Министерство образования и науки Российской Федерации
Федеральное государственное автономное образовательное
учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Институт естественных и точных наук
Факультет математики, механики и компьютерных технологий
Кафедра прикладной математики и программирования
Направление подготовки 01.03.02. Прикладная математика и информатика

РАБОТА ПРОВЕРЕНА
Рецензент, _____,
к.т.н.
_____/ Е.М.Сартасов
«____»_____2018г.

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой, д.ф.-м.н.,
доцент
_____/А. А. Замышляева
«____»_____2018 г.

Исследование асимметричных систем счисления как нового метода энтропийного кодирования информации

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
ЮУрГУ–01.03.02.2018.43.ПЗ ВКР

Руководитель работы, к.п.н.,
доцент
_____/А. Ю. Эвнин
«____»_____2018 г.

Автор работы,
студент группы ЕТ-412
_____/ М. С. Василенко
«____»_____2018 г.

Нормоконтролер, к.э.н., доцент
_____/Д. А. Дрозин
«____»_____2018 г.

Челябинск,
2018

АННОТАЦИЯ

Василенко М.С. Исследование асимметричных систем счисления как нового метода энтропийного кодирования информации. – Челябинск: ЮУрГУ, ЕТ-412, 83 с., 28 ил., 18 табл., библиогр. список – 23 наим., 1 прил.

В работе исследованы асимметричные системы счисления как новый метод энтропийного кодирования.

Исследована предметная область в отношении методов сжатия и кодирования информации. Реализованы наиболее эффективные методы энтропийного кодирования и новый подход, представленный асимметричными системами счисления. Проведен сравнительный анализ асимметричных систем с другими эффективными методами по времени работы и коэффициенту сжатия на основе программной реализации методов. Из анализа сделан вывод, что асимметричные системы счисления комбинируют в себе скорость сжатия метода Хаффмана и коэффициент сжатия арифметического кодирования, что подтверждает теоретические оценки.

Программная реализация осуществлена на языке C++ в среде разработки Microsoft Visual Studio 2012. В приложении приведен текст программы.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ 4
1 МЕТОДЫ СЖАТИЯ ДАННЫХ 5
1.1 История развития теории сжатия информации 5
1.2 Основные характеристики методов сжатия 6
1.3 Энтропийное кодирование 7
1.3.1 Информационная энтропия 8
1.3.1 Кодирование с помощью деревьев Шеннона Фано 9
1.3.2 Кодирование с помощью деревьев Хаффмана 10
1.3.3 Адаптивный алгоритм Хаффмана (алгоритм Виттера) 12
1.3.4 Арифметическое кодирование 14
1.3.5 Адаптивное арифметическое кодирование 16
1.4 Выводы по разделу 17
2 АСИММЕТРИЧНЫЕ СИСТЕМЫ СЧИСЛЕНИЯ 18
2.1 Введение в ANS 18
2.2 Основные положения и версии ANS 20
2.2.1 Основные положения 20
2.2.2 Uniform Asymmetric Binary Systems (uABS) 21
2.2.3 Range Asymmetric Binary / Numeral Systems (rABS и rANS) 23
2.2.4 Потоковая версия с использованием конечного автомата 24
2.2.5 Пример работы алгоритма 25
2.2.6 Необходимые условия и замечания для потоковых формул 26
2.2.7 Анализ одного шага алгоритма 27
2.3 Выводы по разделу 28
3 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ И СРАВНИТЕЛЬНЫЙ АНАЛИЗ 29
3.1 Сравнительный анализ по времени работы 29
3.2 Сравнительный анализ по коэффициенту сжатия 40
3.3 Пользовательский интерфейс для демонстрации результатов работы 45
3.4 Выводы по разделу 49
ЗАКЛЮЧЕНИЕ 50
БИБЛИОГРАФИЧЕСКИЙ СПИСОК 50
ПРИЛОЖЕНИЕ 1. Код программы 53

ВВЕДЕНИЕ

В течение последних пятидесяти лет наблюдается стремительное развитие вычислительной техники. Одновременное увеличение количества используемых компьютеров, рост пропускной способности и общего количества линий связи, появление и развитие глобальных компьютерных сетей, в настоящее время используемых сотнями миллионов пользователей, вызвали быстрое увеличение объемов передаваемой и хранимой информации. Растет частота и производительность процессоров, увеличиваются объемы памяти и ускоряется доступ к ней, однако скорость работы каналов связи, по которым необходимо быстро и надежно передавать большое количество информации, растет значительно меньшими темпами.

Одновременно с этим успешно выполняются проекты по переводу в цифровой вид содержимого огромных хранилищ информации, таких как библиотеки, галереи, музеи. Компании активно переходят к электронному документообороту и способу хранения и мониторинга данных. Алгоритмы компрессии должны выполняться на любых платформах от серверов до цифровых фотоаппаратов. Вычислительная техника непрерывно совершенствуется, поэтому алгоритмы и методы сжатия данных должны также постоянно развиваться и адаптироваться.

Таким образом, задача хранения и передачи текстовой, мультимедийной и другой информации в наиболее компактном виде очень актуальна. Последние десятилетия в этой области ведутся активные разработки. Одной из новых технологий сжатия данных без потерь являются асимметричные системы счисления, относящиеся к семейству методов энтропийного кодирования.

Целью данной работы является исследование асимметричных систем счисления и сравнение их с другими эффективными методами энтропийного кодирования информации по времени работы и коэффициенту сжатия данных.

Для достижения поставленной цели необходимо решить следующие задачи:

- выполнить анализ предметной области;
- разработать математические модели эффективных методов энтропийного кодирования, таких как алгоритм Шеннона Фано, Хаффмана и арифметическое кодирование;
- разработать математические модели асимметричных систем;
- реализовать методы на языке C++;
- реализовать пользовательский интерфейс для демонстрации результатов работы методов;
- протестировать методы на различных экспериментальных данных;
- проанализировать полученные результаты.

1 МЕТОДЫ СЖАТИЯ ДАННЫХ

Сжатие данных – это алгоритмическое преобразование данных, производимое с целью уменьшения занимаемого ими объема, которое применяется для более рационального использования устройств хранения и передачи данных.

Сжатие данных основано на устранении избыточности, содержащейся в исходных данных. В основе любого способа сжатия данных лежит модель источника данных, или модель избыточности. На основе этой модели делаются предположения о подходящих методах для этого источника данных. Модель избыточности может быть либо неизменной для всего потока данных, либо строиться в процессе сжатия или восстановления. Методы, которые позволяют в течение процесса сжатия изменять модель избыточности информации, называются адаптивными.

Все способы сжатия данных можно разделить на две категории: обратимое (сжатие без потерь) и необратимое (сжатие с потерями). При использовании сжатия без потерь закодированные данные могут быть однозначно восстановлены с точностью до бита. Такое сжатие используется, когда сжатые данные должны быть абсолютно идентичны оригиналу, например исполняемые файлы или исходный код. При использовании сжатия данных с потерями декодированные данные отличаются от исходных, но степень отличия не существенна с точки зрения их дальнейшего использования. Такое сжатие применяется для уменьшения объема цифровых, аудио- и видеоданных[1,2].

1.1 История развития теории сжатия информации

Методы сжатия данных имеют историю развития гораздо более давнюю, чем история развития вычислительной техники. История развития сжатия информации шла параллельно с историей развития кодирования и шифровки информации.

В процессе разработки азбуки Морзе в 1838 году изобретатели Альфред Вейл и Сэмюэль Морзе назначили самым часто используемым буквам в английском языке, «e» и «t», самые короткие последовательности (точка и тире соответственно). В 1928 году Ральф Хартли сделал вклад в теорию информации, введя логарифмическую меру информации, которая называется мерой Хартли[3].

Клод Шеннон был одним из первых, кто подошел к криптографии с научной точки зрения. Он первым сформулировал ее теоретические основы и ввел в рассмотрение многие важные понятия, например понятие информационной энтропии, аналогичное энтропии из термодинамики, которая является мерой неопределенности информации. В 1948 году Клод Шеннон, ознакомившись со статьями Хартли и Найквиста, опубликовал статью «Математическая теория связи»[4], в которой он изложил свои идеи, ставшие впоследствии основой

современных теорий и техник обработки, передачи и хранения информации[5]. В этой статье также был опубликован алгоритм Шеннона, который является одним из первых алгоритмов сжатия. Алгоритм использует коды переменной длины: часто встречающийся символ кодируется кодом меньшей длины, редко встречающийся кодом большей длины[6].

Впоследствии Роберт Фано независимо от Шеннона доработал этот алгоритм. Кодирование Шеннона Фано относится к вероятностным методам сжатия. Дэвид Хаффман был студентом в классе у Роберта Фано и в качестве учебной работы выбрал поиск улучшенного метода бинарного кодирования данных, в результате чего ему удалось улучшить алгоритм Шеннона Фано.

Ранние версии алгоритмов Шеннона Фано и Хаффмана использовали заранее определенные коды. Позже стали использоваться коды, созданные динамически на основе исходных данных, предназначенных для сжатия. В 1977 и 1978 годах в статьях израильских математиков Авраама Лемпеля и Якова Зива были опубликованы алгоритмы сжатия без потерь LZ77 и LZ78. Эти алгоритмы наиболее известные варианты в семействе LZ. В 1984 году Терри Велчем был опубликован алгоритм LZW в качестве улучшенной реализации алгоритмов LZ77 и LZ78. Эти алгоритмы основаны на использовании динамически создаваемого словаря, или «скользящего окна».

Алгоритмы семейства LZ получили большую популярность и вызвали волну интереса к проблеме сжатия данных. Большие корпорации использовали алгоритмы сжатия для хранения все увеличивавшихся массивов данных, но настоящий скачок произошел с возникновением интернета в конце 1980-х годов. Пропускная способность каналов была чрезвычайно узкой, и для сжатия данных, передаваемых по сети, были придуманы форматы ZIP, GIF и PNG. В 1993 году Евгений Рошал придумал свой формат и алгоритм RAR, а в 1999 году появилась программа 7-Zip, созданная Игорем Павловым[7,8].

В области сжатия данных ведется активная работа по улучшению уже существующих и созданию новых алгоритмов сжатия. В 2014 году Ярослав (Ярек) Дуда (Jarek Duda, Jagiellonian University, Poland) представил асимметричные системы счисления (Asymmetric Numeral Systems, далее ANS) как семейство методов энтропийного кодирования, комбинирующие в себе степень сжатия арифметического кодирования со стоимостью обработки кодирования методом Хаффмана.

ANS используются в Facebook Zstandart compressor, Apple LZFSE compressor, Google Draco 3D compressor и многих других современных форматах сжатия данных. Далее будут исследованы ANS как новый метод энтропийного кодирования и произведено сравнение с разработанными ранее алгоритмами[9].

1.2 Основные характеристики методов сжатия

Основной характеристикой алгоритмов сжатия является коэффициент сжатия. Он определяется как отношение объема исходных несжатых данных к объему

сжатых данных, то есть —, где коэффициент сжатия, объем исходных данных, объем сжатых данных. Отсюда следует, что чем выше коэффициент сжатия, тем алгоритм эффективнее.

Если то алгоритм не производит сжатия. Если же , то алгоритм порождает сообщение большего размера, чем исходное. Такое возможно, так как число различных сообщений длиной бит равно , а число соответственно различных сообщений длиной меньше или равной будет не больше отсюда невозможно однозначно сопоставить все возможные исходные сообщения сжатым по принципу Дирихле. Поэтому невозможно получить алгоритм сжатия без потерь, который при любых данных имел бы коэффициент сжатия больший или равный единице. Коэффициент сжатия может быть постоянным для всех данных, как например в алгоритме ADPCM[10], а может варьироваться для каждого конкретного сообщения. В общем случае постоянный коэффициент сжатия способны обеспечить только методы сжатия данных с потерями.

