Учреждение образования

«Белорусский государственный технологический университет»

**Защита информации и надежность информационных систем**

**Лабораторная работа №3**

ЭЛЕМЕНТЫ ТЕОРИИ ИНФОРМАЦИИ.

ИНФОРМАТИВНОСТЬ ДАННЫХ

В РАЗЛИЧНЫХ КОДИРОВКАХ

Выполнил:

Студент 3 курса 2 группы ФИТ

Максимова Вера Владимировна

Проверил:

Ржеутская Надежда Викентьевна

**2022 г.**

1. **Теоретические сведения**

Из энтропийных оценок (алфавитов и сообщений), получен-

ных в ходе выполнения лабораторной работы № 2, мы выяснили,

что энтропия зависит от статических характеристик самих алфави-

тов и сообщений (вспомним энтропию по Шеннону и по Хартли).

Энтропия максимальна при равномерном появлении букв на

любом месте сообщения. Для характеристики источника сообще-

ний с различным алфавитом представляет интерес сравнение фак-

тической энтропии источника с максимально возможной. В этом

смысле введено понятие избыточности источника сообщений, или

избыточности алфавита.

Избыточностью алфавита называют уменьшение информаци-

онной нагрузки на один символ вследствие разной вероятности и

взаимозависимости появления его символов в сообщениях.

В наиболее общем виде избыточность алфавита R можно оце-

нить отношением энтропии по Хартли и по Шеннону; при этом пер-

вая рассчитывается по выражению (2.2), вторая – по формуле (2.1):



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, бу-

дут 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 существова-

ла определенная инфраструктура, использующая семибитные ко-

дировки. 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 препи-

нания, латиницу), и символы кириллицы.

Уже созданное к тому времени и работающее программное

обеспечение зачастую было приспособлено для семибитных коди-

ровок, что приводило, например, к тому, что почтовый сервер при

передаче письма обнулял старшие биты в каждом байте сообще-

ния. Одним из решений проблемы стала кодировка (а точнее – ал-

горитм) base64. В PGP алгоритм base64 используется для кодиро-

вания бинарных данных.

Кодирование base64 разработано для представления произ-

вольных последовательностей октетов в форме, позволяющей ис-

пользовать строчные и прописные буквы. Используется 65-сим-

вольное подмножество набора символов US-ASCII, обеспечиваю-

щее представление одним печатным символом 6 битов данных

(дополнительный 65-й символ используется для обозначения

функции специальной обработки).

Процесс кодирования представляет группу из 24 последова-

тельных битов в форме строки из 4 символов. Обработка выполня-

ется слева направо, а 24-битная исходная группа образуется кон-

катенацией 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 мас-

сива из 64 печатных символов. Символы алфавита, соответствую-

щие индексу, помещаются в выходную строку [6]. Кодирование

base64 с безопасным алфавитом используется для представления

URL и имен файлов.

В табл. 3.1 перечислен алфавит, используемый для base64-

кодировки. Значения представлены в различных системах счисле-

ния: десятичной (10), двоичной (2), восьмеричной (8) и шестна-

дцатеричной (16 или hex).

Еще раз обратимся к процессу кодировки. Как было выше ус-

тановлено, каждые 6 битов буфера, начиная с самых старших, ис-

пользуются как индексы строки «ABCDEFGHIJKLMNOPQRSTUV

WXYZabcdefghijklmnopqrstuvwxyz0123456789+/», и ее символы,

на которые указывают индексы, 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.

Процесс повторяется над оставшимися входными данными.

Такая обработка выполняется в тех случаях, когда последняя

группа входных данных содержит меньше 24 битов. Кодируемое

значение всегда завершается полным квантом кодирования.

Если на входе доступно менее 24 битов, входная группа до-

полняется (справа) нулями до формирования целого числа 6-бит-

ных групп. Заполнение в конце данных осуществляется как раз с

использованием символа «=». Поскольку входная информация

base64 всегда включает целое число октетов, возможны лишь пе-

речисленные ниже случаи:

• размер финального блока кодирования на входе кратен 24 би-

там, кодированный результат будет содержать целое число 4-сим-

вольных групп без заполнения символами «=»;

• размер финального блока кодирования на входе составляет

8 битов, выходной блок будет представлять 2 символа, дополнен-

ные последовательностью из двух символов заполнения «==»;

• размер финального блока кодирования на входе составляет

16 битов, выходной блок будет представлять 3 символа, допол-

ненные символом заполнения «=»

