Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

**Отчёт**

Лабораторная работа №11 «Сжатие/распаковка данных арифметическим методом»

Студент: Водчиц Анастасия

ФИТ 3 курс 1 группа

Преподаватель: Нистюк О.А.

Минск 2025

**Цель**: приобретение практических навыков использования арифметических методов сжатия/распаковки данных.

**Задачи**:

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

# Теоретические сведения

Как мы установили, вероятностные (префиксные) методы являются достаточно простыми и эффективными: они основаны на использовании кодов переменной длины и для вероятностей появления символов алфавита, кратных степеням числа 2 (1/2, 1/4, 1/8 и т. п.), дают наилучшие результаты. При других значениях вероятностей, как правило, самый короткий код получается большим, чем двоичный логарифм этой вероятности (взятый с отрицательным знаком). Например, при р(аi) = 0,4 получим −log20,15 = 1,32. Понятно, что мы не можем закодировать этот символ только 2 битами (либо одним), т. е. решение не всегда является оптимальным. Анализируемого недостатка лишены арифметические методы.

Пpи арифметическом сжатии (кодиpовании) текст пpедставляется вещественными числами в интеpвале от 0 до 1. По меpе анализа текста отобpажающий его интеpвал уменьшается, а количество битов для его пpедставления возpастает. Очеpедные символы текста сокpащают величину интеpвала, исходя из значений соответствующих веpоятностей.

Основная идея арифметического метода сжатия заключается в том, чтобы присваивать коды не отдельным символам, а их последовательностям.

Таким образом, как и во всех энтропийных алгоритмах, исходной является информация о частоте встречаемости каждого символа алфавита. Алгоритмы прямого и обратного преобразований базируются на операциях с «рабочим отрезком».

# Практические задания

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

|  |
| --- |
| **public decimal Encode(string textToEncode, List<SymbolInfo> symbolInfoTable, bool verbose = false)**  **{**  **decimal low = 0.0m;**  **decimal high = 1.0m;**  **if (verbose) Console.WriteLine("\n--- Шаги кодирования ---");**  **if (verbose) Console.WriteLine($"Начальный интервал: Low = {low:F28}, High = {high:F28}");**  **int step = 1;**  **foreach (char symbol in textToEncode)**  **{**  **decimal currentRange = high - low;**  **SymbolInfo sInfo = symbolInfoTable.FirstOrDefault(s => s.Symbol == symbol);**  **if (sInfo == null) throw new ArgumentException($"Символ '{symbol}' не найден в таблице вероятностей.");**  **decimal oldLow = low;**  **decimal oldHigh = high;**  **high = low + currentRange \* sInfo.High;**  **low = low + currentRange \* sInfo.Low;**  **if (verbose)**  **{**  **Console.WriteLine($"Шаг {step}: Кодируем символ '{symbol}' (L0={sInfo.Low:F6}, H0={sInfo.High:F6})");**  **Console.WriteLine($" Предыдущий интервал: Low = {oldLow:F28}, High = {oldHigh:F28}, Range = {currentRange:F28}");**  **Console.WriteLine($" Новый интервал: Low = {low:F28}, High = {high:F28}");**  **}**  **step++;**  **}**  **if (verbose) Console.WriteLine("--- Конец кодирования ---");**  **return low;**  **}** |

Листинг 1.1 – Метод для прямого преобразования

|  |
| --- |
| **public string Decode(decimal encodedValue, int originalLength, List<SymbolInfo> symbolInfoTable, bool verbose = false)**  **{**  **StringBuilder decodedText = new StringBuilder();**  **decimal low = 0.0m; decimal high = 1.0m;**  **const decimal MIN\_COMPARISON\_PRECISION = 0.0000000000000000000000000001m;**  **if (verbose) Console.WriteLine("\n--- Шаги декодирования ---");**  **if (verbose) Console.WriteLine($"Начальное значение для декодирования: {encodedValue:F28}");**  **if (verbose) Console.WriteLine($"Начальный интервал: Low = {low:F28}, High = {high:F28}");**  **for (int i = 0; i < originalLength; i++)**  **{**  **decimal currentRange = high - low;**  **if (verbose) Console.WriteLine($"\nШаг {i + 1}:");**  **if (verbose) Console.WriteLine($" Текущий интервал: Low = {low:F28}, High = {high:F28}, Range = {currentRange:F28}");**  **if (currentRange < MIN\_COMPARISON\_PRECISION)**  **{**  **Console.WriteLine($" Критически малый диапазон: {currentRange}. Декодирование может быть неточным.");**  **SymbolInfo emergencySymbol = symbolInfoTable.OrderBy(s => Math.Abs(encodedValue - (low + currentRange \* s.Low))).First();**  **decodedText.Append(emergencySymbol.Symbol);**  **if (verbose) Console.WriteLine($" Аварийно выбран символ '{emergencySymbol.Symbol}' из-за малого диапазона.");**  **decimal temp\_high = low + currentRange \* emergencySymbol.High;**  **low = low + currentRange \* emergencySymbol.Low;**  **high = temp\_high; continue;**  **}**  **decimal valueInCurrentRange = (encodedValue - low) / currentRange;**  **if (verbose) Console.WriteLine($" Масштабированное значение (encodedValue - Low) / Range = {valueInCurrentRange:F28}");**  **SymbolInfo foundSymbol = null;**  **foreach (var sInfo in symbolInfoTable)**  **{**  **if (valueInCurrentRange >= sInfo.Low && (valueInCurrentRange < sInfo.High || (Math.Abs(valueInCurrentRange - sInfo.High) < MIN\_COMPARISON\_PRECISION && sInfo.High == 1.0m)))**  **{**  **foundSymbol = sInfo; break;**  **}**  **}**  **if (foundSymbol == null)**  **{**  **if (Math.Abs(valueInCurrentRange - 1.0m) < MIN\_COMPARISON\_PRECISION && symbolInfoTable.Last().High == 1.0m)**  **{**  **foundSymbol = symbolInfoTable.Last();**  **}**  **else**  **{**  **Console.WriteLine($" Предупреждение: Символ не найден для масштабированного значения {valueInCurrentRange}. EncodedValue: {encodedValue}");**  **foundSymbol = symbolInfoTable.OrderBy(s => Math.Abs(valueInCurrentRange - s.Low)).First();**  **Console.WriteLine($" Аварийно выбран ближайший символ по L0: '{foundSymbol.Symbol}'");**  **}**  **}**  **decodedText.Append(foundSymbol.Symbol);**  **if (verbose) Console.WriteLine($" Найден символ: '{foundSymbol.Symbol}' (L0={foundSymbol.Low:F6}, H0={foundSymbol.High:F6})");**  **decimal oldLow = low;**  **high = low + currentRange \* foundSymbol.High;**  **low = oldLow + currentRange \* foundSymbol.Low;**  **if (verbose) Console.WriteLine($" Обновленный интервал: Low = {low:F28}, High = {high:F28}");**  **}**  **if (verbose) Console.WriteLine("--- Конец декодирования ---");**  **return decodedText.ToString();**  **}** |