Важным критерием оценки методов сжатия данных является допустимость потерь. Применение алгоритмов сжатия без потерь возможно для данных любого типа. Для некоторых типов данных потери недопустимы, например для программ и их исходных текстов, жизненно важных данных, таких как измерения медицинской аппаратуры или контрольных приборов летательных аппаратов. Для других типов данных применение сжатия с потерями должно быть обосновано.

Алгоритмы сжатия данных могут также требовать различного количества ресурсов, на которых они реализованы, а именно оперативной памяти, занимаемой промежуточными данными, постоянной памяти, занимаемой кодом программы, и процессорного времени. Чем эффективнее и универсальнее алгоритм, тем больше требований к вычислительным ресурсам он предъявляет. Однако в отдельных случаях простые и компактные алгоритмы могут работать не хуже сложных и универсальных[1,2].

1.3 Энтропийное кодирование

В теории информации энтропийное кодирование представляет собой схему сжатия данных без потерь, которая не зависит от конкретных характеристик среды. Энтропийное кодирование относится к вероятностным методам сжатия. Предполагается, что до кодирования отдельные элементы последовательности имеют различную вероятность появления. После кодирования в результирующей последовательности вероятности появления отдельных символов практически одинаковы.

Двумя из наиболее распространенных методов энтропийного кодирования являются кодирование Хаффмана и арифметическое кодирование.

Рассмотрим более подробно понятие информационной энтропии и некоторые методы энтропийного кодирования[11].

1.3.1 Информационная энтропия

Возникновение математической теории информации стало возможно после того, как стало ясно, что количество информации можно задать числом. Чтобы измерить количество информации, необходимо закодировать ее в виде двоичной последовательности наиболее рациональным способом. Количество знаков полученного кодового слова может служить мерой количества информации. Единица информации это количество информации, которое можно записать при помощи 0 или 1. Отсюда и термин *бит* (от англ. *bi*nary digi*t* двоичная цифра).

В 1928 году Ральф Хартли предложил научный подход к оценке сообщений. Пусть имеется некоторый алфавит А, из букв которого составляется сообщение, и Тогда количество возможных вариантов различный сообщений равно , где n количество букв в сообщении. Формула Хартли определяется как , где количество информации в битах.

Клод Шеннон попытался обобщить эту формулу на случай, если события не равновероятны. Он предположил, что прирост информации равен утраченной неопределенности. Так как энтропия в термодинамике может интерпретироваться как мера неопределенности некоторой системы, то было введено понятие информационной энтропии.

Информационной энтропией называют количество информации, приходящейся на одно элементарное сообщение источника, вырабатывающего статистически независимые сообщения. Клод Шеннон задал требования к ее измерению.

Во-первых, мера должна быть непрерывной, то есть изменение значения величины вероятности на малую величину должно вызывать малое результирующее изменение функции.

Во-вторых, когда все варианты равновероятны, увеличение количества вариантов должно всегда увеличивать значение функции.

В-третьих, должна быть возможность сделать выбор в два шага, в которых значение функции конечного результата должно являться суммой функций промежуточных результатов. Шеннон показал, что функция энтропии определена и непрерывна для всех и имеет единственно возможный вид:

(1.1)

где K положительная константа, необходимая для выбора единиц измерения энтропии, а вероятности, с которыми некоторая случайная величина над которой проводятся наблюдения, принимает возможные различные значения .

Отсюда энтропия случайной величины , или количество информации, содержащейся в наблюдении случайной величины не зависит от различных

значений , которые она может принимать, а определяется только вероятностями , с которыми принимает эти значения.

Согласно теореме Шеннона, существует предел сжатия без потерь, зависящий от энтропии источника. Рассмотрим любую информационную строку длиной как последовательность независимых, одинаково распределенных дискретных случайных величин (д. с. в.) или как выборку длиной значений одной д.с.в. . Доказано, что среднее количество бит, приходящееся на одно кодируемое значение д.с.в, не меньше энтропии этой величины. Поэтому чем более предсказуемы получаемые данные, тем лучше их можно сжать[12].

1.3.1 Кодирование с помощью деревьев Шеннона Фано

Американские математики Клод Шеннон и Роберт Фано независимо друг от друга изобрели в 1948-1949 гг. следующий способ кодирования. В основе алгоритма лежит идея представления более частых символов с помощью более коротких кодов. При этом коды, полученные с помощью алгоритма Шеннона Фано, обладают свойством префиксности, т.е. ни один код не является началом никакого другого кода. Свойство префиксности гарантирует, что кодирование будет взаимно-однозначным.

Значения д.с.в. (символы алфавита) выписывают по убыванию вероятностей появления их во входном потоке данных, а затем делят на две части так, чтобы суммарные вероятности в каждой части были как можно более близки друг к другу. Это деление не всегда можно осуществить однозначно.

Символам из одной части приписывается 0, а из другой 1. По описанному выше принципу каждая из полученных частей снова разбивается на две части, и это разбиение определяет второй символ кодового слова. Процедура продолжается до тех пор, пока все множество символов не будет разбито на одноэлементные множества. В результате каждому из символов алфавита ставится в соответствие кодовое слово из нулей и единиц.

Алгоритм Шеннона Фано удобно пояснить с помощью бинарного дерева. Из его корня выходят два ребра, помеченные соответственно 0 и 1. Эти ребра соответствуют разбиению кодируемого алфавита на две приблизительно равновероятные группы. Ребра, выходящие из вершин следующего яруса, отвечают разбиению получившихся групп снова на примерно равновероятные подгруппы и т.д. Каждый лист построенного дерева соответствует некоторому кодовому слову. Чтобы указать это слово, нужно пройти весь путь от корня к листу, выписывая последовательно символы ребер на этом пути. Время построения дерева кодирования пропорционально размеру алфавита и зависит от распределения вероятностей, однако для файлов, чей размер сильно больше размера алфавита, время построения дерева несущественно.

Несмотря на сравнительную простоту, алгоритм Шеннона Фано не лишен недостатков, самым существенным их которых является тот факт, что хотя разбиение на каждом шаге и является оптимальным, алгоритм не гарантирует оптимального результата в целом.

Пример. Для случайной величины с заданным распределением вероятностей, приведенным на «таблице 1.1»:

Таблица 1.1

Распределение вероятностей символов

	A	B	C	D	E	F
	0,36	0,19	0,17	0,12	0,09	0,07

Получаем следующее дерево Шеннона–Фано, представленное на «рисунке 1.1»:

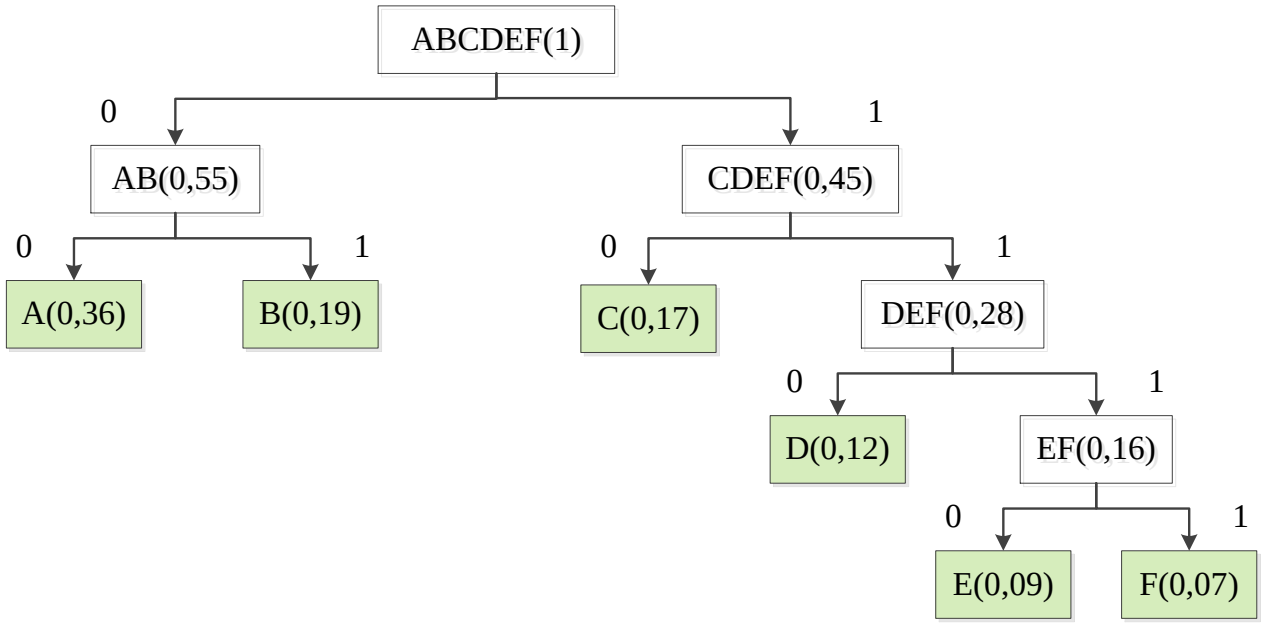

Рисунок 1.1. Дерево Шеннона Фано

Кодировки значений таковы: 00, 01, 10, 110, 1110, 1111. Средняя длина кода равна

бит/сим.

Без использования алгоритма каждый символ кодировался бы 3 битами ( ).

1.3.2 Кодирование с помощью деревьев Хаффмана

Алгоритм был разработан в 1952 году Дэвидом Хаффманом и обладает, как и алгоритм Шеннона Фано, свойством префиксности, а, кроме того, доказанной минимальной избыточностью. Именно этим обусловлено его широкое распространение.

Код строится при помощи бинарного дерева, листья которого соответствуют значениям д.с.в. Весом листа будем называть вероятность соответствующего значения д.с.в. Два листа с наименьшими весами создают родительский узел, вес которого равен сумме весов этих листьев. Далее процесс повторяется рекурсивно.

Двум ребрам, соединяющим родительский узел с двумя потомками, приписываются значения 0 и 1. Как и в алгоритме Шеннона Фано, код каждого значения д.с.в. получается при следовании по маршруту от корня дерева к листу.

Для случайной величины из предыдущего примера получаем следующее дерево, представленное на «рисунке 1.2»:

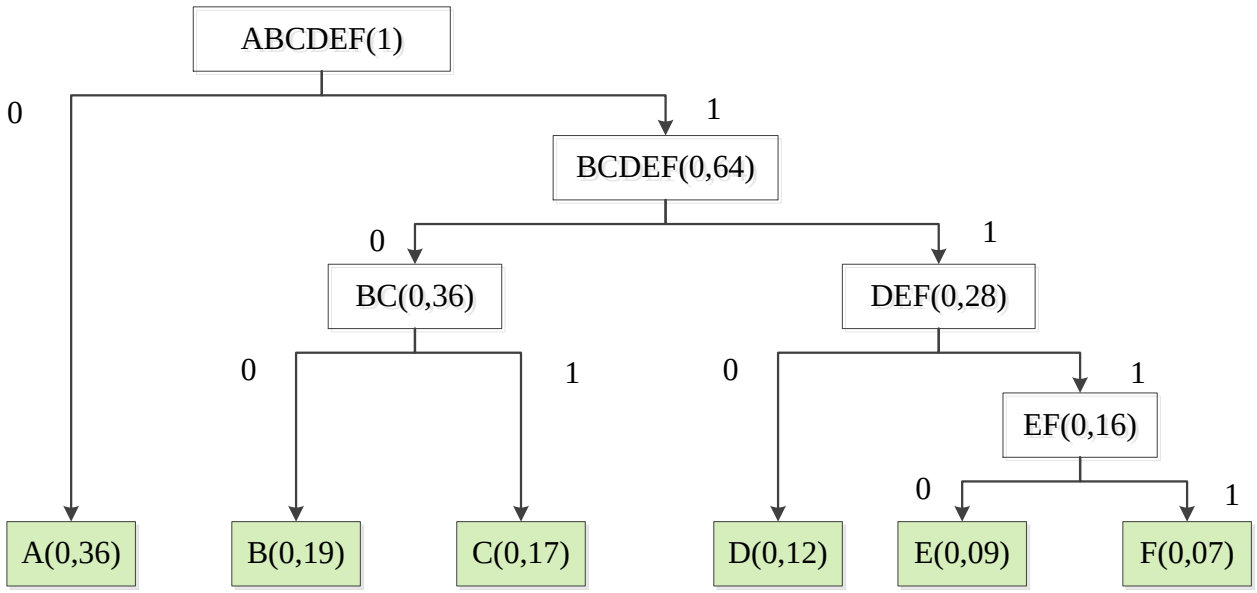

Рисунок 1.2. Дерево Хаффмана

Кодировки значений таковы: 0, 100, 101, 110, 1110, 1111. Средняя длина кода равна
бит/сим.

