Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ

УНИВЕРСИТЕТ»

**Отчёт**

Лабораторная работа №3 «Элементы теории информации.

Информативность данных в различных кодировках»

Студент: Водчиц А. В.

ФИТ 3 курс 1 группа

Преподаватель: Нистюк О. А.

Минск 2025

**Цель:** приобретение практических навыков трансформации данных и сопоставление энтропийных свойств используемых при этом алфавитов.

**Задачи:**

1. Закрепить теоретические знания по взаимной конвертации данных, представленных в кодах ASCII и base64.

2. Разработать приложение для конвертации произвольного документа в формат base64 и обратно.

3. Исследовать энтропийные характеристики используемых в конвертерах алфавитов.

4. Изучить особенности практической реализации операции XOR над данными, представленными в разных форматах.

5. Результаты выполнения лабораторной работы оформить в виде описания разработанного приложения, методики выполнения экспериментов с использованием приложения и результатов эксперимента

1. Создать собственное приложение (приветствуется!) или воспользоваться Base64-онлайн-кодировщиком, с помощью которого конвертировать произвольный документ (а) на латинице (можно использовать документ из лабораторной работы № 1) в документ (б) формата base64.

# Теоретические сведения

Из энтропийных оценок (алфавитов и сообщений), полученных в ходе выполнения лабораторной работы № 2, мы выяснили, что энтропия зависит от статических характеристик самих алфавитов и сообщений (вспомним энтропию по Шеннону и по Хартли). 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. В этом смысле введено понятие избыточности источника сообщений, или избыточности алфавита.

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

При выполнении предыдущей работы мы убедились, что формально одно и то же сообщение, но представленное на основе алфавита русского (белорусского, английского или иного) языка – с одной стороны, и представленное в кодах ASCII – с другой, будут характеризоваться различным количеством содержащейся в них информации. Эта дополнительная избыточность обусловлена переносом сообщения из одной среды в другую или, иначе говоря, кодированием символов исходного алфавита. Утверждение восьмибитных кодировок (ASCII) как стандарта принесло некоторые проблемы. К этому моменту уже существовала определенная инфраструктура, использующая семибитные кодировки. Известны проблемы с «обрезанием восьмого бита» в системе электронной почты. Утверждение восьмибитного символа дало 256 различных значений, что позволило уместить в одной кодовой таблице и общепринятые символы (цифры, знаки препинания, латиницу), и символы кириллицы.

Уже созданное к тому времени и работающее программное обеспечение зачастую было приспособлено для семибитных кодировок, что приводило, например, к тому, что почтовый сервер при передаче письма обнулял старшие биты в каждом байте сообщения. Одним из решений проблемы стала кодировка (а точнее – алгоритм) base64. В PGP 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 представляет группу из 24 последовательных битов в форме строки из 4 символов. Обработка выполняется слева направо, а 24-битная исходная группа образуется конкатенацией трех 8-битных групп (байтов). Данные 24 бита после этого трактуются как 4 сцепленных группы по 6 битов, каждая из которых транслируется в один символ алфавита base64. Каждая 6-битная группа используется в качестве индекса массива из 64 печатных символов. Символы алфавита, соответствующие индексу, помещаются в выходную строку [6]. Кодирование base64 с безопасным алфавитом используется для представления URL и имен файлов.

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 «=».

# Практические задания

**1. Создать собственное приложение (приветствуется!) или воспользоваться Base64-онлайн-кодировщиком, с помощью которого конвертировать произвольный документ (а) на латинице (можно использовать документ из лабораторной работы № 1) в документ (б) формата base64.**