Листинг 1.2 – Метод для обратного преобразование

**Задание 2.** С помощью приложения выполнить прямое и обратное преобразования сообщений в соответствии с таблицей.

Первая часть: кодирование/декодирование сообщения «мультимиллионер».

Вторая часть: кодирование/декодирование составного сообщения, полученного конкатенацией последовательностей «мультимиллионер» и «семенохранилище».

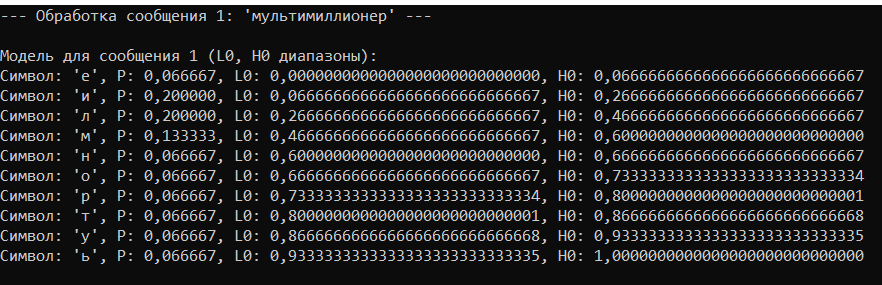


Рисунок 2.1 – Первая часть: список интервалов

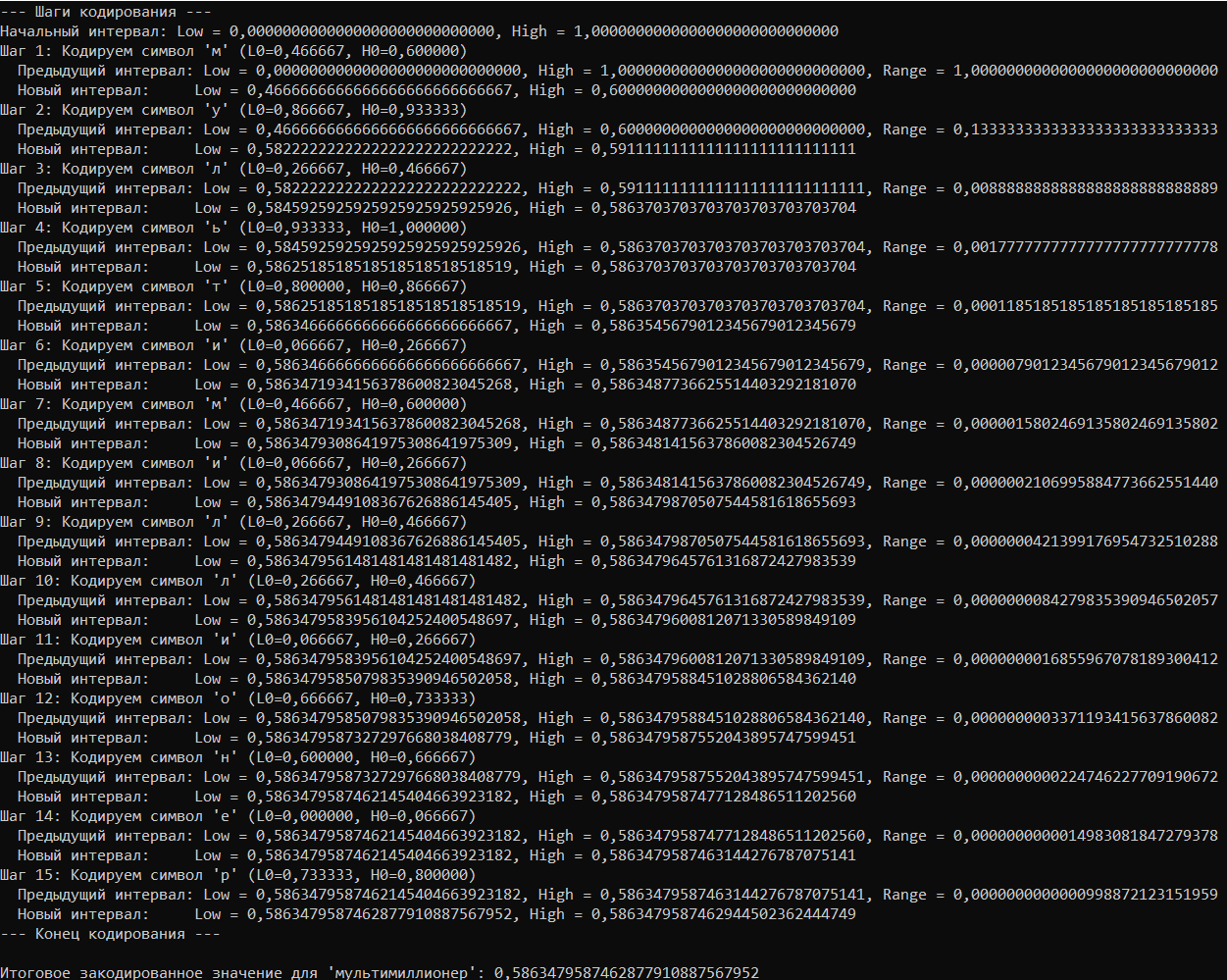


Рисунок 2.2 – Первая часть: кодирование

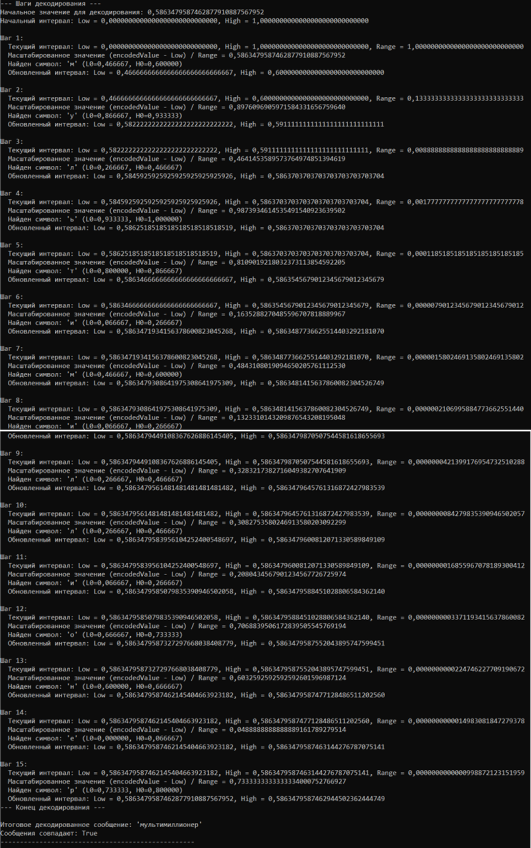


Рисунок 2.3 – Первая часть: декодирование

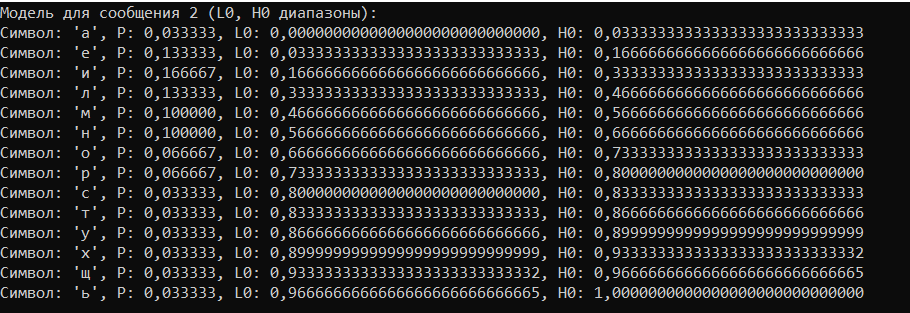