1. **Приложение**

**using System;**

**using System.Collections.Generic;**

**using System.IO;**

**using System.Linq;**

**using System.Text;**

**using System.Threading.Tasks;**

**namespace Lab3**

**{**

**class Program**

**{**

**static void Main(string[] args)**

**{**

**//LAB 2**

**//task 1**

**ToBase64 toBase64 = new ToBase64();**

**toBase64.MethodToBase64();**

**//task 2**

**Superfluity izbitochoct = new Superfluity();**

**izbitochoct.CalculationSuperfluity();**

**//task 3**

**XORNameFamily xORNameFamily = new XORNameFamily();**

**xORNameFamily.CalculateXOR("Vera", "Maksimova");**

**Console.WriteLine();**

**}**

**class ToBase64**

**{**

**public void MethodToBase64()**

**{**

**string base64 = "";**

**using (StreamReader streamReader = new StreamReader(@"D:\3 курс 1 сем\ЗАЩИТА ИНФОРМАЦИИ И НАДЕЖНОСТЬ ИНФОРМАЦИОННЫХ СИСТЕМ\3 Лабораторная работа\Lab3\english.txt"))**

**{**

**string file = streamReader.ReadToEnd();**

**Console.WriteLine("Исходный текст:");**

**Console.WriteLine(file);**

**Console.WriteLine();**

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

**{**

**byte[] ascii = Encoding.ASCII.GetBytes(file);**

**base64 = Convert.ToBase64String(ascii);**

**}**

**Console.WriteLine("Перевод в base64:");**

**Console.WriteLine(base64);**

**}**

**using (StreamWriter sw = new StreamWriter(@"D:\3 курс 1 сем\ЗАЩИТА ИНФОРМАЦИИ И НАДЕЖНОСТЬ ИНФОРМАЦИОННЫХ СИСТЕМ\3 Лабораторная работа\Lab3\base64.txt", false))**