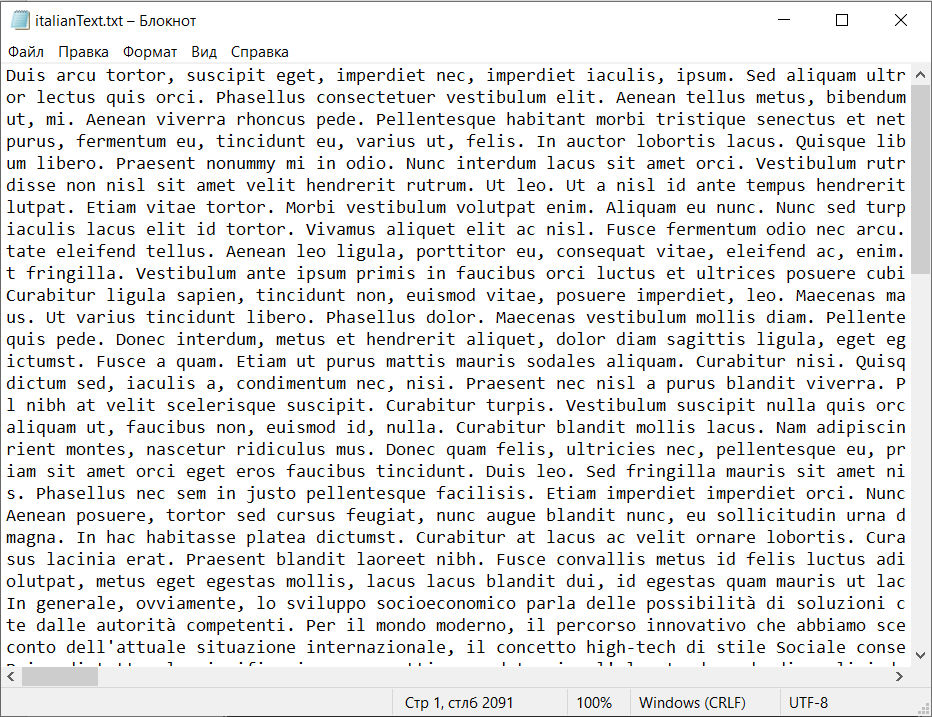


Рисунок 1.1 – Исходный документ

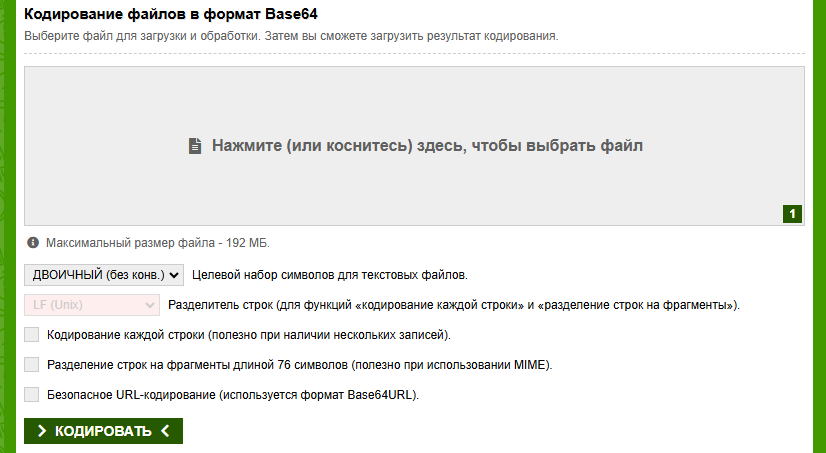


Рисунок 1.2 – Веб-страница кодировщика



Рисунок 1.3 – Результат кодирования

**2. С помощью приложения, созданного в лабораторной работе № 1, получить распределение частотных свойств алфавитов по документам (а) и (б). Вычислить энтропию Хартли и Шеннона, а также избыточность алфавитов. Объяснить полученный результат.**

|  |
| --- |
| public static double CalculateEntropy(string text, char[] alphabet, string filePath)  {  text = new string(text.ToLower().Where(c => alphabet.Contains(c)).ToArray());  int textLength = text.Length;  if (textLength < 100)  {  Console.WriteLine("Текст слишком маленький для того, чтобы рассчитать энтропию");  return 0;  }  var frequency = new Dictionary<char, int>();  foreach (var letter in alphabet) frequency[letter] = 0;  foreach (var c in text) frequency[c]++;  foreach (var kvp in frequency)  {  double probability = (double)kvp.Value / textLength;  }  SaveToExcel(frequency, textLength, filePath);  double entropy = 0;  foreach (var kvp in frequency)  {  double probability = (double)kvp.Value / textLength;  if (probability > 0) entropy += probability \* Math.Log2(probability);  }  return -entropy;  } |