Рисунок 2.4 – Вторая часть: список интервалов

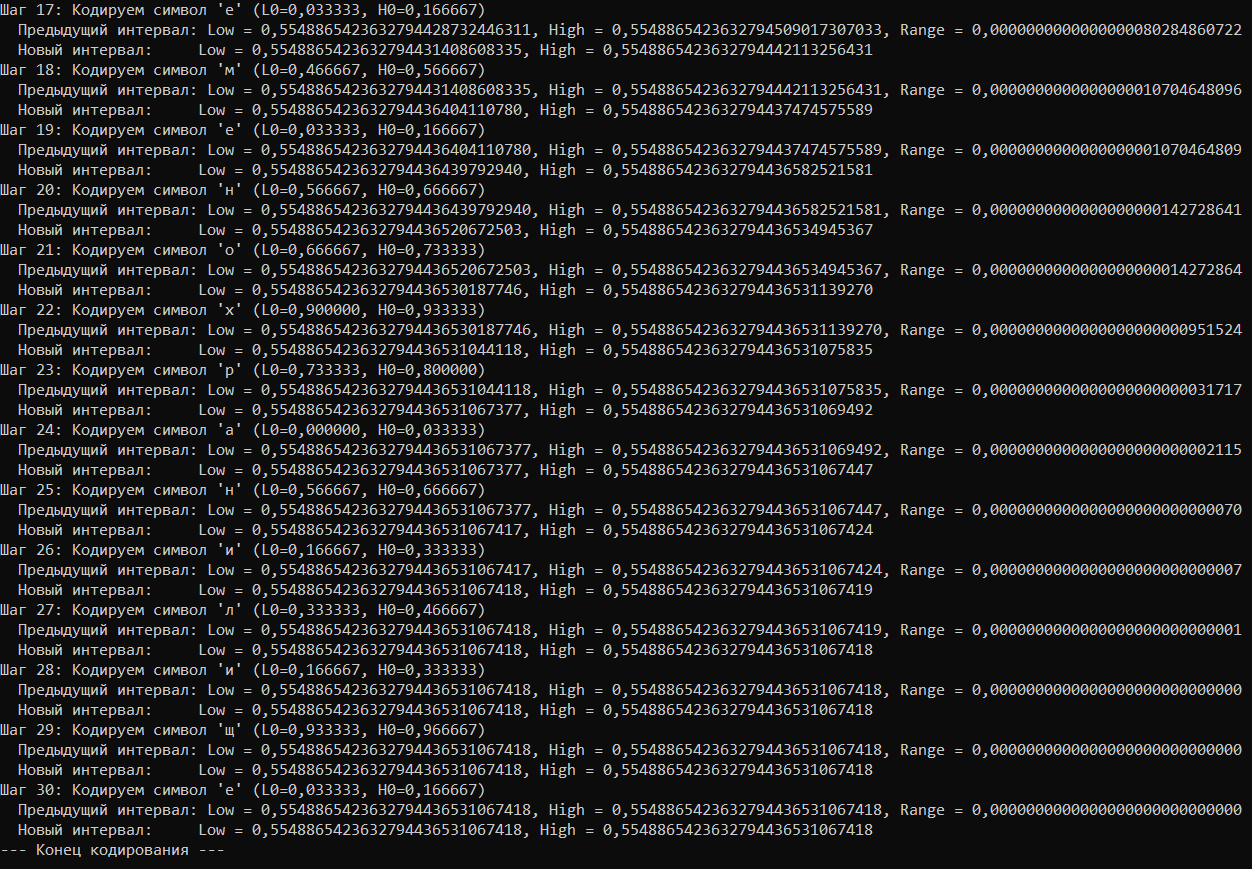
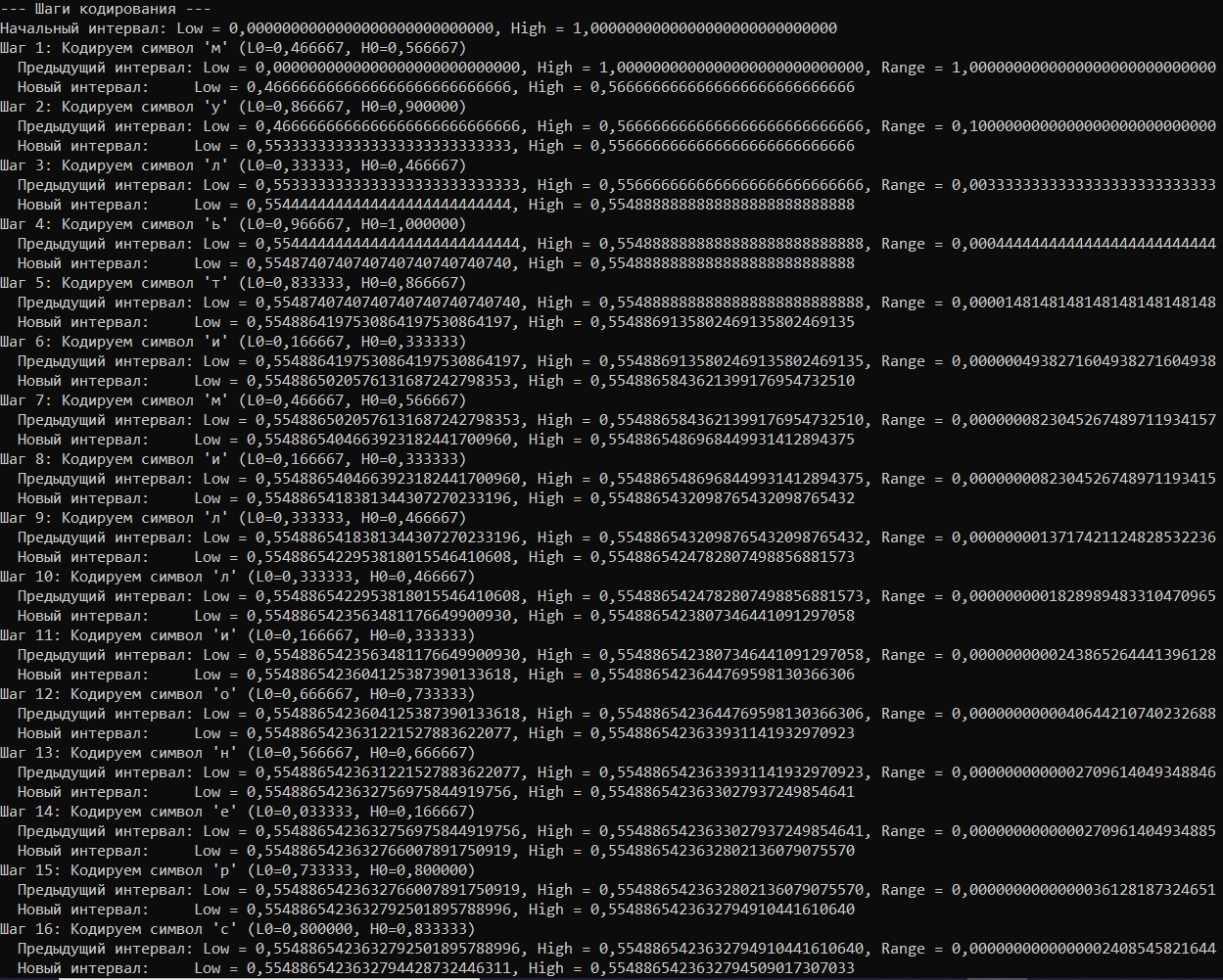
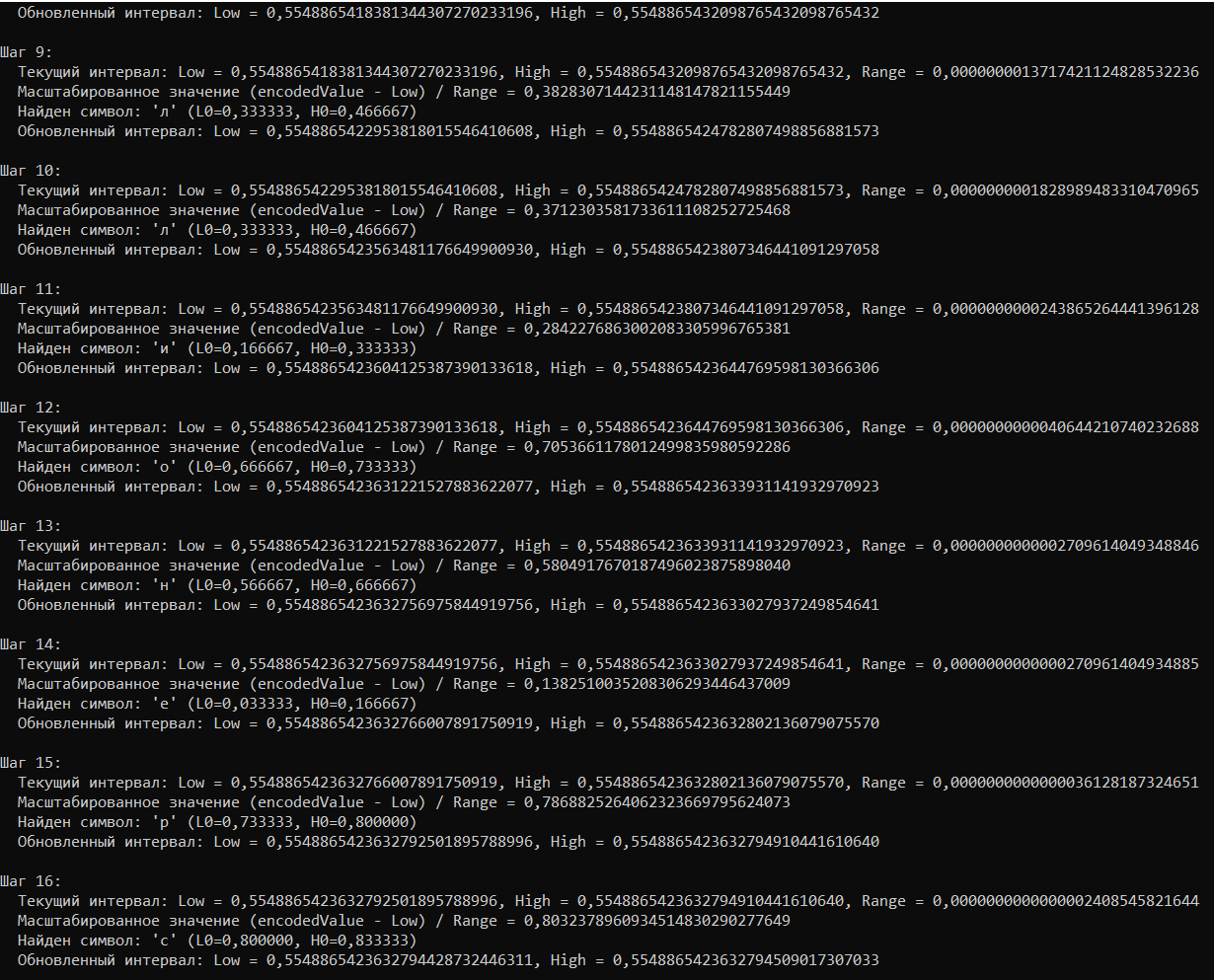
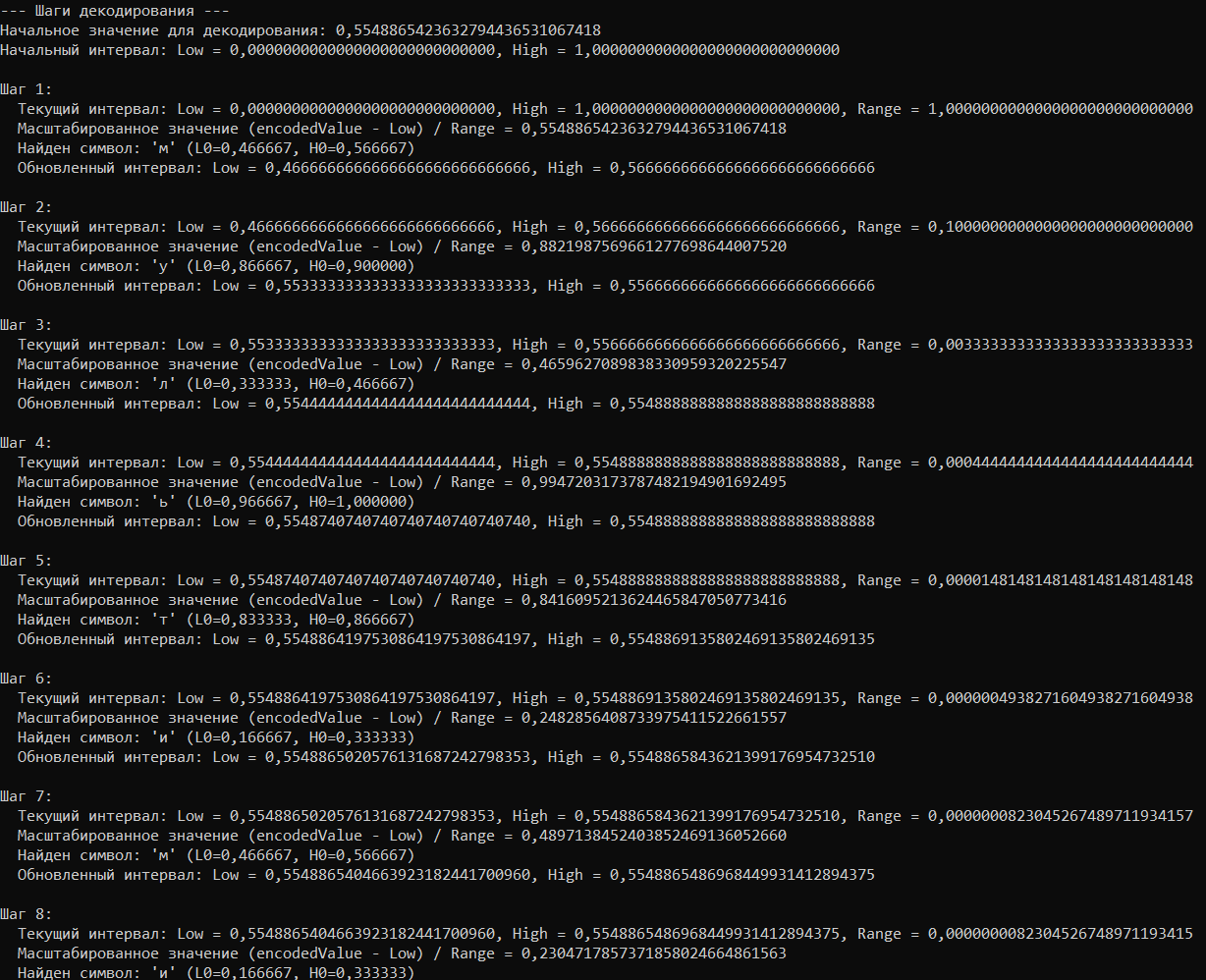


