Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

**Отчёт**

Лабораторная работа №9 «Сжатие/распаковка данных на основе статических методов»

Студент: Водчиц Анастасия

ФИТ 3 курс 1 группа

Преподаватель: Нистюк О.А.

Минск 2025

**Цель**: приобретение практических навыков использования статистических методов Шеннона − Фано и Хаффмана (ShannonFano and Huffman coding) для сжатия/распаковки данных.

**Задачи**:

* Закрепить теоретические знания по алгебраическому описанию и использованию методов сжатия/распаковки (архивации/разархивации) данных на основе методов Шеннона − Фано и Хаффмана.
* Разработать приложение для реализации методов Шеннона − Фано и Хаффмана.
* Результаты выполнения лабораторной работы оформить в виде описания разработанного приложения, методики выполнения экспериментов с использованием приложения и результатов эксперимента.

# **Теоретические сведения**

Статистические алгоритмы позволяют создавать более короткие коды для часто встречающихся и более длинные – для редко встречающихся символов алфавита или конкретного сообщения. В первом случае метод считается статическим статистическим, во втором – динамическим статистическим: вероятностные свойства символов подсчитываются для конкретного сообщения или потока данных.

Частота или вероятность появления того или иного символа алфавита в произвольном сообщении, лежащая в основе алгоритмов, дали название этим алгоритмам и соответствующим методам. Иногда эти методы называют также префиксными.

К примеру, если имеется некоторый код, который записывается как Х1 = А1А2, и другой код – Х2 = А1, то говорят, что Х2 является префиксом Х1. Или если Х1 = 1010, а Х2 = 10101100, то Х2 также является префиксом Х1. Таким образом, использование описываемых методов предусматривает создание кодовой таблицы (подобно кодам ASCII или base64). Формально процедура сжатия (прямое преобразование) состоит в подстановке соответствующего бинарного кода вместо символа исходного алфавита и наоборот – при обратном преобразовании.

Код Шеннона – Фано не является оптимальным (обеспечивает минимальную избыточность) в общем смысле, хотя и дает оптимальные результаты при некоторых распределениях вероятностей. Для одного и того же распределения вероятностей можно построить, вообще говоря, несколько кодов Шеннона – Фано, и все они могут дать различные результаты.

Итак, необходимо выполнить следующие действия:

1. подсчитать вероятностные параметры символов алфавита А = {ai} (реализуется статическая версия алгоритма);
2. отсортировать – обычно в порядке убывания (невозрастания, т. е. могут иметь место повторяющиеся значения) вероятностей р(аi); р(аi) – вероятность появления в сжимаемом сообщении на произвольной позиции символа аi алфавита, т. е. создать таблицу символов алфавита, на основе которого генерируется сжимаемое сообщение;
3. каждому символу отсортированного множества поставить в соответствие бинарный код, для чего это множество (таблица) символов делится на две группы таким образом, чтобы каждая из групп имела приблизительно одинаковую суммарную частоту (вероятность). Очевидно, на первом шаге такая суммарная вероятность в каждой из групп должна быть максимально близка к 0,5. Первому из полученных подмножеств устанавливается первый символ бинарного кода: 0, второй − 1 (или наоборот). Для вычисления следующих битов кодов данная процедура повторяется рекурсивно для каждого из полученных на текущем шаге подмножеств, в котором содержится больше одного символа. Получим таблицу, в которой длина кодовых комбинаций меняется от минимального (lmin) до максимального (lmax) значений.

Алгоритм прямого преобразования: необходимо выполнить одну операцию: заменить символы входного сообщения соответствующими бинарными кодами.

Алгоритм обратного преобразования: на входе – сообщение в виде бинарной последовательности.

Шаг 1. Анализируются lmin начальных бинарных символов: осуществляется поиск в таблице соответствующего совпадения. Если такое будет найдено, то на выходе будет символ исходного алфавита с совпадающим кодом. После этого процедура повторяется, т. е. анализируются очередные lmin символов. Если не найдено в таблице совпадения, переходим к шагу 2.

Шаг 2. Длина анализируемой последовательности увеличивается на 1 бит: lmin + 1. Осуществляется поиск совпадающей бинарной комбинации такой же длины в таблице. Если такая комбинация существует, на выходе распаковщика формируется соответствующий символ исходного алфавита, если нет – длина анализируемой последовательности увеличивается еще на один бит, и т. д.