Она совпала со средней длиной кода при использовании метода Шеннона Фано. Доказано, что по этому показателю алгоритм Хаффмана никогда не уступает алгоритму Шеннона Фано.

Среди недостатков алгоритма Хаффмана можно отметить, что использование для хранения частот символов вещественных переменных сопряжено с потерей точности, поэтому на практике часто используют целочисленные переменные. Однако так как вес родительских узлов постоянно растет, рано или поздно возникает переполнение типа данных. Несмотря на кажущуюся простоту алгоритма, его корректная реализация до сих пор может вызывать некоторые затруднения, особенно при использовании больших алфавитов.

Стоит также отметить, что для алгоритмов Шеннона Фано и Хаффмана каждый раз вместе с самим сообщением необходимо передавать и таблицу кодов. Следовательно, длина сжатого сообщения увеличивается на длину таблицы, которая должна посылаться впереди данных. Кроме того, необходимость наличия полной частотной статистики перед началом кодирования требует двух проходов по сообщению: одного для построения таблицы частот и дерева, другого для собственно кодирования.

Наиболее простая реализация алгоритма использует очередь с приоритетами, для которой узел с наименьшей вероятностью имеет наибольший приоритет. Так

как эффективные структуры данных, реализующие очередь с приоритетами, требуют времени для вставки элемента, а в дереве с листьями вершин, то алгоритм работает за время , где количество символов.

1.3.3 Адаптивный алгоритм Хаффмана (алгоритм Виттера)

Адаптивное сжатие позволяет не передавать таблицы частот вместе с сообщением и ограничиться одним проходом по сообщению как при кодировании, так и при декодировании.

Джеффри Виттер предложил следующую технику адаптивного кодирования на основе алгоритма Хаффмана. В начале работы алгоритма дерево кодирования содержит только один специальный символ ESC, имеющий вес 0. Он необходим для занесения в дерево новых символов. После него код символа передается непосредственно. Будем работать с упорядоченным деревом. В нем узлы идут по неубыванию веса по ярусам снизу вверх и слева направо в каждом ярусе. Если при изменении веса существующего узла упорядоченность дерева нарушается, этот узел меняют местами с последним следующим за ним узлом меньшего веса. После перемены мест узлов нужно пересчитать веса всех их предков.

Закодируем адаптивным алгоритмом Хаффмана с упорядоченным деревом (алгоритмом Виттера) сообщение ABBAC, считая, что маркер конца сообщения символ E.

На 1-м шаге добавляется новый символ A.

На 2-м шаге добавляется новый символ B.

На 3-м шаге изменяется вез узла B, дерево перестраивается (меняются местами узлы 2/B и 1/A).

На 4-м шаге изменяется вез узла A.

На 5-м шаге добавляется новый символ C, после чего меняются местами потомки корня дерева.

На 6-м шаге добавляется новый символ E.

Процесс построения дерева представлен на «рисунке 1.3»:

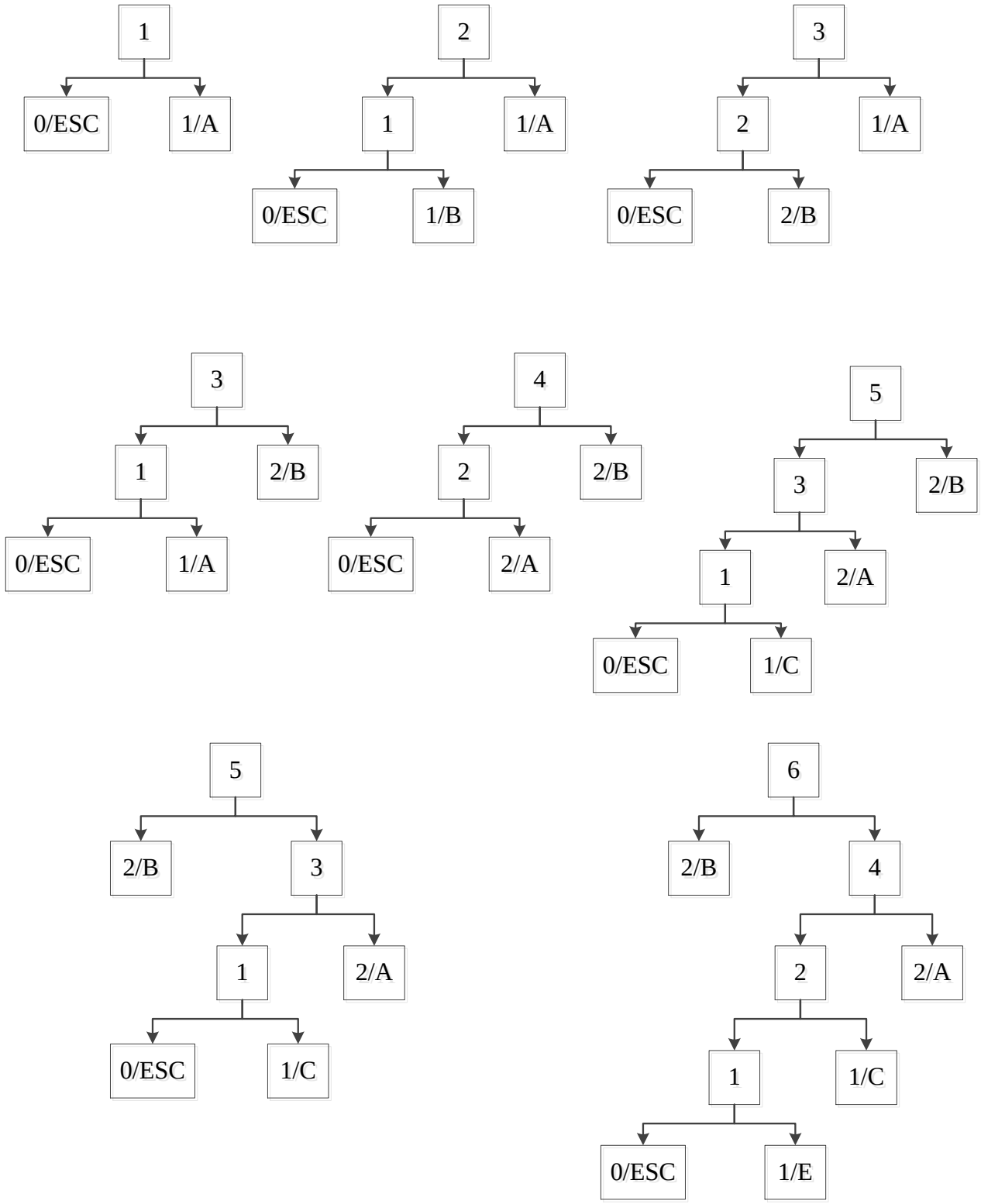

Рисунок 1.3. Процесс построения адаптивного дерева Хаффмана

Если считать, что для кодировки каждого из четырех символов используется по два бита, общая длина кода равна В среднем на один символ сообщения приходится 3 бита.

При декодировании также строится кодовое дерево.

Неадаптивный алгоритм Хаффмана требует прочтения исходного сообщения два раза, при этом построение дерева выполняется по уже подсчитанной таблице вероятностей. Время построения дерева где – размер алфавита, данные же обрабатываются за где – размер сообщения, – средний размер кода символа. На больших реальных данных размер алфавита небольшой, а значит и средняя длина кода, поэтому процесс построения дерева занимает незначительное время по сравнению с обработкой большого входного потока.

Адаптивный алгоритм не требует дополнительного прохода по сообщению, однако при прочтении очередного символа его необходимо добавить в дерево. На больших реальных данных нам придется почти каждый раз перестраивать дерево, что является довольно дорогостоящей операцией и в худшем случае будет выполнено за размер дерева. Отсюда асимптотика будет , где – размер сообщения, – размер дерева, который в момент встречи каждого символа алфавита хотя бы раз будет равняться 2 . Отсюда на больших реальных данных асимптотика адаптивного алгоритма не лучше асимптотики неадаптивного[13-19].

1.3.4 Арифметическое кодирование

Арифметическое кодирование – это форма энтропийного кодирования, используемая при сжатии данных без потерь. Разработано в 1979 году фирмой IBM. Является одним из наиболее эффективных способов сжатия информации. В отличие от алгоритма Хаффмана арифметическое кодирование позволяет кодировать сообщения с энтропией меньше 1 бита на символ. Так как большинство алгоритмов арифметического кодирования защищены патентами, далее рассмотрим только основные идеи.

Пусть значения д.с.в. принимают значения Каждому значению сопоставляем отрезок длиной причем объединение этих отрезков есть отрезок [0;1]. Для каждого из отрезков находим внутреннюю точку вида — где , причем принимает наименьшее возможное значение.

Отметим, что по значению показателя числитель определяется однозначно. Действительно, во-первых, число нечетно (иначе можно было бы уменьшить значение Во-вторых, если для какого-то нечетного числа числа — — попадали в данный отрезок, то ему принадлежала бы и дробь — с четным числителем, которую можно было бы сократить на 2, уменьшив тем самым знаменатель.

Число — и будет кодом для , если в его двоичной записи оставить только дробную часть.

Пусть у нас имеется отрезок для сообщений длиной . Тогда для разбиения отрезка, отвечающего сообщению длиной , предыдущий отрезок делим на частей так же, как делается самое первое разбиение.

Пример. Пусть – – Соответствующие отрезки – –

Для сообщений 00, 01, 10, 11 получаем отрезки – – – – – – .

Для сообщений 000, 001, 010, 011, 100, 101, 110, 111 получаем отрезки — — – – — — – – — — – – —

Теперь в каждом из отрезков выбираем двоичную дробь с наименьшим возможным знаменателем:

– — – — –

– – — – — –

– – — – — –

— – — — —

Сравним коды для сообщений из трех символов, которые получаются в результате применения метода Хаффмана и арифметического кодирования, и представим в «таблице 1.2»:

Таблица 1.2

Сравнение кодов

сообщение	000	001	010	011	100	101	110	111
27	8	4	4	2	4	2	2	1
Код Хаффмена	11	101	100	011	001	010	0001	0000
Арифм.код	01	011	1	101	11	111	1111	11111

Среднее количество бит на единицу сообщения для блочного метода Хаффмана — , а для арифметического кодирования — .

Алгоритм декодирования. Символы сообщения определяются по одному, слева направо. В таблице для кодирования значений д.с.в. определяется интервал, содержащий текущий код. По нему определяется один символ исходного сообщения. В случае если этот символ – маркер конца сообщения, либо известно, что это последний символ, то алгоритм заканчивает свою работу. Из текущего кода вычитается нижняя граница содержащего его интервала, полученная разность делится на длину интервала. Полученное число новое текущее значение кода.

Эффективность арифметического кодирования растет с ростом длины сжимаемого сообщения, однако минусом такого способа кодирования является чрезвычайно высокая сложность процессов кодирования и декодирования. На каждом шаге кодирования очередного символа необходимо провести

перенормировку отрезков, итоговое время работы будет равно , где N – размер алфавита, M – размер файла. При этом в процессе кодирования необходимо работать с длинными числами, чтобы не терять точность вычислений, представляя границы отрезков в виде несократимых дробей, что увеличивает время работы.

