической точки зрения энтропия алфавита показывает, какое количество информации приходится в среднем на один символ алфавита.

Для вычисления энтропии Шеннона в начале необходимо подсчитать количество появлений каждого символа в текстовом файле для алфавита. В ней мы инициализируем количество вхождений символов и после из текста файла перебираем каждый символ. Далее для всех символов, которые встретились в файле, вычисляем вероятность их появления и энтропию Шеннона для алфавитов. Для этого используется функция, представленная в листинге 3.1.

|  |
| --- |
| const calculateShannonEntropy = (text) => {  const freq = {};  for (let char of text) {  if (!freq[char]) freq[char] = 0;  freq[char]++;  }  const totalChars = text.length;  let entropy = 0;  for (let char in freq) {  const p = freq[char] / totalChars;  entropy -= p \* *Math*.log2(p);  }  return entropy.toFixed(3); }; |

Листинг 3.1 – Функция расчета энтропии Шеннона

Для вычисления энтропии Хартли нужно знать, какова мощность алфавитов. Функция для ее вычисления представлена в листинге 3.2.

|  |
| --- |
| const calculateHartleyEntropy = (alphabet) => {  const result = *Math*.log2(alphabet.length);  return result.toFixed(3); }; |

Листинг 3.2 – Функция расчета энтропии Хартли

Избыточность алфавита рассчитывается по следующей формуле:

То есть для того, чтобы вычислить избыточность алфавита, необходимо найти разницу между энтропией Хартли и Шеннона, а после разделить полученную энтропию Хартли и умножить полученный результат 100. Расчет избыточности алфавита с помощью программы продемонстрирован в листинге 3.3.

|  |
| --- |
| const calculateRedundancy = (shannonEntropy, hartleyEntropy) => {  const result = ((hartleyEntropy - shannonEntropy) / hartleyEntropy) \* 100;  return result.toFixed(3); }; |

Листинг 3.3 – Функция расчета избыточности алфавита

В результате получим энтропию Шеннона и Хартли, а также избыточность для документов italian.txt и italianBase64.txt. Результат показан на рисунке 3.4.

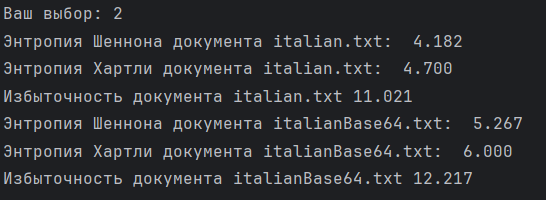


Рисунок 3.4 – Вывод результатов

Исходя из этих результатов, можно сделать вывод, что текст на итальянском языке (документ «a») имеет более низкую энтропию и избыточность по сравнению с текстом, закодированным в формате base64 (документ «b»). Текст на итальянском языке обычно содержит меньше разнообразия символов и требует больше информации для его представления. А текст, закодированный в формате base64, имеет более высокую энтропию и избыточность, что связано с использованием широкого набора символов в кодировке base64, потому что каждый символ base64 содержит более высокую информационную плотность по сравнению с символами в обычном тексте.

# **4 Операция XOR**

Операция XOR – побитовая логическая операция, которая сравнивает два бита.

Для начала опишем функцию, которая преобразует строку в бинарное представление. Здесь разбиваем строку на массив символов, после возвращаем для каждого символа его код в таблице Unicode, преобразуем числовой код символа в двоичную строку и, чтобы убедиться, что двоичная строка состоит из 8 битов, дополняем строку слева нулями, если она короче 8 бит. В конце все бинарные строки объединяем в одну длинную строку. Реализация функции продемонстрирована в листинге 4.1.

|  |
| --- |
| const toBinary = (str) => {  return str.split('').map(char => {  return char.charCodeAt(0).toString(2).padStart(8, '0');  }).join(''); }; |

Листинг 4.1 – Функция преобразования строки в бинарное представление

Далее разработали функцию, которая выполняет операцию XOR для двух бинарных строк. Сначала вычисляем длину самой длинной строки, чтобы оба бинарных представления были одинаковой длины. Далее дополняем более короткую строку нулями слева, чтобы обе строки стали одинаковой длины. После проходимся по каждому биту обеих строк и выполняем операцию XOR для всех битов. Программная реализация функции показана в листинге 4.2.

|  |
| --- |
| const xorBinaryStrings = (binaryA, binaryB) => {  const length = *Math*.max(binaryA.length, binaryB.length);  binaryA = binaryA.padStart(length, '0');  binaryB = binaryB.padStart(length, '0');   let result = '';  for (let i = 0; i < length; i++) {  result += (binaryA[i] === binaryB[i] ? '0' : '1');  }   return result; }; |

Листинг 4.2 – Функция XOR для двоичных строк

Следующим шагов была разработка функции, которая преобразует строку из бинарных данных обратно в обычную строку. Сначала создаем пустую строку для хранения результата. После в цикле проходим по всем битам переданной бинарной строки, для каждого шага цикла выделяем бит из бинарной строки и преобразуем 8-битный бинарный блок в десятичное число. Это число интерпретируем как код символа в таблице Unicode, и этот код преобразуется обратно в символ. Добавляем полученный символ в пустую строку для хранения результата и возвращаем ее. Реализация функции продемонстрирована в листинге 4.3.