Рисунок 2.5 – Вторая часть: кодирование



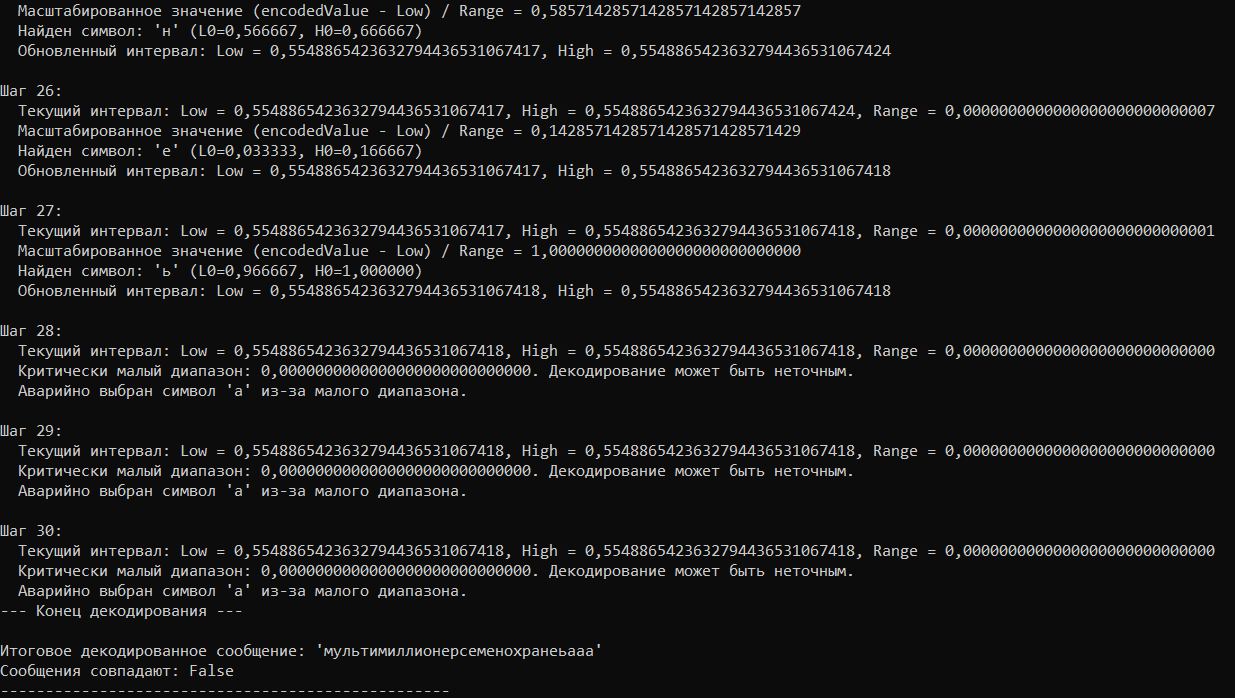
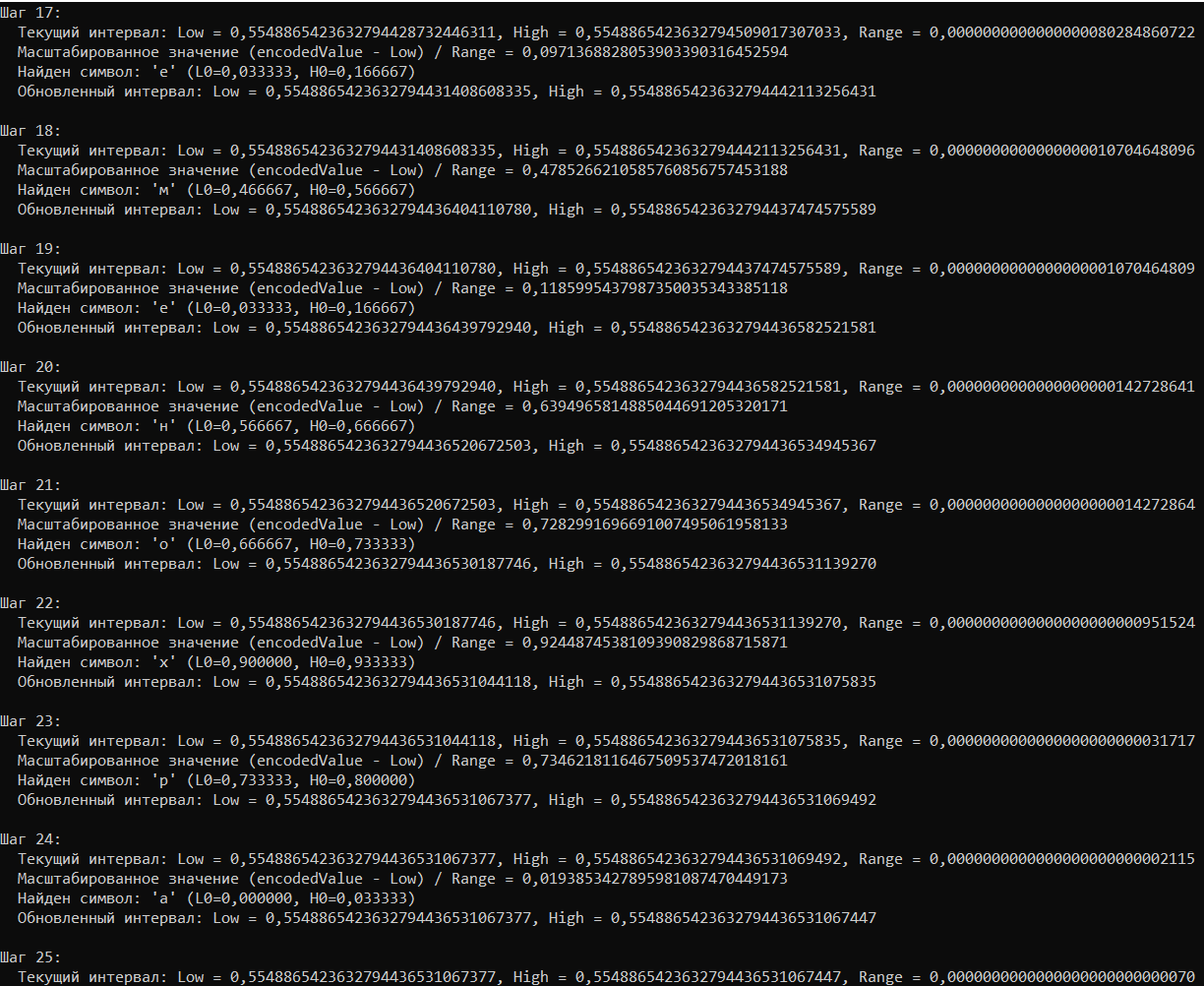