Листинг 2.1 – Метод для подсчёта энтропии Шеннона

|  |
| --- |
| public static double CalculateEntropyHartly(char[] alphabet)  {  return Math.Log2(alphabet.Length);  } |

Листинг 2.2 – Метод для подсчёта энтропии Хартли

|  |
| --- |
| public static double AlphabetRedundancy(double EntropyShannon, double EntropyHartly)  {  return 1 - EntropyShannon / EntropyHartly;  } |

Листинг 2.3 – Метод для подсчёта избыточности алфавита

|  |
| --- |
| public static void Main()  {  string italianText = FileReader.ReadTextFromFile("italianText.txt");  string base64Text = FileReader.ReadTextFromFile("encoded-20250302175431.txt");  char[] ItalianAlphabet = "abcdefghijklmnopqrstuvwxyzàèéìíîòóùú".ToCharArray();  char[] Base64Alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/=".ToCharArray();  Console.WriteLine("==================================================");  double italianEntropy = EntropyCalculator.CalculateEntropy(italianText, ItalianAlphabet, "frequency\_data\_italian.xlsx");  double italianEntropyHartly = EntropyCalculator.CalculateEntropyHartly(ItalianAlphabet);  Console.WriteLine($"Энтропия для итальянского текста по Шеннону: {italianEntropy:F4}");  Console.WriteLine($"Энтропия для итальянского текста по Хартли: {italianEntropyHartly:F4}");  Console.WriteLine($"Избыточность алфавита: {EntropyCalculator.AlphabetRedundancy(italianEntropy, italianEntropyHartly):F4}");  Console.WriteLine("==================================================\n");  Console.WriteLine("\n==================================================");  double base64Entropy = EntropyCalculator.CalculateEntropy(base64Text, Base64Alphabet, "frequency\_data\_base64.xlsx");  double base64EntropyHartly = EntropyCalculator.CalculateEntropyHartly(Base64Alphabet);  Console.WriteLine($"Энтропия для документа формата base64 по Шеннону: {base64Entropy:F4}");  Console.WriteLine($"Энтропия для документа формата base64 по Хартли: {base64EntropyHartly:F4}");  Console.WriteLine($"Избыточность алфавита: {EntropyCalculator.AlphabetRedundancy(base64Entropy, base64EntropyHartly):F4}");  Console.WriteLine("==================================================\n");  } |