1.3.5 Адаптивное арифметическое кодирование

Как и для алгоритма Хаффмана, рассмотрим адаптивный метод арифметического кодирования. Каждому символу сопоставляется его вес. Вначале для каждого символа он равен 1. Вероятность появления каждого символа вычисляется как отношение его веса к сумме весов всех символов. После получения очередного символа и постройки интервала для него вес этого символа увеличивается на 1. Для того чтобы обеспечить остановку алгоритма распаковки, в начале сжимаемого сообщения нужно поставить его длину или ввести дополнительный символ-маркер конца сообщения. Если известен формат сжимаемого файла, то вместо начального равномерного распределения весов можно выбрать распределение с учетом этих знаний.

Пример. Пусть алфавит состоит из букв A, B и C. Маркером конца сообщения будем считать символ E. Сжимаемое сообщение – «ABBAC». Процесс представлен на «таблице 1.3»:

Таблица 1.3

Процесс сжатия адаптивным арифметическим кодированием

Веса				Общий вес	Кодир. символ	Текущая длина	Получ. промежуток
A	B	C	E				
1	1	1	1	4	A	1	
2	1	1	1	5	B		
2	2	1	1	6	B		
2	3	1	1	7	A		
3	3	1	1	8	C		—— ——
3	3	2	1	9	E		——— ———

В получившийся промежуток попадает число —— Поэтому исходное сообщение представляется двоичным числом 0,00011110111011, а сообщение кодируется 14 битами, – бит/сим.

Декодирование происходит следующим образом. На каждом шаге определяется промежуток, содержащий данный код, по нему однозначно находится кодируемый символ. Затем из текущего кода вычитаем нижнюю границу содержащего его промежутка, а полученную разность делим на длину

этого промежутка. Полученное число – новое текущее значение кода. Получение маркера конца означает конец работы.

В качестве примера распакуем код 00011110111011, зная алфавит сообщения A, B, C, E, где E – маркер конца. Начальное число – код 0,00011110111011 = 1979/16384. Пример представлен на «таблице 1.4»:

Таблица 1.4

Процесс декодирования адаптивным арифметическим кодированием

Веса				Числовой код	Промежуток	Длина пром.	Декод. символ
A	B	C	E				
1	1	1	1		[0;1/4]	1/4	A
2	1	1	1		[2/5; 3/5]	1/5	B
2	2	1	1		[1/3; 2/3]	1/3	B
2	3	1	1		[0; 2/7]	2/7	A
3	3	1	1		[6/8; 7/8]	1/8	C
3	3	2	1		[8/9; 1]	1/9	E

При кодировании и декодировании адаптивным арифметическим кодированием нам не нужно дважды просматривать исходный файл, однако на каждом шаге происходит процесс перенормировки отрезков, что является более дорогостоящей операцией, чем дополнительный просмотр файла[20].

1.4 Выводы по разделу

Методы энтропийного кодирования являются одними из самых эффективных методов сжатия данных. Данное свойство позволяет применять их во многих алгоритмах и программах для сжатия данных.

В данной главе были рассмотрены основные методы энтропийного кодирования, такие как алгоритм Шеннона Фано, Хаффмана, адаптивный метод Хаффмана, арифметическое кодирование и адаптивное арифметическое кодирование.

Далее будут рассмотрены асимметричные системы счисления, сочетающие в себе степень сжатия арифметического кодирования со стоимостью обработки кодирования методом Хаффмана.

2 АСИММЕТРИЧНЫЕ СИСТЕМЫ СЧИСЛЕНИЯ

Современные методы сжатия данных основаны на двух подходах энтропийного кодирования: кодировании Хаффмана (Huffman coding, HC) и арифметическом кодировании (Arythmetic coding, AC). Первый метод гораздо быстрее, но хорошо аппроксимирует вероятности, являющиеся степенями двойки, и обычно приводит к относительно небольшой степени сжатия. Второй метод использует почти точные вероятности, легко достигая теоретического предела сжатия (энтропии Шеннона), но требует большой вычислительной стоимости.

Асимметричные системы счисления (Asymmetric numeral systems, ANS) являются новым подходом точного энтропийного кодирования, который позволяет завершить разногласия между скоростью и сжатием. Теоретические исследования показывают, что они позволяют кодировать и декодировать данные так же быстро, как HC со степенью сжатия такой же, как у AC. Это преимущество связано с тем, что ANS проще, чем AC: используют единственное натуральное число в качестве состояния вместо двух, отображающих отрезок. Помимо упрощения перенормировки, это позволяет внести поведение для данного распределения вероятностей в относительно небольшую таблицу, описывающую автомат энтропийного кодирования. Такая таблица для алфавита из 256 символов занимает в памяти несколько килобайт.

Экспериментальные результаты демонстрируют эффективность предлагаемого энтропийного кодировщика[21].

2.1 Введение в ANS

В стандартных системах счисления имеющаяся информация хранится в натуральном числе , имеющем знаков. Мы можем добавить информацию из символа двумя различными способами: в старший разряд или в младший разряд . Первый подход означает, что для добавления новой цифры необходимо оперировать большими диапазонами чисел, а также запоминать наиболее значащую позицию .

Напротив, при добавлении информации в младший разряд текущим состоянием будет являться единственное натуральное число. Этот подход и используется в ANS. В то время как в стандартных системах счисления на очередном шаге добавления нового разряда число просто меняет свою четность в зависимости от , подход, использующийся в ANS, переопределяет такое разделение. Таким образом, «четные» и «нечетные» числа все еще «равномерно распределены», но с различными плотностями, соответствующими распределению вероятностей символов, которые мы хотели бы закодировать.

Давайте рассмотрим оба подхода с точки зрения теории информации. Для AC информация о том, что мы находимся в данной половине отрезка, составляет 1

бит, поэтому текущее сообщение занимает логарифм по основанию 2 от размера отрезка, деленного на размер подотрезка бит информации. Если на текущем шаге кодируется символ с вероятностью , то размер отрезка делится на , увеличивая количество информации на − бит, как и ожидалось.

С точки зрения ANS, считая, что состояние содержит бит информации, при добавлении символа с вероятностью количество информации должно увеличиться до − − бит. Поэтому нам необходимо обеспечить переход вида . Интуитивное правило состоит в том, что становится -м элементом множества для символа , если это множество равномерно распределено с плотностью . Способ сделать бинарные системы асимметричными представлен на «рисунке 2.1»:

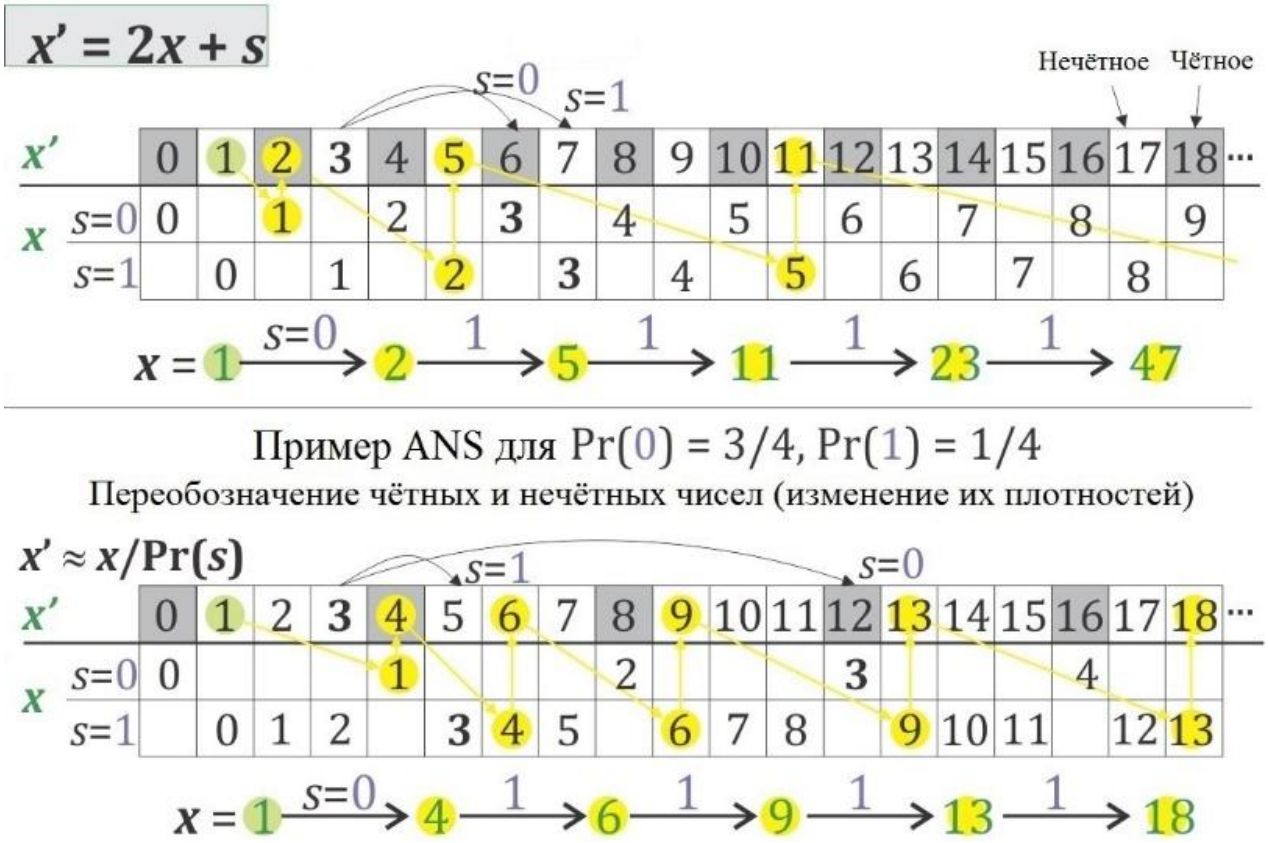

Рисунок 2.1. Способ сделать бинарные системы счисления асимметричными.

Рассмотрим «рисунок 2.1» более подробно. Имея некоторую информацию, хранящуюся в натуральном числе , чтобы присоединить информацию о символе (0 или 1), мы можем добавить его в старший разряд , или в младший разряд . Первый подход AC может сделать асимметричным, изменяя пропорции диапазонов. Второй подход асимметричным может сделать ANS, переопределив четные и нечетные числа так, что они все еще равномерно распределены, но с различной плотностью. Теперь будет -м элементом подмножества для .

Существует несколько различных вариантов для приложения подхода ANS или его бинарного случая, асимметричных двоичных систем счисления (Asymmetric Binary Systems, ABS). Кратко опишем эти варианты.

Основное преимущество подхода ANS состоит в том, что он реализует наиболее простую и эффективную перенормировку состояний. По сравнению с AC, мы можем быстро уменьшить количество бит состояния или хранить их в таблице – будем просто передавать целый блок бит за один шаг, что значительно быстрее посимвольного подхода.

Равномерные асимметричные двоичные системы (Uniform Asymmetric Binary Systems, uABS) используют прямую арифметическую формулу для равномерно распределенных символов.

Диапазонные асимметричные двоичные системы (Range Asymmetric Binary Systems, rABS) используются для распределения символов в диапазонах и предоставляют все еще прямую формулу, немного менее точную, но более простую в вычислениях.

Дипазонные асимметричные системы счисления (Range Asymmetric Numeral Systems, rANS) могут быть рассмотрены как прямая альтернатива AC, но с небольшим преимуществом: вместо двух умножений за шаг требуется только одно.

Потоковая версия rANS обладает большим преимуществом по отношению к AC, так как избегает работы с длинными числами, сохраняя состояние в процессе кодирования и декодирования в определенном диапазоне, и передает уже готовую часть кода в выходной поток.