По различным причинам при анализе очередной последовательности длиной lmax совпадение в таблице может быть не найдено. Для нейтрализации подобных коллизий архиваторы содержат средства контроля ошибок с помощью корректирующих кодов.

Метод основан на алгоритме оптимального префиксного кодирования алфавита: исходный алгоритм Хаффмана является оптимальным для посимвольного кодирования с известным входным распределением вероятностей, т. е. для отдельного кодирования несвязанных символов в таком потоке данных [30]. Отличается от метода Шеннона – Фано лишь в части кодирования символов исходного алфавита. В данном случае бинарные коды создаются на основе дерева, ветви которого обозначаются бинарными символами. Бинарным кодом символа исходного алфавита будет последовательность обозначений ветвей дерева от корня до листа, соответствующего этому символу. В основе бинарного кода лежит следующее положение.

Лемма. Для любого заданного алфавита (источника) с N > 2 символами существует оптимальный двоичный код, в котором два наименее вероятных символа (слова) имеют одну и ту же длину и отличаются лишь последним битом.

Построение дерева начинается с сортирования символов исходного алфавита в порядке убывания (невозрастания).

Далее выбираются два символа (ai, aj) с наименьшими вероятностями (р(ai), р(aj)) и объединяются в узел. Ветви этого узла обозначаются «1» и «0». Этот узел рассматривается далее как новый, виртуальный символ (aij), которому будет соответствовать вероятность р(aij) = р(ai) + р(aj). Такой виртуальный символ будет рассматриваться далее наравне с остальными символами исходного алфавита. Два его потомка из дальнейшего рассмотрения исключаются. Создаются новые узлы дерева по тому же принципу. Корень дерева образуют два символа с наибольшими вероятностями.

# **Практические задания**

**Задание 1.** Разработать авторское приложение в соответствии с целью лабораторной работы.

|  |
| --- |
| **public void BuildCodes(List<SymbolInfo> symbolInfos)**  **{**  **\_symbolInfos = symbolInfos;**  **if (!\_symbolInfos.Any()) return;**  **double totalFrequency = \_symbolInfos.Sum(s => s.Frequency);**  **if (totalFrequency == 0 && \_symbolInfos.Any())**  **{**  **totalFrequency = \_symbolInfos.Count;**  **\_symbolInfos.ForEach(s => { if (s.Frequency == 0) s.Frequency = 1; });**  **}**  **if (totalFrequency > 0)**  **{**  **foreach (var si in \_symbolInfos)**  **{**  **si.Probability = si.Frequency / totalFrequency;**  **}**  **}**  **else if (\_symbolInfos.Count == 1)**  **{**  **\_symbolInfos.First().Probability = 1.0;**  **}**  **var sortedSymbols = \_symbolInfos.OrderByDescending(s => s.Probability).ToList();**  **Console.WriteLine("\n--- Построение кодов Шеннона-Фано ---");**  **Console.WriteLine("Символы, изначально отсортированные по вероятности:");**  **sortedSymbols.ForEach(s => Console.WriteLine($" '{s.Symbol}' (Вероятность: {s.Probability:F4})"));**  **AssignCodesRecursive(sortedSymbols, "");**  **Codes.Clear();**  **\_reverseCodes.Clear();**  **foreach (var si in \_symbolInfos.Where(s => !string.IsNullOrEmpty(s.Code)))**  **{**  **if (!Codes.ContainsKey(si.Symbol))**  **{**  **Codes.Add(si.Symbol, si.Code);**  **\_reverseCodes.Add(si.Code, si.Symbol);**  **}**  **}**  **Console.WriteLine("\n--- Итоговая таблица кодов ---");**  **foreach (var codeEntry in Codes.OrderBy(kvp => kvp.Key))**  **{**  **Console.WriteLine($"'{codeEntry.Key}': {codeEntry.Value}");**  **}**  **}** |