Рисунок 2.6 – Вторая часть: декодирование

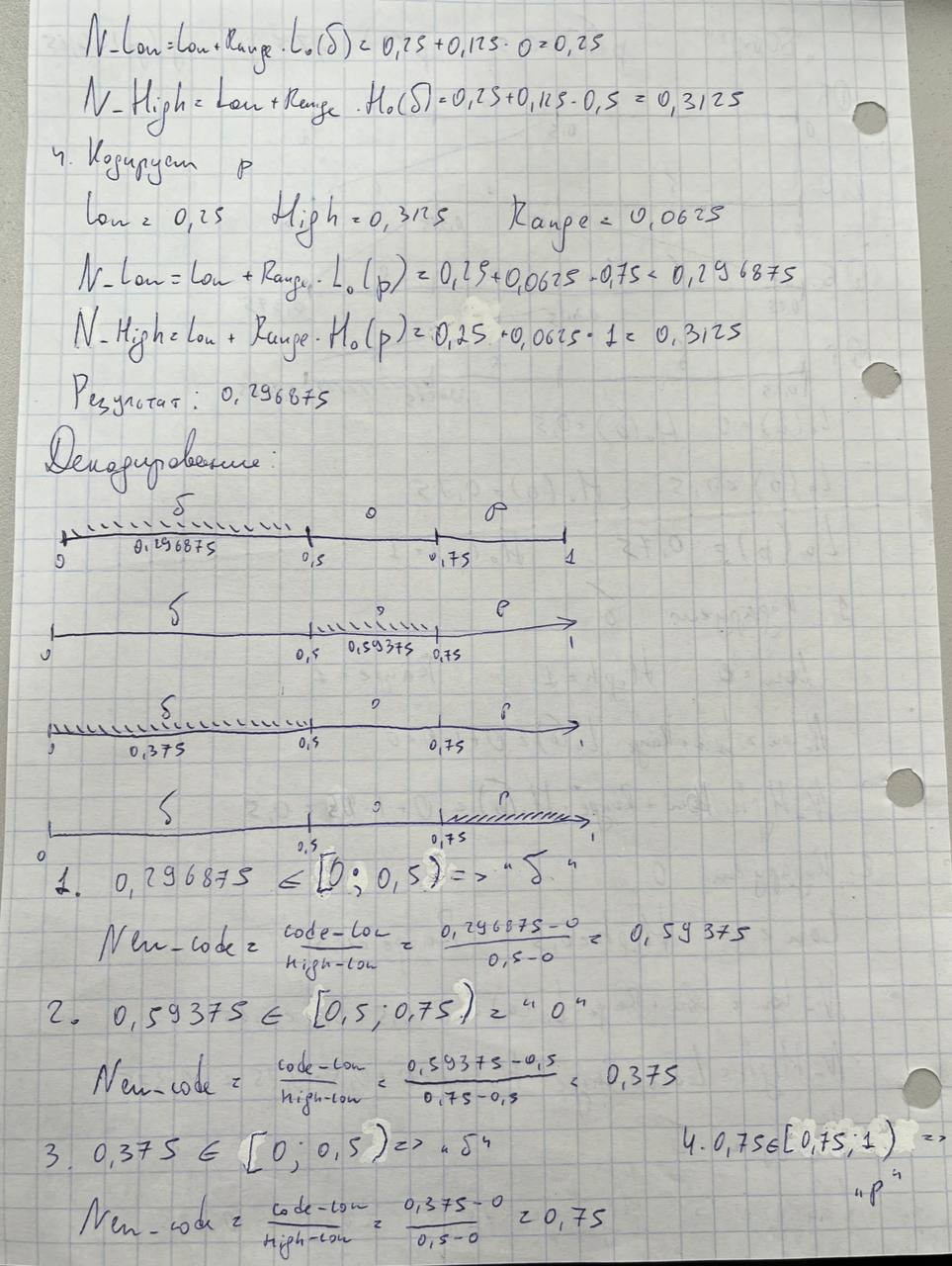
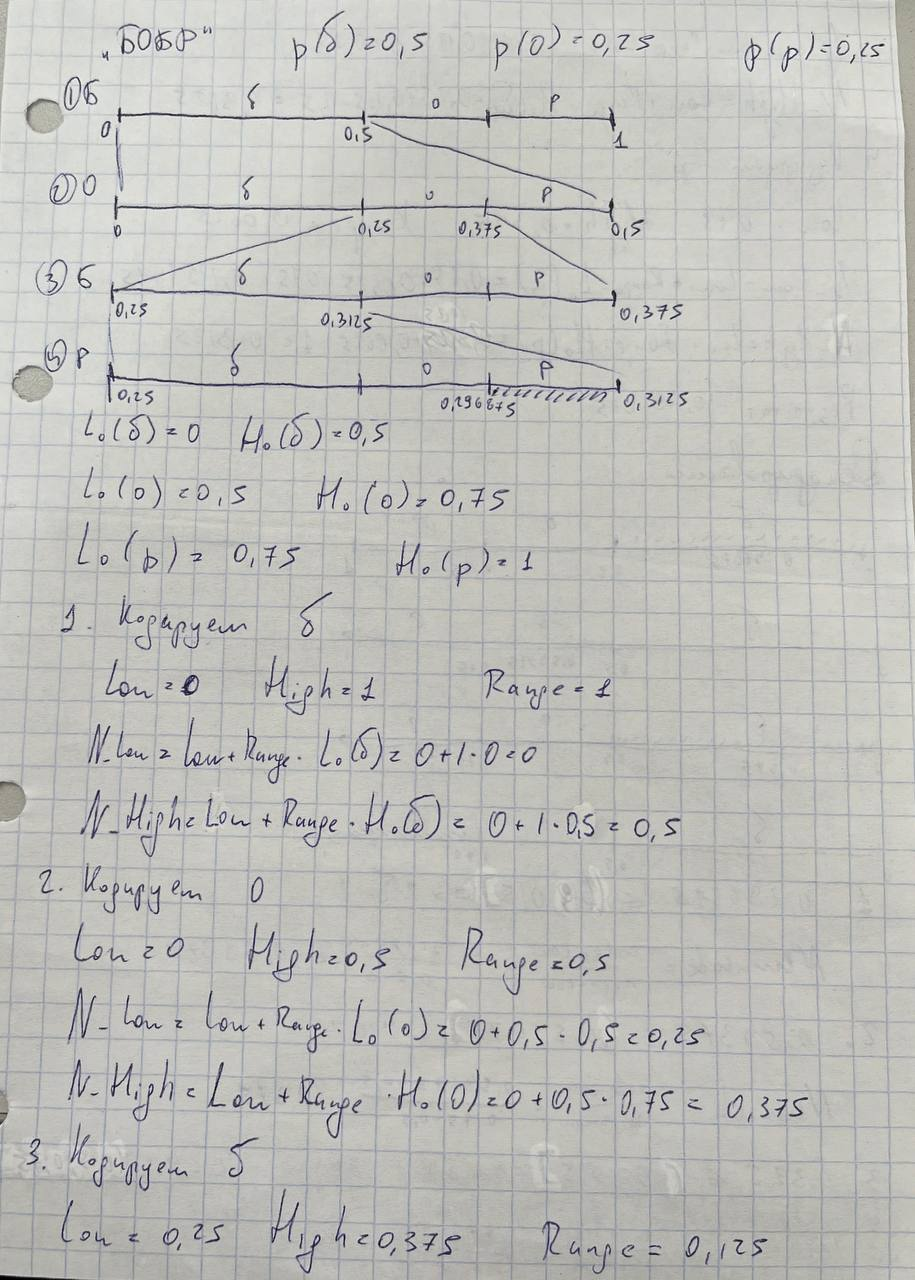