Существует большая свобода выбора соответствующего подхода для заданных параметров. Рассмотрим эти подходы более подробно.

2.2 Основные положения и версии ANS

Ниже будут рассмотрены основные положения для кодирования информации с помощью ANS и найдены аналитические формулы для различных модификаций ANS.

2.2.1 Основные положения

Пусть задан алфавит и распределение вероятностей , Состояние будет содержать информацию о уже обработанной части символьной последовательности.

Нам необходимо найти кодирующую и декодирующую функции. Первая из них принимает состояние и символ и преобразует их в новое состояние . Если рассматривать как возможность выбрать число из интервала , то будет содержать бит информации. Символ с вероятностью появления содержит — бит информации, поэтому должно содержать приблизительно — — бит информации;

должно быть примерно равно , позволяя соответственно выбрать значение из большего интервала . В итоге мы должны будем получить функции, определяющие следующие единичные шаги:

(2.1)

Для стандартной двоичной системы счисления мы имеем:

– (2.2)

Можно заметить, что может менять четность состояния.

В общем случае мы хотим переопределить подмножества, относящиеся к различным : как четные и нечетные числа, они все еще должны равномерно покрывать натуральные числа, но с различными плотностями: . Мы можем определить это разделение натуральных чисел как символьное распределение:

(2.3)

В то время как в стандартной двоичной системе счисления это -е вхождение четного или нечетного числа, в нашем случае будет -ым вхождением множества символа . Поэтому функция декодирования будет выглядеть как:

(2.4)

и обратная функция. Также мы имеем, что .

В итоге мы можем представить, что это стек символов, – операция добавления элемента на вершину стека, – операция извлечения элемента с вершины стека. Алгоритм кодирования примет вид: начинаем с и затем последовательно используем для кодирования символов. Такой алгоритм может привести к появлению больших натуральных чисел, из которых мы можем извлечь все символы в обратном порядке, используя . Чтобы избежать неудобных операций над большими числами, существует потоковая версия алгоритма, в которой накопленные биты извлекаются, чтобы оставался в фиксированном интервале .

2.2.2 Uniform Asymmetric Binary Systems (uABS)

Выведем некоторые явные формулы для бинарного случая и почти равномерного распределения вероятностей символов .

Обозначим Чтобы получить можем для примера выбрать :

(2.5)

Теперь , если есть прыжок в следующую позицию:

(2.6)

Таким образом, была найдена декодирующая функция

(2.7)

Найдем теперь соответствующую кодирующую функцию: для заданных и мы хотим найти . Обозначим , отсюда

(2.8)

(2.9)

(2.10)

В итоге функция кодирования:

(2.11)

Для это стандартная двоичная система счисления.

Рассмотрим пример кодирующего процесса вставками последующих символов для :

Таблица 2.1

Пример работы алгоритма uABS

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
		0	1		2	3		4	5	6		7	8		9	10		11	12	13	
	0			1			2				3			4							6

2.2.3 Range Asymmetric Binary / Numeral Systems (rABS и rANS)

Для данного подхода наиболее важным является то, что он позволяет оперировать с большими алфавитами, что делает его прямой альтернативой арифметическому кодированию, так как будет требоваться единственное умножение в расчете на символ по сравнению с двумя, используемыми в арифметическом кодировании. Этот подход можно рассматривать как то, что мы берем стандартную систему счисления и сливаем в один некоторые последовательные символы.

Мы начнем рассмотрение с двоичной системы (rABS), а затем опишем общие формулы. Рассмотрим процесс на примере:

Посмотрим на -вероятностное распределение. Мы бы хотели взять стандартную систему счисления по основанию 4 и, используя цифры , объединить 3 из них, например 1, 2 и 3. Поэтому пока символьное распределение для стандартной троичной системы будет выглядеть как с циклом (0123), мы сделаем его асимметричным и рассмотрим распределение с циклом (0111):

(2.12)

Поведение для в точности такое, как в троичной системе счисления. Напротив, при мы перейдем к -ей из (0111) четверке:

(2.13)

где «+1» это сдвиг через единственное вхождение «0». Декодирование выглядит так:

(2.14)

в другом случае

Аналогично мы можем определить кодирующую/декодирующую функции для различных дробей, а для больших алфавитов мы рассмотрим rANS. Будем считать, что распределение вероятностей это дробь вида где

Теперь давайте по аналогии представим, что мы начинаем с системы счисления с основанием и объединяем вместе соответствующие количества

последовательных цифр вместе. Таким образом, символьное распределение представляет собой цикл (00..011..1…n 1), и количество вхождений символа равно . Давайте определим как символ на позиции в этом цикле:

(2.15)

и начальная позиция символа в этом цикле. Эти должны быть внесены в таблицу для кодировщика. Внесение в таблицы сократит количество их использования до одного за шаг.

Теперь шаги кодирования и декодирования выглядят так:

(2.16)

Если мы выбираем как степень двойки, умножение и деление может быть сделано битовыми сдвигами, при помощи битового "и" с маской, декодирование требует единственного умножения за шаг. В то же время при арифметическом кодировании необходимо два умножения за шаг, поэтому rANS работают быстрее.

2.2.4 Потоковая версия с использованием конечного автомата

Чтобы предотвратить бесконечный рост состояния ANS, мы будем следить за тем, чтобы оставался в некотором выбранном диапазоне Таким образом можно считать буфером, содержащим от до бит информации. Поэтому в то время как мы работаем с символами, содержащими нецелое число бит, поскольку они накапливаются внутри буфера, мы можем извлекать полные биты информации и оставаться в рабочем диапазоне. Эти извлеченные биты передаются в выходной поток и используются в процессе декодирования.

Чтобы сделать наши соображения более общими, мы будем извлекать цифры в системе счисления по основанию . Обычно мы будем использовать но иногда использование большего может стать более удобным, для примера позволяет извлечь бит за раз. Извлечение наименее значащей цифры в системе счисления с основанием означает, что и переходит в выходной поток.

Заметим, что для того чтобы сохранять интервал в виде:

(2.17)

для любого мы будем иметь в точности один из трех случаев:

- или
- тогда или
- тогда

Мы будем называть этот вид интервала как -уникальный, так как вставляя или удаляя некоторые из наименее значащих цифр , мы всегда в итоге придем к интервалу уникальным способом. Давайте также определим:

(2.18)

Теперь мы можем определить шаги для потокового декодирования: использовать функцию и затем забирать наименее значащие цифры из потока символов до тех пор, пока мы не вернемся в , и для потокового кодирования: отправлять наименее значащие цифры в поток до тех пор, пока мы не сможем использовать функцию .

Приведенные выше функции взаимно противоположны друг другу, только если тоже -уникальны: в таком случае существуют , такие что:

(2.19)

Мы должны быть уверены в том, что это необходимое условие выполнено, что не является автоматически верным, как мы увидим в примере. Далее будут также рассмотрены более компактная форма этого условия и обеспечение его работы для больших алфавитов.

2.2.5 Пример работы алгоритма

Возьмем для примера uABS для и трансформируем его в потоковую версию для (мы переносим одиночные наименее значащие биты). Выбирая мы имеем . Мы можем увидеть, что – диапазоны, из которых мы приходим к после кодирования соответственно 0 или 1. Эти интервалы не являются -уникальными, поэтому для примера, если мы захотим закодировать из , при передаче наименее

значащих битов мы сначала уменьшим его до а затем до не попадая в диапазон .

Однако, если мы выберем , мы будем иметь и , которые являются -уникальными. Рассмотрим процесс кодирования для : вначале мы передаем некоторые из наименее значащих битов в поток, до тех пор пока мы не уменьшим до диапазона . Затем мы используем ABS формулу:

Таблица 2.2

Пример работы алгоритма

	9	10	11	12	13	14	15	16	17
Передача бит для уменьшение	-	-	-	0	1	0	1	0	1
	9	10	11	6	6	7	7	8	8
	14	15	17	9	9	11	11	12	12
Передача бит для уменьшение	1	0	1	0, 0	1, 0	0, 1	1, 1	0, 0	1, 0
	4	5	5	3	3	3	3	4	4
	13	16	16	10	10	10	10	13	13

Ниже приведен пример изменения состояния и последовательности бит:

Эта найденная кодирующая таблица полностью определяет процесс кодирования, и аналогично мы можем сгенерировать таблицу для процесса декодирования.

Декодер в начале использует , чтобы получить символ и уменьшить , затем добавляет наименее значащие биты из потока.

2.2.6 Необходимые условия и замечания для потоковых формул

Для однозначного декодирования нам необходимо, чтобы все были -уникальными, что не всегда является верным, как мы увидели в примере выше. Проверим, когда же uABS, rABS и rANS удовлетворяют этому условию.

В случае uABS, обозначив , нам нужно проверить, что . Мы имеем поэтому . Это означает, что из выполнения одного условия следует выполнение второго.

Давайте проверим, когда У нас есть , поэтому критерий выглядит таким образом:

Критерий: Потоковая версия uABS может быть использована в случае если:

(2.20)

Основная ситуация, в которой этот критерий будет выполнен – это когда , что означает, что определено с точностью .

Для rABS и rANS для простоты будем считать, что делит : (могут быть также и другие варианты). В этом случае, – первое вхождение символа в , – последнее, как и требуется.

Критерий: Если делит , rABS и rANS могут быть использованы в потоковой версии.

2.2.7 Анализ одного шага алгоритма

Давайте теперь внимательно изучим единственный шаг потокового кодирования и декодирования. Заметим, что количество цифр для передачи, необходимое чтобы попасть из в будет равно , поэтому новое состояние (новый символ, символьная последовательность) в потоковом кодировании будет выглядеть как:

(2.21)

При этом – это информация, которая теряется из в процессе передачи цифр – это значение отправляется в поток в системе счисления с основанием . Исходный алгоритм дает эти цифры в обратном порядке – от менее значимых к более значимым.

Заметим, что для данного символа , количество цифр для передачи может принимать одно из двух значений: или , где . Наименьший , требующий передачи цифр, это . Поэтому в итоге количество бит для передачи в процессе кодирования – это для и для .

При декодировании потока текущее состояние определяет в настоящее время символ . Однако, в примере из Таблицы 6, состояние является исключением: оно декодируется в . Теперь если первая цифра равна 1, то состояние становится равно . Но если первая цифра равна 0, то состояние становится равно , и нам необходимо взять еще одну цифру, получив в итоге .

Проблема в том, что декодирование из заданного состояния может потребовать или для передачи, что означает, что для некоторого мы имеем: , в то время как . Это возможно, только если не делит .

Просуммируем наблюдения. Если делит , декодирование передает количество бит, фиксированное для каждого состояния . Таким образом, мы можем обрабатывать эти цифры блоками и хранить их в прямом или обратном порядке. Если же не делит , состояние само по себе иногда не может определять точное количество бит: мы должны для начала взять бит, и если все еще не находится в , добавить еще один бит[22-23].

2.3 Выводы по разделу

В данном разделе были подробно рассмотрены основные подходы ANS, такие как uABS, rABS, rANS, потоковые версии алгоритмов. Подходы uABS и rABS используются, когда необходимо эффективно закодировать данные, размер алфавита которых равен 2. rANS является расширением подхода rABS и является более эффективным, чем AC, так как состояние состоит из одного натурального числа вместо двух. Потоковые версии позволяют избежать работы с длинными числами, что существенно сокращает время работы, и удерживать состояние в определенном диапазоне, параллельно выводя уже обработанные биты в выходной поток.

В данной главе была теоретически оценена эффективность различных подходов ANS. В следующей главе будет проведен анализ эффективности алгоритмов на основе их программных реализаций.