Листинг 1.1 – Основной метод для построения таблицы кодов Шеннона-Фано

|  |
| --- |
| **private void AssignCodesRecursive(List<SymbolInfo> symbols, string currentCode)**  **{**  **if (!symbols.Any()) return;**  **Console.WriteLine($"\nПоследовательность: [{string.Join(", ", symbols.Select(s => $"'{s.Symbol}'({s.Probability:F4})"))}] с префиксом '{currentCode}'");**  **if (symbols.Count == 1)**  **{**  **symbols[0].Code = currentCode;**  **var originalSymbol = \_symbolInfos.FirstOrDefault(s => s.Symbol == symbols[0].Symbol);**  **if (originalSymbol != null) originalSymbol.Code = currentCode;**  **Console.WriteLine($" -> Листовой узел: '{symbols[0].Symbol}' получает код '{currentCode}'");**  **return;**  **}**  **if (symbols.Count == 0)**  **{**  **return;**  **}**  **int splitIndex = FindSplitPoint(symbols);**  **Console.WriteLine($" Индекс разделения: {splitIndex} (от 0 до {splitIndex} против от {splitIndex + 1} до {symbols.Count - 1})");**  **List<SymbolInfo> group1 = new List<SymbolInfo>();**  **List<SymbolInfo> group2 = new List<SymbolInfo>();**  **for (int i = 0; i < symbols.Count; i++)**  **{**  **if (i <= splitIndex)**  **{**  **group1.Add(symbols[i]);**  **}**  **else**  **{**  **group2.Add(symbols[i]);**  **}**  **}**  **Console.WriteLine($" Группа 1 (получает '0'): [{string.Join(", ", group1.Select(s => $"'{s.Symbol}'"))}]");**  **AssignCodesRecursive(group1, currentCode + "0");**  **Console.WriteLine($" Группа 2 (получает '1'): [{string.Join(", ", group2.Select(s => $"'{s.Symbol}'"))}]");**  **AssignCodesRecursive(group2, currentCode + "1");**  **}** |

Листинг 1.2 – Разделение списка символов и присвоение битов

|  |
| --- |
| **private int FindSplitPoint(List<SymbolInfo> symbols)**  **{**  **if (symbols.Count <= 1) return 0;**  **double totalProbability = symbols.Sum(s => s.Probability);**  **double minDifference = double.MaxValue;**  **int bestSplitIndex = 0;**  **for (int i = 0; i < symbols.Count - 1; i++)**  **{**  **double sumGroup1 = 0;**  **for (int j = 0; j <= i; j++)**  **{**  **sumGroup1 += symbols[j].Probability;**  **}**  **double sumGroup2 = totalProbability - sumGroup1;**  **double difference = Math.Abs(sumGroup1 - sumGroup2);**  **if (difference < minDifference)**  **{**  **minDifference = difference;**  **bestSplitIndex = i;**  **}**  **}**  **return bestSplitIndex;**  **}** |

Листинг 1.3 – Нахождение оптимального индекса для разделения отсортированного списка символов на две подгруппы

|  |
| --- |
| **public string Encode(string source)**  **{**  **if (Codes == null || !Codes.Any())**  **{**  **Console.WriteLine("Ошибка: Таблица кодов не построена или пуста.");**  **return null;**  **}**  **StringBuilder encodedSource = new StringBuilder();**  **Console.WriteLine("\n--- Процесс кодирования ---");**  **Console.WriteLine($"Источник: \"{source}\"");**  **foreach (char c in source)**  **{**  **if (Codes.TryGetValue(c, out string code))**  **{**  **encodedSource.Append(code);**  **Console.WriteLine($" '{c}' -> {code}");**  **}**  **else**  **{**  **Console.WriteLine($" Предупреждение: Символ '{c}' не найден в таблице кодов. Пропуск.");**  **}**  **}**  **Console.WriteLine($"Закодировано: {encodedSource.ToString()}");**  **return encodedSource.ToString();**  **}** |

Листинг 1.4 – Кодирование входной строки

|  |
| --- |
| **public string Decode(string encodedSource)**  **{**  **if (\_reverseCodes == null || !\_reverseCodes.Any())**  **{**  **Console.WriteLine("Ошибка: Обратная таблица кодов не построена или пуста.");**  **return null;**  **}**  **StringBuilder decodedSource = new StringBuilder();**  **StringBuilder currentCode = new StringBuilder();**  **Console.WriteLine("\n--- Процесс декодирования ---");**  **Console.WriteLine($"Закодированный источник: {encodedSource}");**  **foreach (char bit in encodedSource)**  **{**  **currentCode.Append(bit);**  **Console.Write($" Текущие биты: {currentCode}");**  **if (\_reverseCodes.TryGetValue(currentCode.ToString(), out char decodedChar))**  **{**  **decodedSource.Append(decodedChar);**  **Console.WriteLine($" -> Совпадение! Декодировано: '{decodedChar}'");**  **currentCode.Clear();**  **}**  **else**  **{**  **Console.WriteLine(" -> Пока нет совпадений.");**  **}**  **}**  **Console.WriteLine($"Декодировано: \"{decodedSource.ToString()}\"");**  **return decodedSource.ToString();**  **}** |