Листинг 2.4 – Метод main

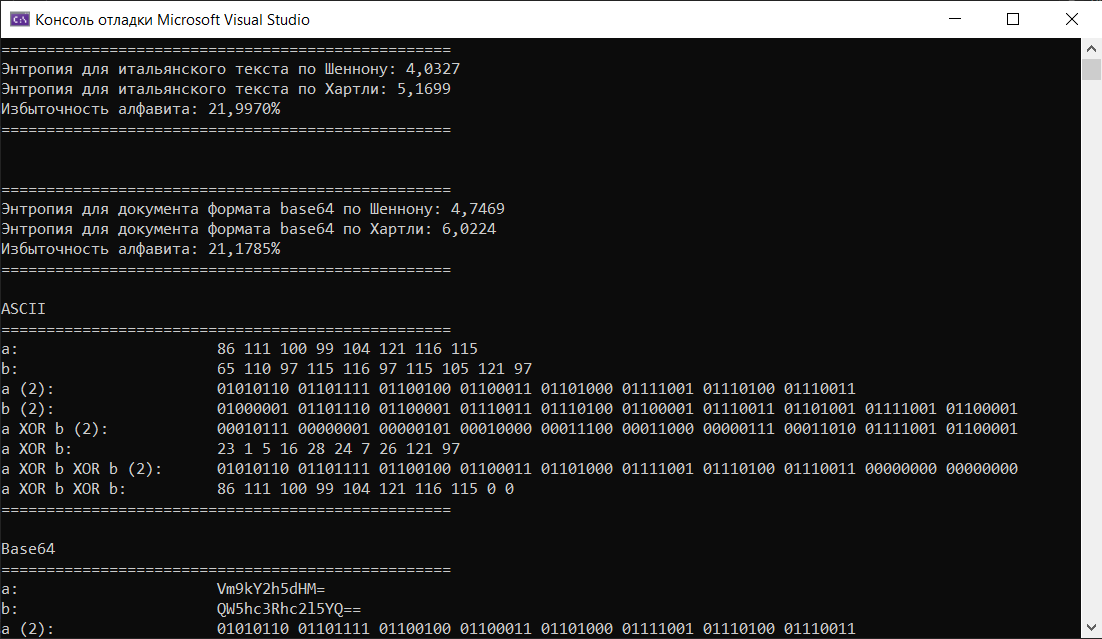


Рисунок 2.1 – Результат расчёта энтропии и избыточности

**3. Написать функцию, которая принимает в качестве аргументов два буфера (а и b) одинакового размера и возвращает XOR (собственная фамилия (а) и имя (b); при разной длине меньшую дополнить нулями). Входные аргументы представлять: 1) в кодах ASCII; 2) в кодах base64. Что будет результатом операции аXORbXORb? При написании не использовать стандартные функции языка программирования. Итоговые данные сравнить с результатами использования стандартных функций языка программирования (если они есть).**

|  |
| --- |
| class XOR  {  public static byte[] XORBuffers(byte[] a, byte[] b)  {  int maxLength = Math.Max(a.Length, b.Length);  byte[] result = new byte[maxLength];  for (int i = 0; i < maxLength; i++)  {  byte byteA = (i < a.Length) ? a[i] : (byte)0;  byte byteB = (i < b.Length) ? b[i] : (byte)0;  result[i] = (byte)(byteA ^ byteB);  }  return result;  }  public static string ToBinary(byte[] data) => string.Join(" ", data.Select(b => Convert.ToString(b, 2).PadLeft(8, '0')));  } |

Листинг 3.1 – Метод для вычисления XOR

|  |
| --- |
| string sname = "Vodchyts";  string fname = "Anastasiya";  byte[] a\_ascii = Encoding.ASCII.GetBytes(sname);  byte[] b\_ascii = Encoding.ASCII.GetBytes(fname);  string a\_Base64 = Convert.ToBase64String(a\_ascii);  string b\_Base64 = Convert.ToBase64String(b\_ascii);  byte[] a\_Base64\_byte = Convert.FromBase64String(a\_Base64);  byte[] b\_Base64\_byte = Convert.FromBase64String(b\_Base64);  byte[] result\_ascii = XOR.XORBuffers(a\_ascii, b\_ascii);  byte[] result\_Base64 = XOR.XORBuffers(a\_Base64\_byte, b\_Base64\_byte);  byte[] result\_ascii\_double = XOR.XORBuffers(result\_ascii, b\_ascii);  byte[] result\_Base64\_double = XOR.XORBuffers(result\_Base64, b\_Base64\_byte);  Console.WriteLine("ASCII");  Console.WriteLine("==================================================");  Console.WriteLine("a: \t\t\t" + string.Join(" ", a\_ascii));  Console.WriteLine("b: \t\t\t" + string.Join(" ", b\_ascii));  Console.WriteLine("a (2): \t\t\t" + XOR.ToBinary(a\_ascii));  Console.WriteLine("b (2): \t\t\t" + XOR.ToBinary(b\_ascii));  Console.WriteLine("a XOR b (2): \t\t" + XOR.ToBinary(result\_ascii));  Console.WriteLine("a XOR b: \t\t" + string.Join(" ", result\_ascii));  Console.WriteLine("a XOR b XOR b (2): \t" + XOR.ToBinary(result\_ascii\_double));  Console.WriteLine("a XOR b XOR b: \t\t" + string.Join(" ", result\_ascii\_double));  Console.WriteLine("==================================================\n");  Console.WriteLine("Base64");  Console.WriteLine("==================================================");  Console.WriteLine("a: \t\t\t" + a\_Base64);  Console.WriteLine("b: \t\t\t" + b\_Base64);  Console.WriteLine("a (2): \t\t\t" + XOR.ToBinary(a\_Base64\_byte));  Console.WriteLine("b (2): \t\t\t" + XOR.ToBinary(b\_Base64\_byte));  Console.WriteLine("a XOR b (2): \t\t" + XOR.ToBinary(result\_Base64));  Console.WriteLine("a XOR b: \t\t" + Convert.ToBase64String(result\_Base64));  Console.WriteLine("a XOR b XOR b (2): \t" + XOR.ToBinary(result\_Base64\_double));  Console.WriteLine("a XOR b XOR b: \t\t" + Convert.ToBase64String(result\_Base64\_double));  Console.WriteLine("==================================================\n"); |