Рисунок 2.3 – Пример прямого и обратного преобразования

**Задание 3.** Дать оценку возможности переполнения при выполнении вычислений.

Да, возможность переполнения при выполнении вычислений в арифметическом кодировании — это существенный аспект, который необходимо учитывать.

Арифметическое кодирование работает путем представления сообщения в виде интервала [a, b] внутри отрезка [0, 1]. С каждым кодируемым символом этот интервал сужается. Длины этих сужающихся интервалов пропорциональны вероятностям символов.

Когда интервал [a, b] становится очень маленьким, разница (b - a) может стать меньше, чем позволяет точность используемых чисел (например, чисел с плавающей запятой). Это может привести к тому, что a и b станут неразличимы, или интервал перестанет сужаться должным образом.

На практике арифметическое кодирование часто реализуется с использованием целочисленной арифметики для избежания проблем с плавающей запятой и для скорости. В этом случае интервал [0, 1] масштабируется до большого целочисленного диапазона. Границы интервала a и b также становятся целыми числами. При вычислениях (умножения для определения новых границ) эти целые числа могут превысить максимальное значение, которое может хранить используемый тип данных (например, 32-битное или 64-битное целое). Это и есть классическое переполнение.

Методы борьбы:

* Ренормализация (масштабирование). Когда текущий интервал [a, b] полностью попадает в одну из половин [0, 0.5) или [0.5, 1), или когда старшие биты a и b совпадают, эти биты можно вывести в качестве части сжатого кода, а оставшийся интервал расширить (например, удвоить), чтобы предотвратить чрезмерное сужение и потерю точности. Это позволяет продолжать кодирование длинных последовательностей.
* Использование арифметики высокой точности. Можно использовать специальные библиотеки для работы с числами произвольной точности, но это снижает производительность.

Возможность переполнения (или критической потери точности) является высокой, если не применять специальные техники, такие как ренормализация. Для длинных сообщений без этих техник кодирование станет невозможным или неточным.

**Задание 4.** Сравнить характеристики арифметического сжатия с вероятностными алгоритмами.

Основной принцип вероятностных алгоритмов (Хаффман, Шеннон-Фано):

* + Каждому символу исходного алфавита присваивается уникальный двоичный код.
  + Более частым символам присваиваются более короткие коды, редким – более длинные.
  + Ограничение: длина каждого кода должна быть целым числом битов.
  + Код Хаффмана оптимален в том смысле, что он дает минимальную среднюю длину кода для символа при данном распределении вероятностей, при условии целочисленности длин кодов.

Основной принцип арифметического сжатия:

* + Вся последовательность символов (сообщение) кодируется в одно число (обычно дробное, в интервале [0,1)).
  + Не присваивает отдельный код каждому символу, а сужает интервал на основе вероятности текущего символа.
  + Может достигать степени сжатия, очень близкой к теоретическому пределу (энтропии источника). Эффективно распределяет биты между символами, позволяя кодировать символ дробным числом битов в среднем.

Эффективность сжатия:

* + Арифметическое: обычно обеспечивает лучшее сжатие, так как ближе подходит к энтропийному пределу. Особенно это заметно, когда вероятности символов сильно отличаются от степеней 1/2 (например, 0.5, 0.25, 0.125).
  + Вероятностные: оптимальны при условии целочисленной длины кодов. Потери эффективности возникают, когда длина кода не является целым числом.

Сложность реализации и скорость:

* + Арифметическое: более сложно в реализации из-за необходимости работы с арифметикой высокой точности (или сложной целочисленной арифметикой с ренормализацией) и управления интервалами. Обычно медленнее, чем Хаффман.
  + Вероятностные: относительно просто реализуется (построение дерева, затем табличный поиск кода). Обычно быстрее.

**Вывод**: были рассмотрены этапы прямого (сжатие) и обратного (распаковка) преобразования. Сжатие заключается в итеративном сужении рабочего отрезка, а итоговым кодом становится число из финального, очень узкого интервала. Распаковка восстанавливает символы, анализируя, в какой подынтервал попадает закодированное число на каждом шаге. Основным преимуществом арифметического кодирования является его способность достигать степени сжатия, очень близкой к теоретическому пределу (энтропии). Это достигается за счет того, что метод эффективно присваивает символам «дробное» среднее количество бит, в отличие от методов с целочисленной длиной кода (например, Хаффмана), что особенно выгодно при вероятностях символов, не являющихся степенями 1/2.