Листинг 1.5 – Декодирование бинарной строки

**Задание 2.** С помощью приложения выполнить прямое и обратное преобразования сообщения, состоящего из собственных имени и фамилии.

При этом таблица отсортированных символов строится: а) на основе данных, полученных в лабораторной работе № 2; б) динамически, на основе анализа сжимаемого сообщения.

Входящая последовательность: "водчицанастасия"

Строка "водчицанастасия" содержит следующие уникальные символы:  
а, в, д, и, н, о, с, т, ц, ч, я.

Алгоритм требует, чтобы символы были отсортированы по убыванию вероятности. Символы , отсортированные по вероятности:

'а' (Вероятность: 0,2065)

'н' (Вероятность: 0,1915)

'о' (Вероятность: 0,1294)

'и' (Вероятность: 0,1226)

'д' (Вероятность: 0,1199)

'т' (Вероятность: 0,0969)

'с' (Вероятность: 0,0528)

'в' (Вероятность: 0,0277)

'ч' (Вероятность: 0,0261)

'ц' (Вероятность: 0,0196)

'я' (Вероятность: 0,0069)

Начальный список символов (отсортированный):  
[('а', 0.2065), ('н', 0.1915), ('о', 0.1294), ('и', 0.1226), ('д', 0.1199), ('т', 0.0969), ('с', 0.0528), ('в', 0.0277), ('ч', 0.0261), ('ц', 0.0196), ('я', 0.0069)]  
Общая сумма вероятностей этих символов 1.0.

Далее происходит построение двоичного дерева, где каждому левому переходу соответствует '0', а правому – '1'.

**Итерация 1.**

Текущий список: ['а'(0,2065), 'н'(0,1915), 'о'(0,1294), 'и'(0,1226), 'д'(0,1199), 'т'(0,0969), 'с'(0,0528), 'в'(0,0277), 'ч'(0,0261), 'ц'(0,0196), 'я'(0,0069)] с префиксом ''''. Нам нужно разделить этот список на две части так, чтобы суммы вероятностей в них были максимально близки. Сумма вероятностей всех символов = 1.0. Разделяем после символа 'о'.

Группа 1 (получит '0'): [('а', 0.2065), ('н', 0.1915), ('о', 0.1294)]. Сумма вероятностей: 0.2065 + 0.1915 + 0.1294 = 0.5274.

Группа 2 (получит '1'): [('и', 0.1226), ('д', 0.1199), ('т', 0.0969), ('с', 0.0528), ('в', 0.0277), ('ч', 0.0261), ('ц', 0.0196), ('я', 0.0069)].

Сумма вероятностей: 0.1226 + ... + 0.0069 = 0.4725. Разница сумм |0.5274 - 0.4725| = 0.0549. Это оптимальное разделение для данного списка.

**Итерация 1.1 (Обработка Группы 1 с префиксом "0"):**

Текущий список: [('а', 0.2065), ('н', 0.1915), ('о', 0.1294)]. Сумма вероятностей = 0.5274. Разделяем после символа 'а'.

Группа 1.1.1 (получит '0'): [('а', 0.2065)]. Сумма вероятностей: 0.2065

Группа 1.1.2 (получит '1'): [('н', 0.1915), ('о', 0.1294)]. Сумма вероятностей: 0.1915 + 0.1294 = 0.3209

Разница сумм |0.2065 - 0.3209| = 0.1144.

**Итерация 1.1.1 (Обработка Группы 1.1.1 с префиксом "00"):**

Текущий список: [('а', 0.2065)].

В группе один символ. Это конечный узел (лист дерева). Присваиваем код: 'а' -> "00"

**Итерация 1.1.2 (Обработка Группы 1.1.2 с префиксом "01"):**

Текущий список: [('н', 0.1915), ('о', 0.1294)]. Сумма вероятностей = 0.3209. Разделяем после символа 'н'.