3 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ И СРАВНИТЕЛЬНЫЙ АНАЛИЗ

3.1 Сравнительный анализ по времени работы

На основе программных реализаций алгоритмов Шеннона Фано, Хаффмана и потоковой версии rANS проведем их сравнительный анализ по времени работы. Протестируем методы на различных текстах с размерами алфавита 161, 97, 27 и 11 символов. Данные размеры алфавитов выбраны не случайно. Таблица ASCII кодов состоит из 256 символов, из них 161 – используемые при написании текстов символы (латинские и русские буквы, знаки препинания и перевод строки), 97 – латинские буквы, знаки препинания и перевод строки, 27 – маленькие латинские буквы и перевод строки и 11 – цифры и перевод строки. Для каждого размера алфавита сгенерируем по 10 тестов соответственных размеров, как указано в таблице, и выполним измерения времени работы. Результаты измерений и сравнительного анализа представлены в таблицах 3.1 – 3.8 и рисунках 3.1 – 3.13:

Таблица 3.1

Сравнение процессов кодирования на тестах с размером алфавита 161 символ

Количество символов в тесте	Кодирование Шеннона Фано, мс	Кодирование Хаффмана, мс	Потоковая версия rANS, мс
500	5	2	1
1000	5	2	1
5000	15	9	5
10000	16	13	12
50000	62	55	48
100000	110	102	95
500000	484	490	488
1000000	1000	990	950
5000000	4750	4850	4600
10000000	9454	10480	9221

Таблица 3.2

Сравнение процессов декодирования на тестах с размером алфавита 161 символ

Количество символов в тесте	Кодирование Шеннона Фано, мс	Кодирование Хаффмана, мс	Потоковая версия rANS, мс
500	5	1	1
1000	5	2	1
5000	15	9	6
10000	15	14	15
50000	54	55	54
100000	109	105	107
500000	500	495	510
1000000	953	1005	1042
5000000	4712	4750	5137
10000000	9382	9411	10133

Таблица 3.3

Сравнение процессов кодирования на тестах с размером алфавита 97 символов

Количество символов в тесте	Кодирование Шеннона Фано, мс	Кодирование Хаффмана, мс	Потоковая версия rANS, мс
500	5	1	1
1000	5	2	2
5000	15	9	6
10000	15	13	12
50000	50	45	44
100000	98	92	85
500000	450	467	432
1000000	913	954	870
5000000	4492	4365	4182
10000000	8772	9008	8450

Таблица 3.4

Сравнение процессов декодирования на тестах с размером алфавита 97 символов

Количество символов в тесте	Кодирование Шеннона Фано, мс	Кодирование Хаффмана, мс	Потоковая версия rANS, мс
500	5	1	1
1000	5	2	2
5000	10	6	5
10000	15	16	12
50000	46	49	50
100000	108	86	101
500000	455	448	486
1000000	914	900	1015
5000000	4467	4333	4580
10000000	8673	8600	8950

Таблица 3.5

Сравнение процессов кодирования на тестах с размером алфавита 27 символов

Количество символов в тесте	Кодирование Шеннона Фано, мс	Кодирование Хаффмана, мс	Потоковая версия rANS, мс
500	1	1	1
1000	2	2	2
5000	5	7	5
10000	10	10	9
50000	33	37	35
100000	68	70	65
500000	343	331	325
1000000	665	665	635
5000000	3310	3201	3050
10000000	6440	6647	5850

Таблица 3.6

Сравнение процессов декодирования на тестах с размером алфавита 27 символов

Количество символов в тесте	Кодирование Шеннона Фано, мс	Кодирование Хаффмана, мс	Потоковая версия rANS, мс
500	1	1	1
1000	2	2	2
5000	7	6	5
10000	12	11	10
50000	40	38	43
100000	70	74	78
500000	359	357	363
1000000	705	691	722
5000000	3372	3346	3550
10000000	6789	6780	7090

Таблица 3.7

Сравнение процессов кодирования на тестах с размером алфавита 11 символов

Количество символов в тесте	Кодирование Шеннона Фано, мс	Кодирование Хаффмана, мс	Потоковая версия rANS, мс
500	1	1	1
1000	2	2	1
5000	5	4	4
10000	8	8	7
50000	26	24	25
100000	48	53	47
500000	258	248	236
1000000	390	480	430
5000000	2373	2415	2220
10000000	4635	4744	4350

Таблица 3.8

Сравнение процессов декодирования на тестах с размером алфавита 11 символов

Количество символов в тесте	Кодирование Шеннона Фано, мс	Кодирование Хаффмана, мс	Потоковая версия rANS, мс
500	1	1	1
1000	2	2	1
5000	6	5	6
10000	11	8	8
50000	35	31	29
100000	66	54	60
500000	285	284	301
1000000	570	545	560
5000000	2752	2727	2840
10000000	5322	5298	5590

На основе полученных измерений для каждого размера алфавита построим графики зависимости времени работы методов от размера файла.

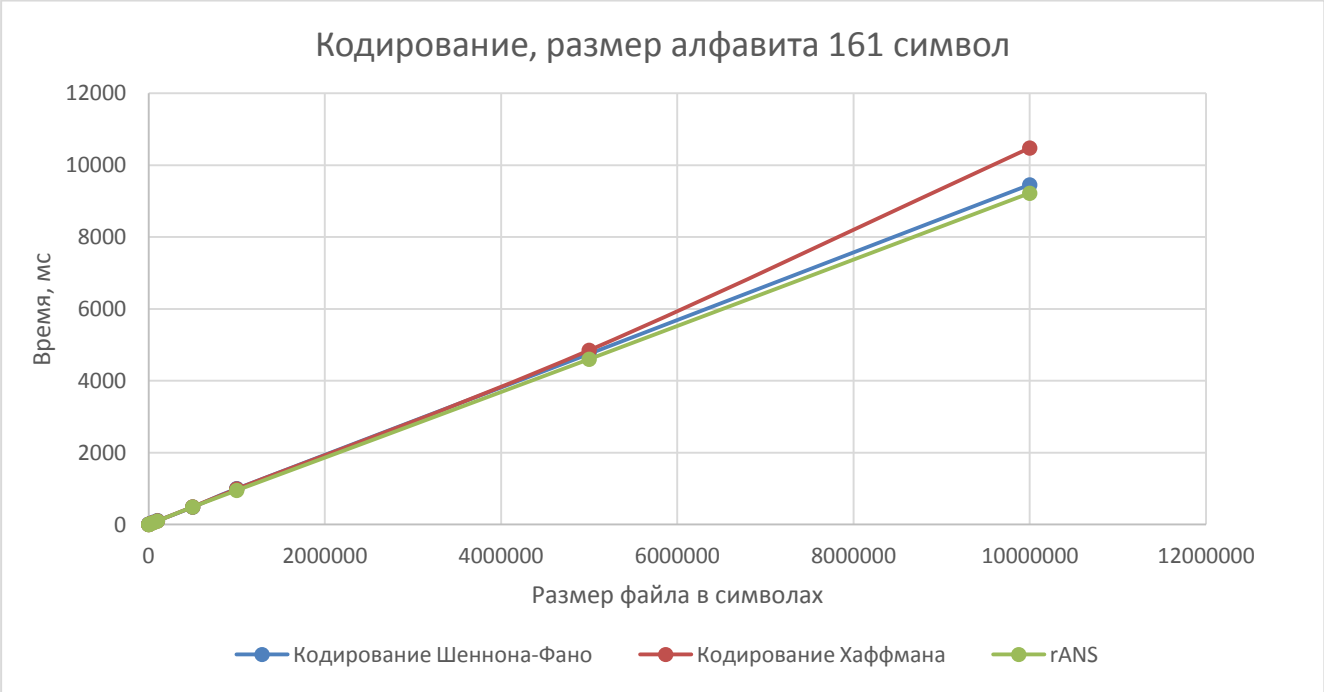

Рисунок 3.1. Сравнение результатов кодирования по времени для алфавита из 161 символа

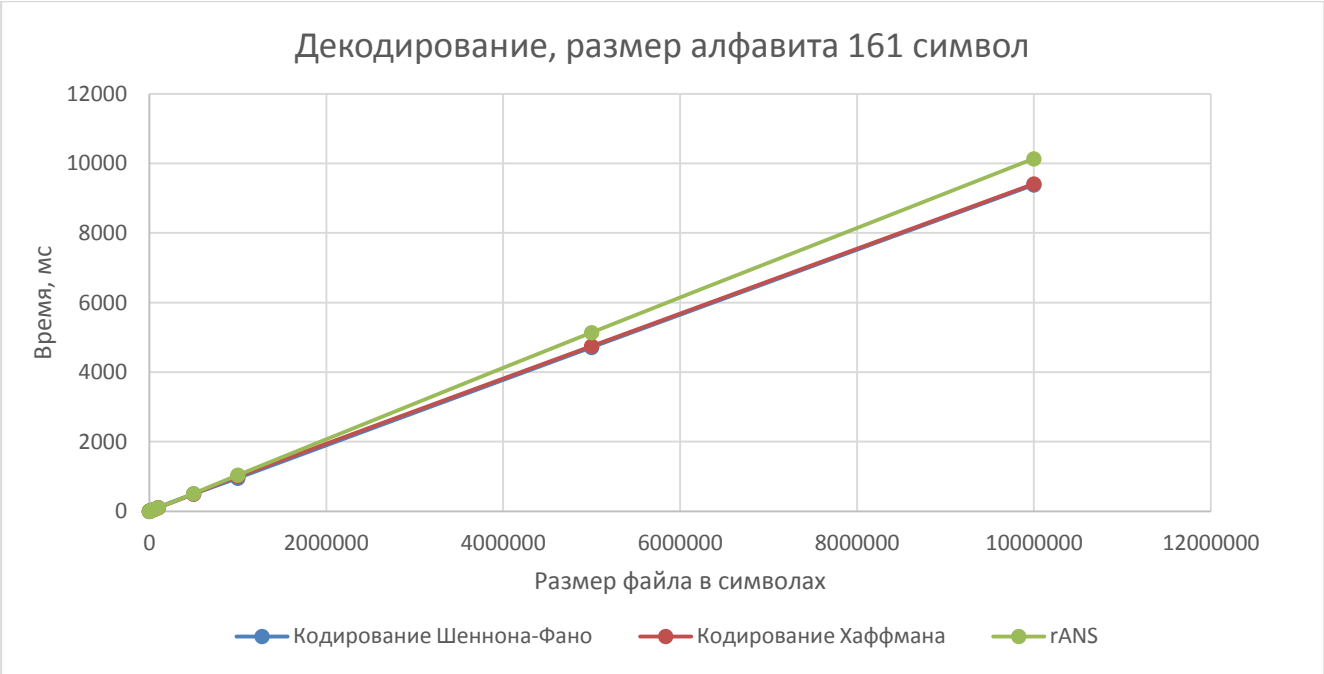

Рисунок 3.2. Сравнение результатов декодирования по времени для алфавита из 161 символа