Листинг 3.2 – Метод main

|  |  |  |
| --- | --- | --- |
| X | Y | X XOR Y |
| 1 | 1 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 0 | 0 | 0 |

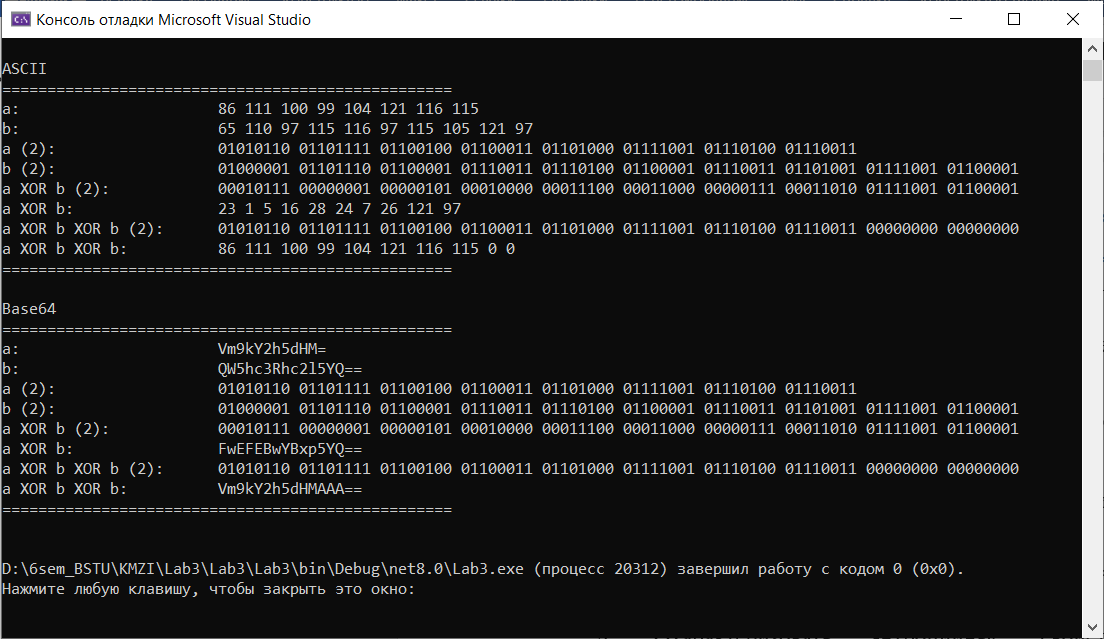


Рисунок 3.1 – Результат вычисления XOR

x ⊕ x = 0, так как XOR даёт 1, если биты разные, и 0, если одинаковые.  
Когда число выполняет операцию само с собой, все биты одинаковые, значит, получается 0.

x ⊕ 0 = x, так как XOR с нулём не меняет число, потому что 0 не вносит изменений.

Рассмотрем выражение a ⊕ b ⊕ b

* b XOR b даёт 0, так как любое число XOR само с собой обнуляется
* Осталось: a ⊕ 0 = a

Итог: a ⊕ b ⊕ b = a

Операция XOR с числом дважды возвращает исходное число.

**4. Результаты оформить в виде отчета по установленным правилам.**

Вывод: в ходе выполнения данной лабораторной работы были закреплены теоретические знания и получены практические навыки по преобразованию данных между текстовым форматом и кодировкой Base64. Было произведено кодирование исходного документа на латинице в формат Base64. С использованием программных средств, разработанных в предыдущей работе, был выполнен анализ частотных свойств и расчет энтропийных характеристик (энтропия по Шеннону, энтропия по Хартли, избыточность) для исходного алфавита и для алфавита Base64. Кроме того, была реализована функция для выполнения побитовой операции XOR над данными, представленными в виде байтовых массивов (коды ASCII и байты из Base64).