Группа 1.1.2.1 (получит '0'): [('н', 0.1915)]

Группа 1.1.2.2 (получит '1'): [('о', 0.1294)]

**Итерация 1.1.2.1 (Обработка Группы 1.1.2.1 с префиксом "010"):**

Текущий список: [('н', 0.1915)]

Присваиваем код: 'н' -> "010"

**Итерация 1.1.2.2 (Обработка Группы 1.1.2.2 с префиксом "011"):**

Текущий список: [('о', 0.1294)]

Присваиваем код: 'о' -> "011"

**Итерация 1.2 (Обработка Группы 2 с префиксом "1"):**

Текущий список: [('и', 0.1226), ('д', 0.1199), ('т', 0.0969), ('с', 0.0528), ('в', 0.0277), ('ч', 0.0261), ('ц', 0.0196), ('я', 0.0069)]. Сумма вероятностей = 0.4725. Разделяем после символа 'д'.

Группа 1.2.1 (получит '0'): [('и', 0.1226), ('д', 0.1199)]. Сумма вероятностей: 0.1226 + 0.1199 = 0.2425

Группа 1.2.2 (получит '1'): [('т', 0.0969), ('с', 0.0528), ('в', 0.0277), ('ч', 0.0261), ('ц', 0.0196), ('я', 0.0069)]. Сумма вероятностей: 0.0969 + ... + 0.0069 = 0.2300

Разница сумм |0.2425 - 0.2300| = 0.0125.

**Итерация 1.2.1 (Обработка Группы 1.2.1 с префиксом "10"):**

Текущий список: [('и', 0.1226), ('д', 0.1199)]

Группа 1.2.1.1 (получит '0'): [('и', 0.1226)] -> Присваиваем код: 'и' -> "100"

Группа 1.2.1.2 (получит '1'): [('д', 0.1199)] -> Присваиваем код: 'д' -> "101"

**Итерация 1.2.2 (Обработка Группы 1.2.2 с префиксом "11"):**

Текущий список: [('т', 0.0969), ('с', 0.0528), ('в', 0.0277), ('ч', 0.0261), ('ц', 0.0196), ('я', 0.0069)]. Сумма вероятностей = 0.2300. Разделяем после символа 'т'.

Группа 1.2.2.1 (получит '0'): [('т', 0.0969)]. Сумма вероятностей: 0.0969  
-> Присваиваем код: 'т' -> "110"

Группа 1.2.2.2 (получит '1'): [('с', 0.0528), ('в', 0.0277), ('ч', 0.0261), ('ц', 0.0196), ('я', 0.0069)]. Сумма вероятностей: 0.1331

**Итерация 1.2.2.2 (Обработка Группы 1.2.2.2 с префиксом "111"):**

**Текущий список:** [('с', 0.0528), ('в', 0.0277), ('ч', 0.0261), ('ц', 0.0196), ('я', 0.0069)]. Сумма вероятностей = 0.1331. Разделяем после символа 'с'.

Группа 1.2.2.2.1 (получит '0'): [('с', 0.0528)] -> Присваиваем код: 'с' -> "1110"

Группа 1.2.2.2.2 (получит '1'): [('в', 0.0277), ('ч', 0.0261), ('ц', 0.0196), ('я', 0.0069)] (Префикс "1111")

**Итерация 1.2.2.2.2 (Обработка Группы 1.2.2.2.2 с префиксом "1111"):**

Список: [('в', 0.0277), ('ч', 0.0261), ('ц', 0.0196), ('я', 0.0069)]

'в' (0.0277) получает '0' (код "11110")

Остальные [('ч', 0.0261), ('ц', 0.0196), ('я', 0.0069)] получают '1' (префикс "11111")

**Дальнейшие шаги для префикса "11111":**

Список: [('ч', 0.0261), ('ц', 0.0196), ('я', 0.0069)]

'ч' (0.0261) получает '0' (код "111110")

Остальные [('ц', 0.0196), ('я', 0.0069)] получают '1' (префикс "111111")

Список: [('ц', 0.0196), ('я', 0.0069)]

'ц' (0.0196) получает '0' (код "1111110")

'я' (0.0069) получает '1' (код "1111111")

Собрав все присвоенные коды, получаем:

'а': 00

'в': 11110

'д': 101

'и': 100

'н': 010

'о': 011

'с': 1110

'т': 110

'ц': 1111110

'ч': 111110

'я': 1111111

Это и есть коды Шеннона-Фано для данной последовательности символов и их вероятностей. Как видно, символы с большей вероятностью ('а', 'н', 'о') получили более короткие коды, а символы с меньшей вероятностью ('ц', 'я') – более длинные, что и является целью алгоритма для достижения сжатия.