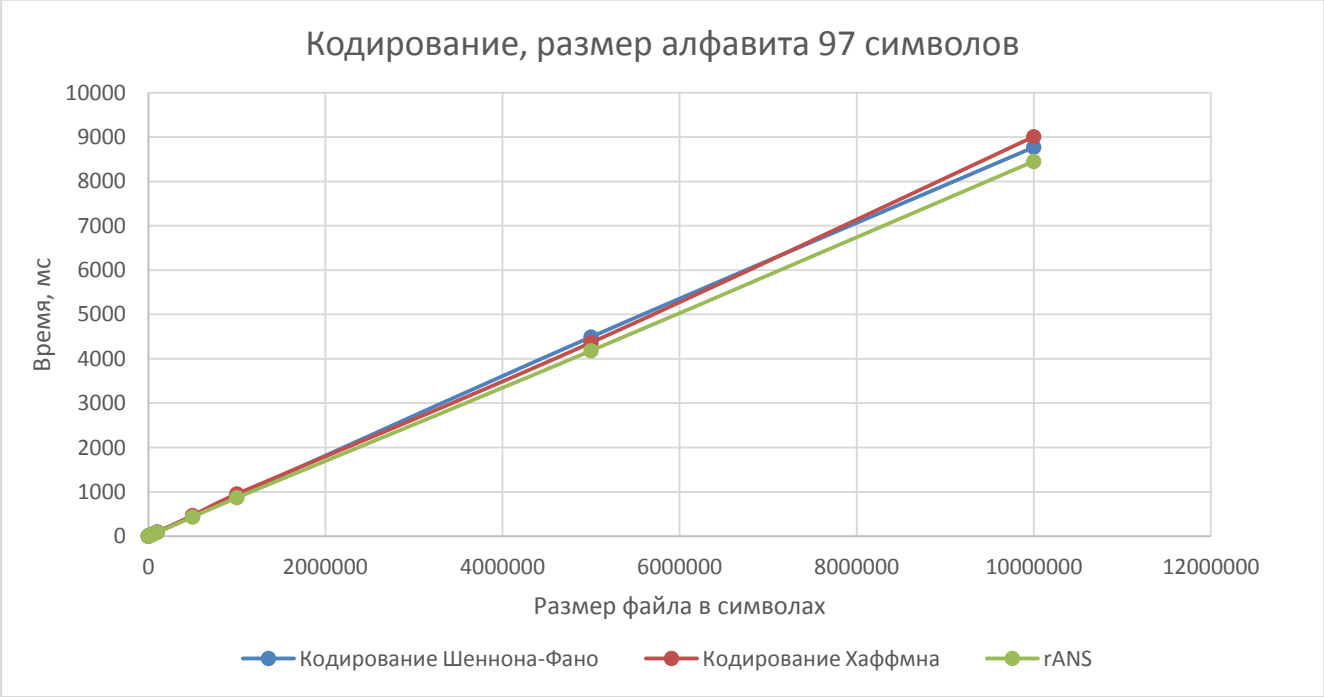

Рисунок 3.3. Сравнение результатов кодирования по времени для алфавита из 97 символов

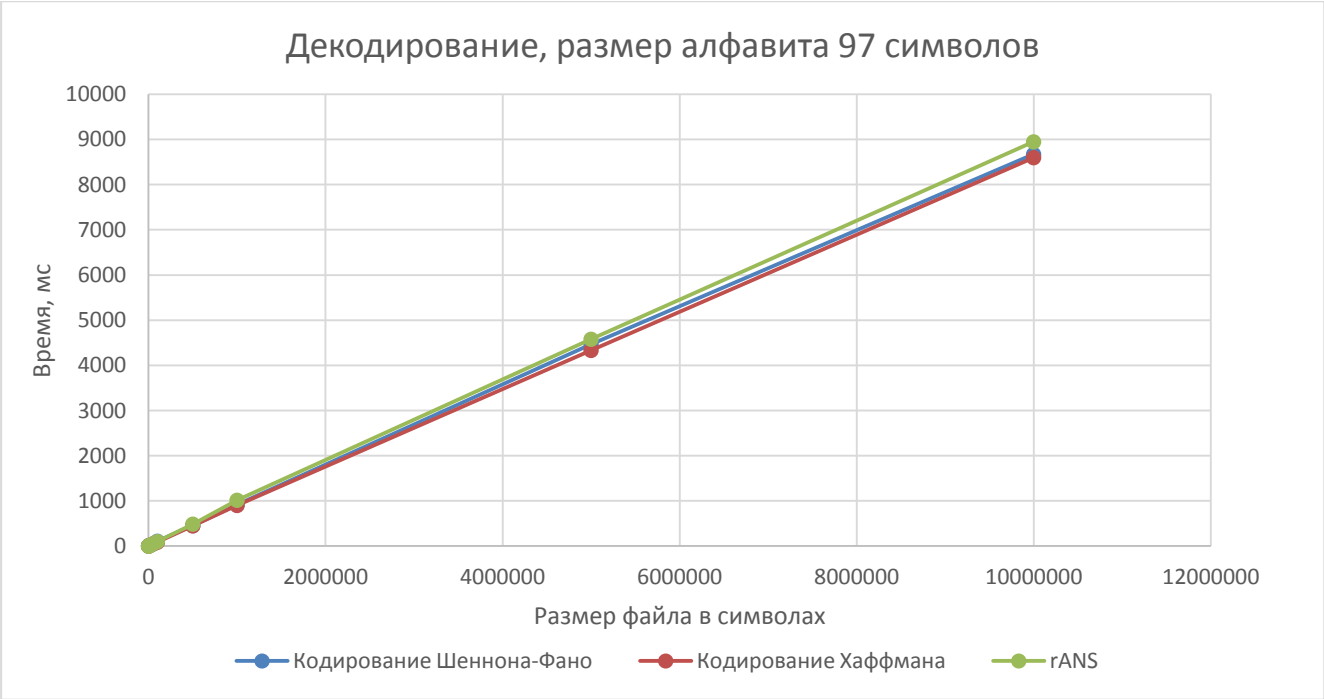

Рисунок 3.4. Сравнение результатов декодирования по времени для алфавита из 97 символов

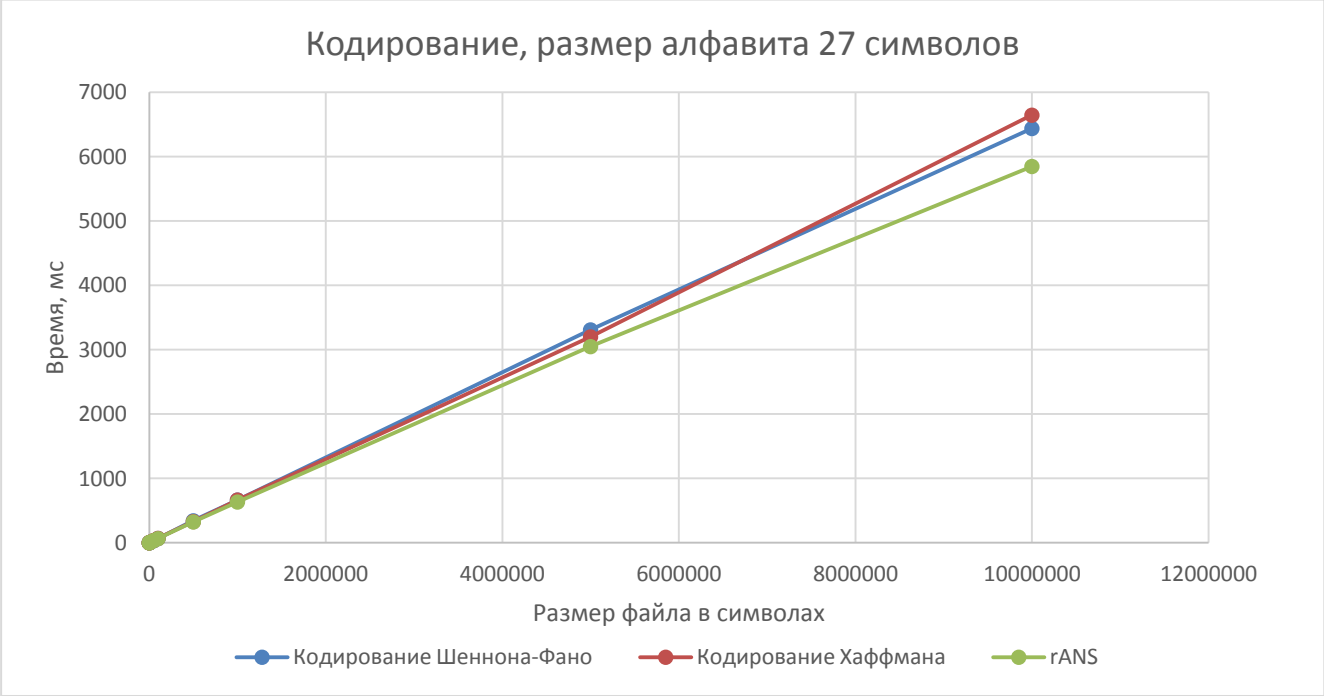

Рисунок 3.5. Сравнение результатов кодирования по времени для алфавита из 27 символов

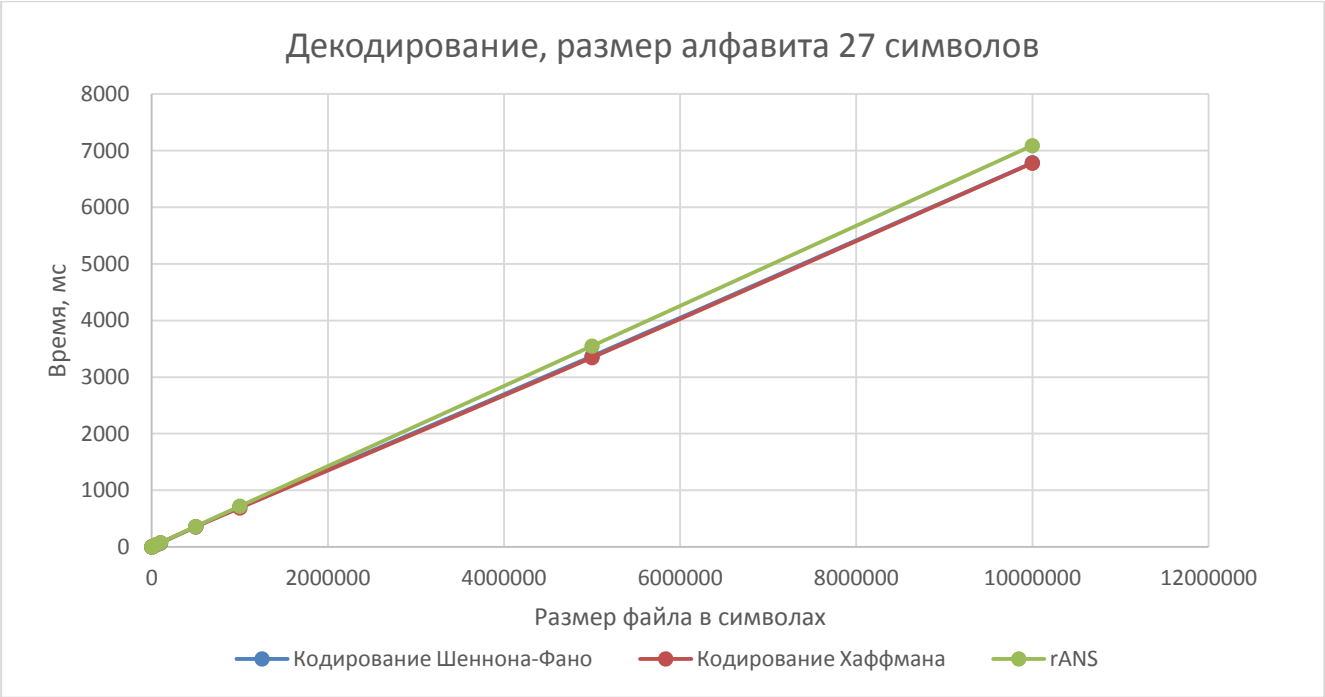

Рисунок 3.6. Сравнение результатов декодирования по времени для алфавита из 27 символов