**{**

**sw.WriteLine(base64);**

**}**

**}**

**}**

**class Superfluity**

**{**

**public void CalculationSuperfluity()**

**{**

**//english**

**EntropyEnglish entropyenglish = new EntropyEnglish();**

**entropyenglish.EntropyEnglishAlphabet();**

**double entropyEnglishAlphabetHartly = Math.Log(33, 2);**

**Console.WriteLine($"Энтропия по Хартли (EN): {entropyEnglishAlphabetHartly}");**

**double superfluityEnglish = 1 - (ShannonEntropyOfTheGAlphabet / entropyEnglishAlphabetHartly);**

**Console.WriteLine($"Избыточность информации (EN): {superfluityEnglish}");**

**//base**

**EntropyBase entropyBase = new EntropyBase();**

**entropyBase.EntropyBaseAlphabet();**

**double entropyBinaryAlphabetHartly = Math.Log(64, 2);**

**Console.WriteLine($"Энтропия по Хартли Base текста: {entropyBinaryAlphabetHartly}");**

**double superfluityBinary = 1 - (ShannonEntropyOfTheBAlphabet / entropyBinaryAlphabetHartly);**

**Console.WriteLine($"Избыточность информации Base текста: {superfluityBinary}");**

**}**

**}**

**class XORNameFamily**

**{**

**public void CalculateXOR(string a, string b)**

**{**

**byte[] asciiName = Encoding.ASCII.GetBytes(a);**

**byte[] asciFamily = Encoding.ASCII.GetBytes(b);**

**string base64Name = "";**

**string base64Family = "";**

**byte[] newasciiName = new byte[] { };**

**byte[] newasciiFamily = new byte[] { };**

**if (asciiName.Length < asciFamily.Length)**

**{**

**Console.WriteLine();**

**newasciiFamily = asciFamily;**

**newasciiName = new byte[asciFamily.Length];**

**for (int i = 0**

**; i < asciiName.Length; i++)**

**{**

**newasciiName[i] = asciiName[i];**

**Console.Write(newasciiName[i] + " ");**

**}**

**for (int j = asciiName.Length; j < asciFamily.Length; j++)**

**{**

**newasciiName[asciiName.Length + 1] = 0;**

**Console.Write(newasciiName[j] + " ");**

**}**

**Console.WriteLine();**

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

**{**

**Console.Write(newasciiFamily[i] + " ");**

**}**

**base64Name = Convert.ToBase64String(newasciiName);**

**base64Family = Convert.ToBase64String(asciFamily);**

**Console.WriteLine();**

**Console.WriteLine(base64Name);**

**Console.WriteLine(base64Family);**

**}**

**else**

**{**

**Console.WriteLine();**

**newasciiName = asciiName;**

**newasciiFamily = new byte[asciiName.Length];**

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

**{**

**newasciiFamily[i] = asciFamily[i];**

**Console.WriteLine(newasciiFamily[i]);**

**}**

**for (int j = asciFamily.Length; j < asciiName.Length; j++)**

**{**

**newasciiFamily[asciFamily.Length + 1] = 0;**

**Console.WriteLine(newasciiFamily[j]);**

**}**

**base64Name = Convert.ToBase64String(newasciiName);**

**base64Family = Convert.ToBase64String(asciFamily);**

**Console.WriteLine(base64Name);**

**Console.WriteLine(base64Family);**

**}**

**Console.WriteLine("\n\nРезультат логического сложения в ASCII:");**

**int[] result = new int[newasciiName.Length];**

**int[] aaa = new int[base64Name.Length];**

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

**{**

**result[i] = newasciiName[i] ^ newasciiFamily[i] ^ newasciiFamily[i];**

**//Console.Write(result[i] + " ");**

**char ch = (char)result[i];**

**Console.Write(ch);**

**}**

**string res = "";**

**Console.WriteLine();**

**for (int i = 0; i < base64Name.Length; i++)**

**{**

**aaa[i] = base64Name[i] ^ base64Family[i] ^ base64Family[i];**

**//Console.Write(result[i] + " ");**

**char ch = (char)aaa[i];**

**res += ch;**

**Console.Write(ch);**

**}**

**Console.WriteLine();**

**Console.WriteLine(Encoding.ASCII.GetString(Convert.FromBase64String(res)));**

**}**

**}**

**static double ShannonEntropyOfTheGAlphabet = 0;**

**static double ShannonEntropyOfTheBAlphabet = 0;**

**static double ShannonEntropyOfTheBinaryAlphabet = 0;**

**class EntropyEnglish**

**{**

**public void EntropyEnglishAlphabet()**

**{**

**int[] countLetter = new int[26];**

**int countLettersInFile = 0;**

**double[] probabilityLetters = new double[26];**

**// double ShannonEntropyOfTheAlphabet = 0;**

**string letters = "abcdefghijklmnopqrstuvwxyz";**

**using (StreamReader streamReader = new StreamReader(@"D:\3 курс 1 сем\ЗАЩИТА ИНФОРМАЦИИ И НАДЕЖНОСТЬ ИНФОРМАЦИОННЫХ СИСТЕМ\3 Лабораторная работа\Lab3\english.txt"))**

**{**

**string file = streamReader.ReadToEnd().ToLower();**

**countLettersInFile = file.Count();**

**Console.WriteLine($"Количество символов в файле: {countLettersInFile}");**

**Console.WriteLine();**

**Console.WriteLine("Количество вхождений каждой буквы:\n");**

**for (int j = 0; j < 26; j++)**

**{**

**countLetter[j] = file.Count(x => x == letters[j]);**

**if (countLetter[j] != 0)**

**{**

**Console.WriteLine($"{letters[j]}: {countLetter[j]}");**

**probabilityLetters[j] = (double)countLetter[j] / countLettersInFile;**

**Console.WriteLine($"P({letters[j]}): {probabilityLetters[j]}");**

**Console.WriteLine();**

**ShannonEntropyOfTheGAlphabet += probabilityLetters[j] \* (Math.Log(probabilityLetters[j]) / Math.Log(2)) \* (-1);**

**}**

**}**

**Console.WriteLine("Энтропия Английского алфавита по Шеннону:");**

**Console.WriteLine(ShannonEntropyOfTheGAlphabet);**

**}**

**}**

**}**

**class EntropyBase**

**{**

**public void EntropyBaseAlphabet()**

**{**

**int[] countLetter = new int[64];**

**int countLettersInFile = 0;**

**double[] probabilityLetters = new double[64];**

**// double ShannonEntropyOfTheAlphabet = 0;**

**string letters = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/";**

**using (StreamReader streamReader = new StreamReader(@"D:\3 курс 1 сем\ЗАЩИТА ИНФОРМАЦИИ И НАДЕЖНОСТЬ ИНФОРМАЦИОННЫХ СИСТЕМ\3 Лабораторная работа\Lab3\base64.txt"))**