|  |
| --- |
| const binaryToString = (binary) => {  let str = '';  for (let i = 0; i < binary.length; i += 8) {  const byte = binary.slice(i, i + 8);  str += String.fromCharCode(parseInt(byte, 2));  }  return str; }; |

Листинг 4.3 – Функция преобразования бинарного представления в строку

Далее опишем функцию, которая является реализацией двух первых функций. Здесь преобразуем строки a и b в их бинарное представление, выполняем операцию XOR между бинарными строками a и b, а также между полученным результатом и b. В конце возвращаем объект, который содержит результат первой и второй операции XOR и исходные бинарные строки. Программная реализация функции показана в листинге 4.4.

|  |
| --- |
| const xorBinaryOperation = (a, b) => {  const binaryA = toBinary(a);  const binaryB = toBinary(b);   const xorAB = xorBinaryStrings(binaryA, binaryB);  const xorABB = xorBinaryStrings(xorAB, binaryB);   return { xorAB, xorABB, binaryA, binaryB }; }; |

Листинг 4.4 – Функция XOR для двоичных данных

Следующим шагом была разработана функция, которая преобразует строку в формат base64. Создаем буфер из строки, представленной в кодировке UTF-8, и преобразуем этот буфер в строку, закодированную в формате base64. Реализация функции продемонстрирована в листинге 4.5.

|  |
| --- |
| const toBase64 = (str) => {  return *Buffer*.from(str, 'utf-8').toString('base64'); }; |

Листинг 4.5 – Функция для преобразования строки в base64

В конце разработали функцию, которая выполняет операцию XOR над двумя строками, представленных в формате base64. Сначала преобразуем строки в их представления в формате base64 и после преобразуем полученные строки base64 в бинарные строки. Далее выполняем операцию XOR на двух бинарных строках. Результат этой операции сохраняется в переменных *xorAB* и *xorABB*. В последней переменной будет отмененная первая операция XOR, так как применение XOR дважды с одним и тем же значением восстанавливает исходное значение. После преобразуем бинарные строки обратно в текстовое представление. Функция возвращает объект, который содержит результаты первой и второй операции XOR и исходные строки. Реализация функции показана в листинге 4.6.

|  |
| --- |
| const xorBase64Operation = (a, b) => {  const base64A = toBase64(a);  const base64B = toBase64(b);   const binaryA = toBinary(base64A);  const binaryB = toBinary(base64B);   const xorAB = xorBinaryStrings(binaryA, binaryB);  const xorABB = xorBinaryStrings(xorAB, binaryB);   const xorBase64 = binaryToString(xorAB);  const xorBase64\_XORb = binaryToString(xorABB);   return { xorBase64, xorBase64\_XORb, originalA: a, originalB: b }; }; |

Листинг 4.6 – Функция XOR для строк в формате base64

В результате получим выполнение XOR в ASCII. Результат показан на рисунке 4.1.

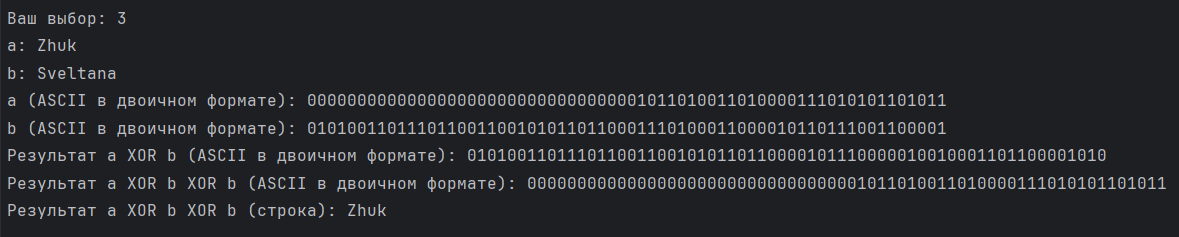


Рисунок 4.1 – Вывод результатов для ASCII операции XOR

Рассчитаем то же самое, только для формата base64. Вывод продемонстрирован на рисунке 4.2.

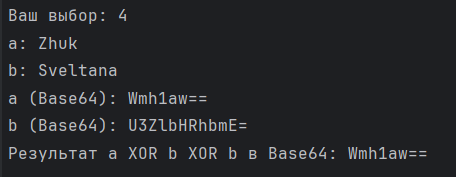


Рисунок 4.2 – Вывод результатов для base64 операции XOR

# **Вывод**

В ходе выполнения лабораторной работы были изучены основные принципы теории информации и получены практические навыки в области преобразования данных и анализа их энтропийных характеристик.

Кроме того, в процессе работы было разработано специализированное приложение для расчёта энтропии по формулам Шеннона и Хартли, а также для определения избыточности алфавитов ASCII и base64. В приложении присутствует возможность выполнить операцию XOR.

Низкая энтропия и высокая избыточность указывает на большое количество повторяющихся символов в документе. Это свидетельствует о том, что информация в нем менее разнообразна. Высокая энтропия и избыточность означают наличие множества уникальных символов в документе. Такие данные обычно содержат более разнообразную информацию и могут быть более ценными для анализа.

Также сделан вывод о том, что анализ энтропии и избыточности позволяет оценить информационную насыщенность документа, что может быть полезным для решения задач сжатия данных или их дальнейшей обработки.