Процесс кодирования и декодирования затем использует эту таблицу:

**Кодирование "водчицанастасия":** Просто заменяем каждый символ его кодом: в (11110) о (011) д (101) ч (111110) и (100) ц (1111110) а (00) н (010) а (00) с (1110) т (110) а (00) с (1110) и (100) я (1111111)  
Результат: 11110011101111110100111111000010001110110001110100111111

**Декодирование**: читаем биты последовательно, пока не найдем совпадение в нашей таблице кодов:

Текущие биты: 1 -> Пока нет совпадений.

Текущие биты: 11 -> Пока нет совпадений.

Текущие биты: 111 -> Пока нет совпадений.

Текущие биты: 1111 -> Пока нет совпадений.

Текущие биты: 11110 -> Совпадение! Декодировано: 'в'

Текущие биты: 0 -> Пока нет совпадений.

Текущие биты: 01 -> Пока нет совпадений.

Текущие биты: 011 -> Совпадение! Декодировано: 'о'

Текущие биты: 1 -> Пока нет совпадений.

Текущие биты: 10 -> Пока нет совпадений.

Текущие биты: 101 -> Совпадение! Декодировано: 'д'

Текущие биты: 1 -> Пока нет совпадений.

Текущие биты: 11 -> Пока нет совпадений.

Текущие биты: 111 -> Пока нет совпадений.

Текущие биты: 1111 -> Пока нет совпадений.

Текущие биты: 11111 -> Пока нет совпадений.

Текущие биты: 111110 -> Совпадение! Декодировано: 'ч'

Текущие биты: 1 -> Пока нет совпадений.

Текущие биты: 10 -> Пока нет совпадений.

Текущие биты: 100 -> Совпадение! Декодировано: 'и'

Текущие биты: 1 -> Пока нет совпадений.

Текущие биты: 11 -> Пока нет совпадений.

Текущие биты: 111 -> Пока нет совпадений.

Текущие биты: 1111 -> Пока нет совпадений.

Текущие биты: 11111 -> Пока нет совпадений.

Текущие биты: 111111 -> Пока нет совпадений.

Текущие биты: 1111110 -> Совпадение! Декодировано: 'ц'

Текущие биты: 0 -> Пока нет совпадений.

Текущие биты: 00 -> Совпадение! Декодировано: 'а'

Текущие биты: 0 -> Пока нет совпадений.

Текущие биты: 01 -> Пока нет совпадений.

Текущие биты: 010 -> Совпадение! Декодировано: 'н'

Текущие биты: 0 -> Пока нет совпадений.

Текущие биты: 00 -> Совпадение! Декодировано: 'а'

Текущие биты: 1 -> Пока нет совпадений.

Текущие биты: 11 -> Пока нет совпадений.

Текущие биты: 111 -> Пока нет совпадений.

Текущие биты: 1110 -> Совпадение! Декодировано: 'с'

Текущие биты: 1 -> Пока нет совпадений.

Текущие биты: 11 -> Пока нет совпадений.

Текущие биты: 110 -> Совпадение! Декодировано: 'т'

Текущие биты: 0 -> Пока нет совпадений.

Текущие биты: 00 -> Совпадение! Декодировано: 'а'

Текущие биты: 1 -> Пока нет совпадений.

Текущие биты: 11 -> Пока нет совпадений.

Текущие биты: 111 -> Пока нет совпадений.

Текущие биты: 1110 -> Совпадение! Декодировано: 'с'

Текущие биты: 1 -> Пока нет совпадений.

Текущие биты: 10 -> Пока нет совпадений.

Текущие биты: 100 -> Совпадение! Декодировано: 'и'

Текущие биты: 1 -> Пока нет совпадений.

Текущие биты: 11 -> Пока нет совпадений.

Текущие биты: 111 -> Пока нет совпадений.

Текущие биты: 1111 -> Пока нет совпадений.