**{**

**string file = streamReader.ReadToEnd().ToLower();**

**countLettersInFile = file.Count();**

**Console.WriteLine($"Количество символов в файле: {countLettersInFile}");**

**Console.WriteLine();**

**Console.WriteLine("Количество вхождений каждой буквы:\n");**

**for (int j = 0; j < 64; j++)**

**{**

**countLetter[j] = file.Count(x => x == letters[j]);**

**if (countLetter[j] != 0)**

**{**

**Console.WriteLine($"{letters[j]}: {countLetter[j]}");**

**probabilityLetters[j] = (double)countLetter[j] / countLettersInFile;**

**Console.WriteLine($"P({letters[j]}): {probabilityLetters[j]}");**

**Console.WriteLine();**

**ShannonEntropyOfTheBAlphabet += probabilityLetters[j] \* (Math.Log(probabilityLetters[j]) / Math.Log(2)) \* (-1);**

**}**

**}**

**Console.WriteLine("Энтропия алфавита Base по Шеннону:");**

**Console.WriteLine(ShannonEntropyOfTheBAlphabet);**

**}**

**}**

**}**

**class EntropyBinaryDigit**

**{**

**public void EntropyBinaryAlphabet()**

**{**

**int[] countLetter = new int[2];**

**int countLettersInFile = 0;**

**double[] probabilityLetters = new double[2];**

**//double ShannonEntropyOfTheBinaryAlphabet = 0;**

**string letters = "01";**

**using (StreamReader streamReader = new StreamReader(@"D:\3 курс 1 сем\ЗАЩИТА ИНФОРМАЦИИ И НАДЕЖНОСТЬ ИНФОРМАЦИОННЫХ СИСТЕМ\3 Лабораторная работа\Lab3\base64.txt"))**

**{**

**string file = streamReader.ReadToEnd();**

**countLettersInFile = file.Count();**

**Console.WriteLine();**

**Console.WriteLine();**

**Console.WriteLine("BINARY DIGIT");**

**Console.WriteLine($"Количество символов в файле: {countLettersInFile}");**

**Console.WriteLine();**

**Console.WriteLine("Количество вхождений каждой цифры:");**

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

**{**

**for (int j = 0; j < 2; j++)**

**{**

**countLetter[j] = file.Count(x => x == letters[j]);**

**Console.WriteLine($"{letters[j]}: {countLetter[j]}");**

**probabilityLetters[j] = (double)countLetter[j] / countLettersInFile;**

**Console.WriteLine($"P({letters[j]}): {probabilityLetters[j]}");**

**Console.WriteLine();**

**ShannonEntropyOfTheBinaryAlphabet += (probabilityLetters[j] \* (Math.Log(probabilityLetters[j])) / Math.Log(2)) \* (-1);**

**}**

**Console.WriteLine($"Энтропия бинарного алфавита по Шеннону: {ShannonEntropyOfTheBinaryAlphabet}");**

**break;**

**}**

**}**

**}**

**}**

**}**

**}**

1. **Ответы на контрольные вопросы**

1. Что такое base64?

**Base64** — стандарт кодирования двоичных данных при помощи только 64 символов ASCII

2. Как проверить, была ли определенная строка символов закодирована в base64?

* проверьте, что длина кратна 4 символам
* проверьте, что каждый символ находится в наборе A-Z, a-z, 0-9, +, / за исключением заполнения в конце, которое равно 0, 1 или 2 '=' символов

3. Как с помощью base64 проверить подлинность вводимых данных в форму пароля и логина?

4. Охарактеризуйте энтропийные свойства алфавитов в проанализированных форматах данных.

энтропия на base64 выходит больше и избыточная информация тоже в base64 больше, чем в обычном английском алфавите(проверено на практике )

5. Объяснить результат операции аXORbXORb. Где может найти применение такая операция?

6. Как будут выглядеть строки: efd8b295a633908a3c0828b2 faea8766 4d72cde3aaa0 после их конвертации в base64?

ZWZkOGIyOTVhNjMzOTA4YTNjMDgyOGIy

ZmFlYTg3NjY=

NGQ3MmNkZTNhYWEw

7. Результат операции аXORb (а – каждый байт строки, b – некоторая неизменная величина) будет строка:

1f180d1e1f04051c404c0f19 1f180308050d024c030a4c18 04094c1f18030009024c1c00

Найти значение b.