Текущие биты: 11111 -> Пока нет совпадений.

Текущие биты: 111111 -> Пока нет совпадений.

Текущие биты: 1111111 -> Совпадение! Декодировано: 'я'

Декодировано: "водчицанастасия"

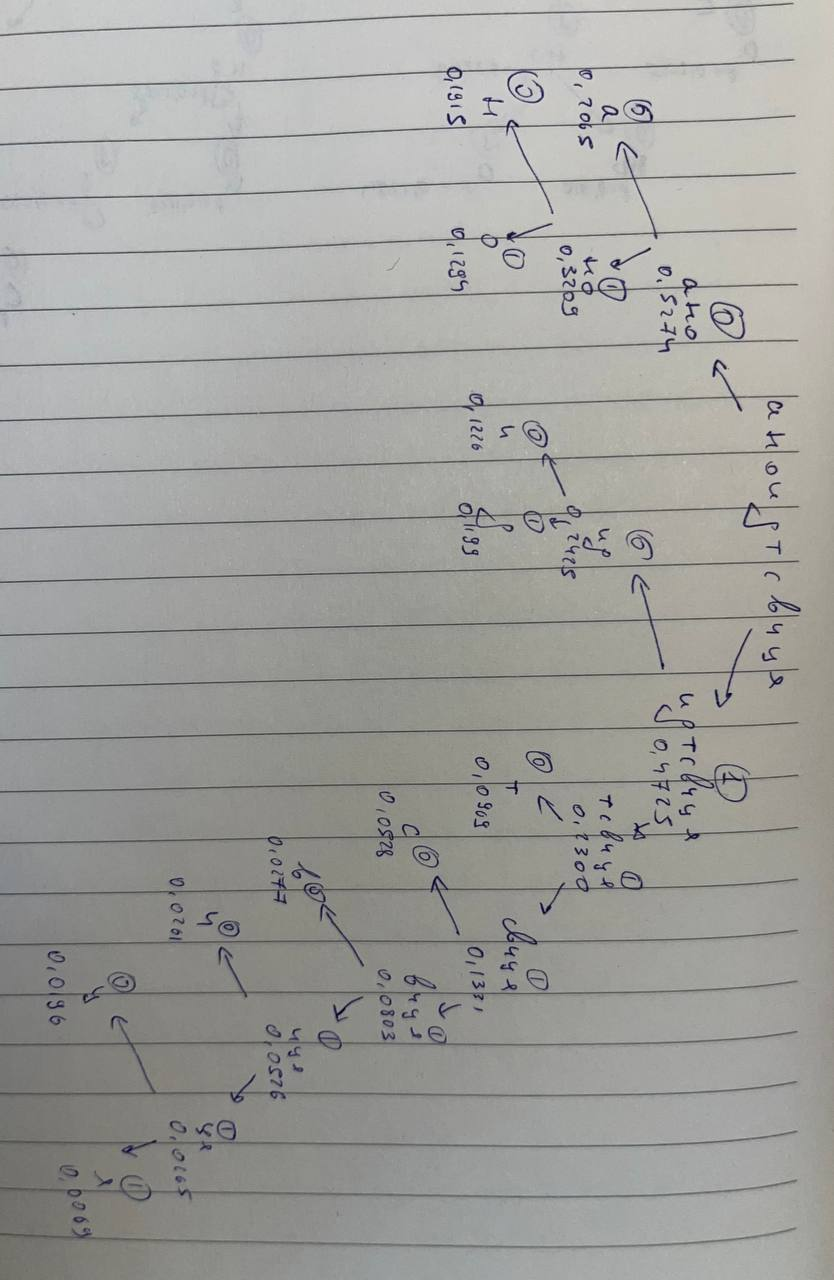


Рисунок 2.1 – Построение дерева групп

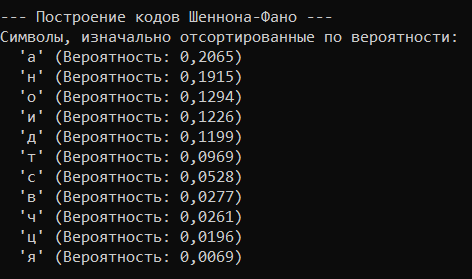


Рисунок 2.2 – Вероятность символов (статическая)

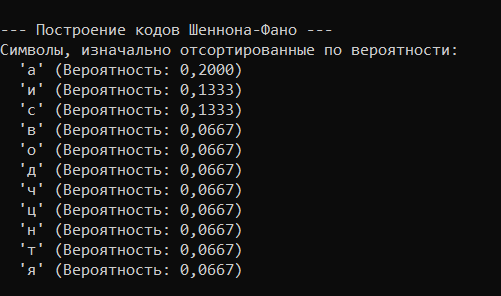


Рисунок 2.3 – Вероятность символов (динамическая)

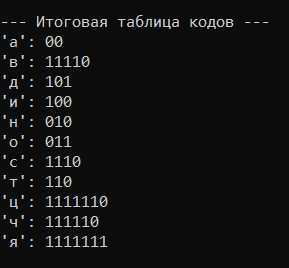


Рисунок 2.4 – Итоговая таблица кодов

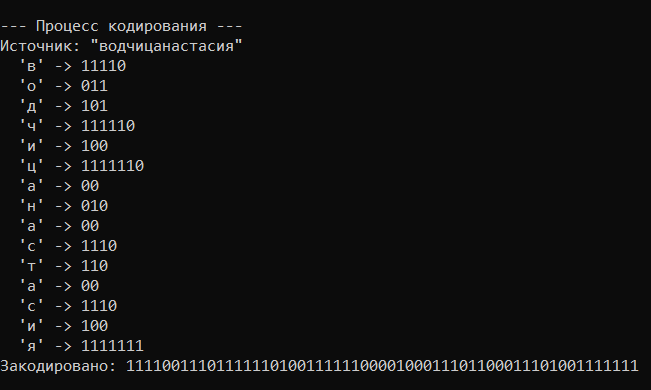


Рисунок 2.5 – Процесс кодирования

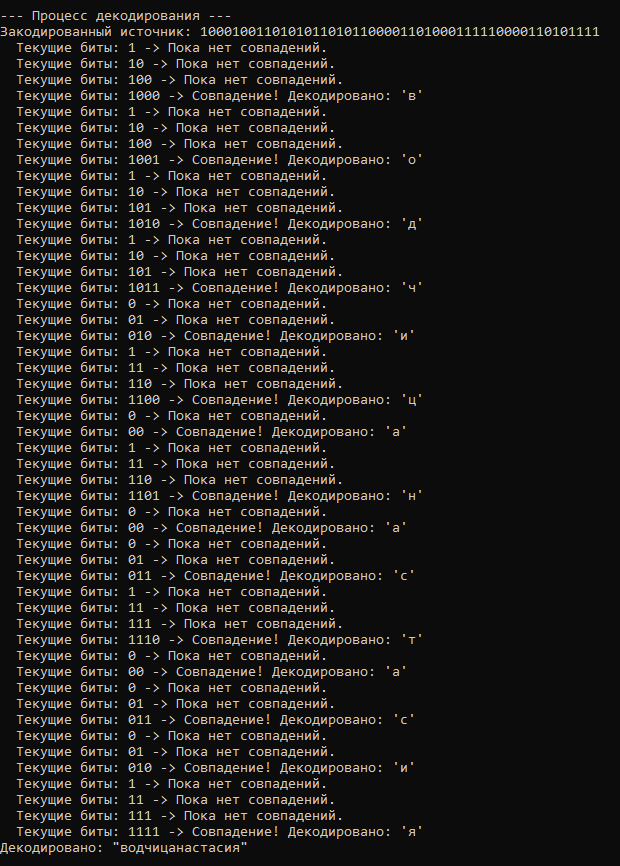


Рисунок 2.6 – Процесс декодирования

**Задание 3.** Определить эффективность (в сравнении с кодами ASCII) сжатия сообщения.

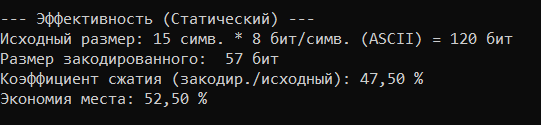


Рисунок 3.1 – Подсчёт эффективности

**Вывод**: лабораторная работа позволила на практике освоить метод сжатия данных Шеннона-Фано, продемонстрировать его способность уменьшать объем информации за счет использования статистических свойств данных без потерь. Были закреплены теоретические знания о принципах его работы и особенностях построения префиксных кодов.