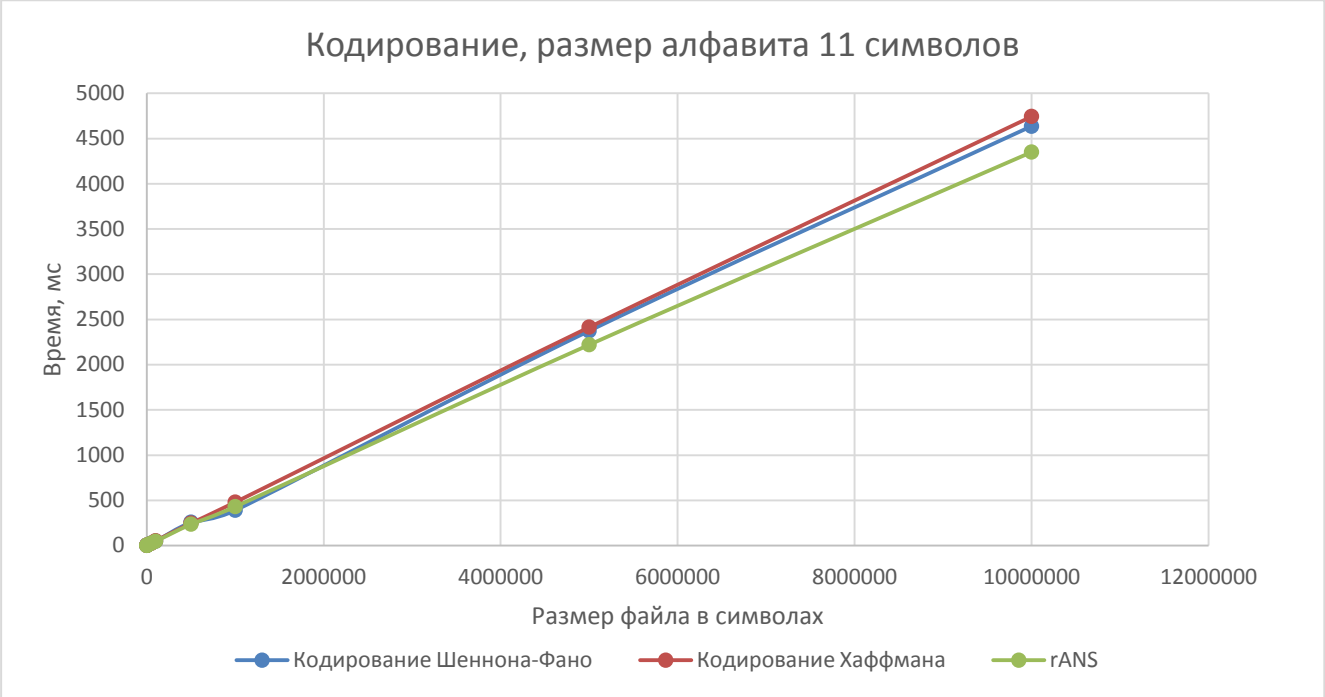

Рисунок 3.7. Сравнение результатов кодирования по времени для алфавита из 11 символов

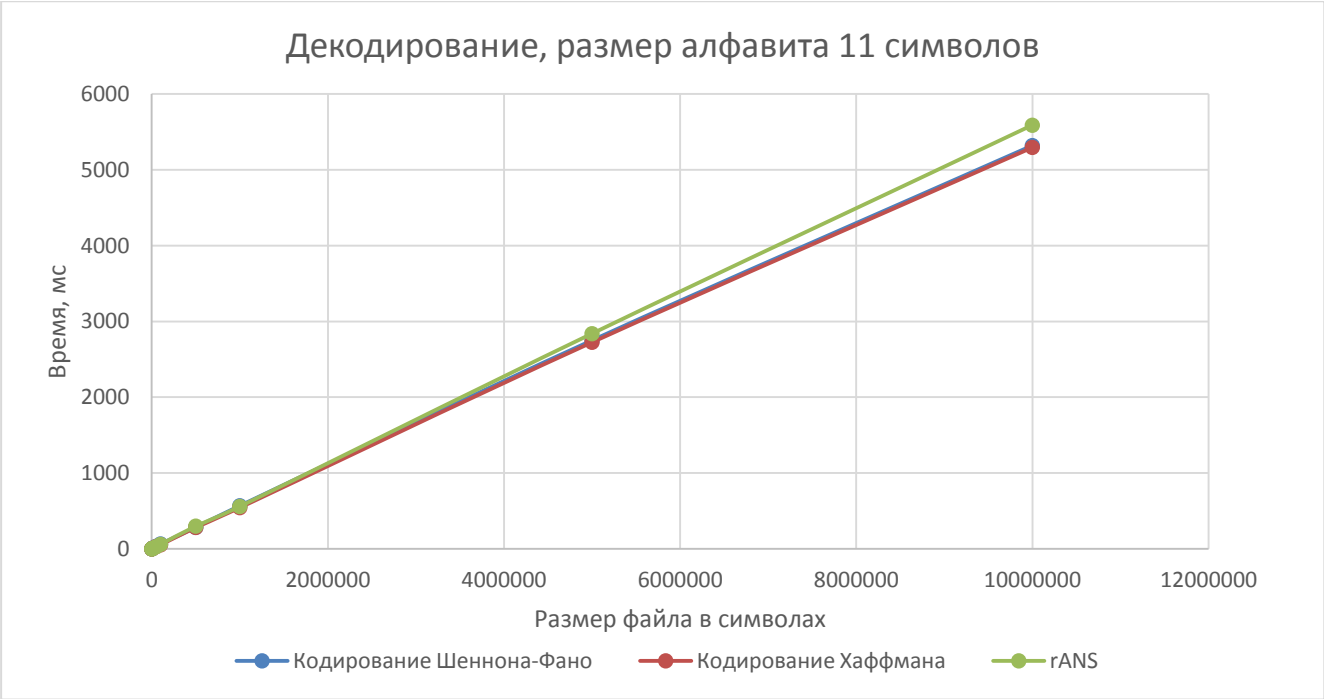

Рисунок 3.8. Сравнение результатов кодирования по времени для алфавита из 11 символов

Создадим сводные графики для каждого размера алфавита, объединив данные по кодированию и декодированию вместе.

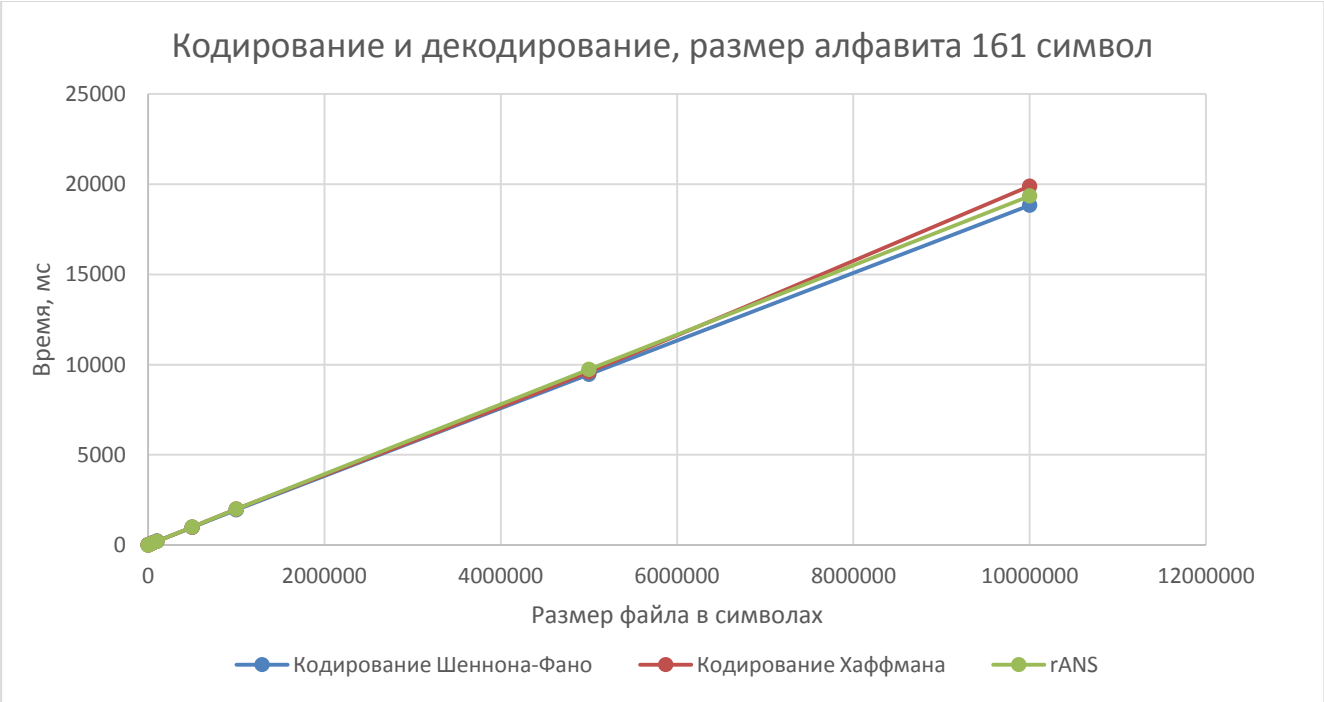

Рисунок 3.9. Сравнение объединенных результатов кодирования и декодирования по времени для алфавита из 161 символа

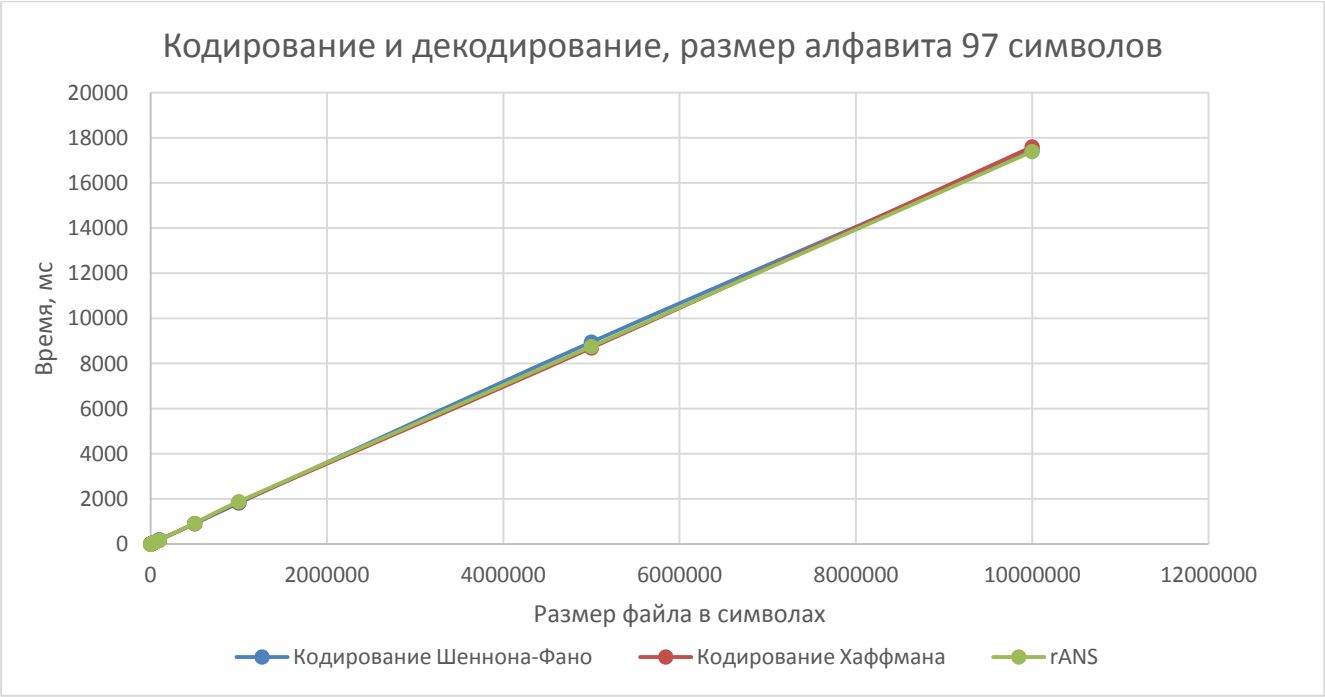

Рисунок 3.10. Сравнение объединенных результатов кодирования и декодирования по времени для алфавита из 97 символов

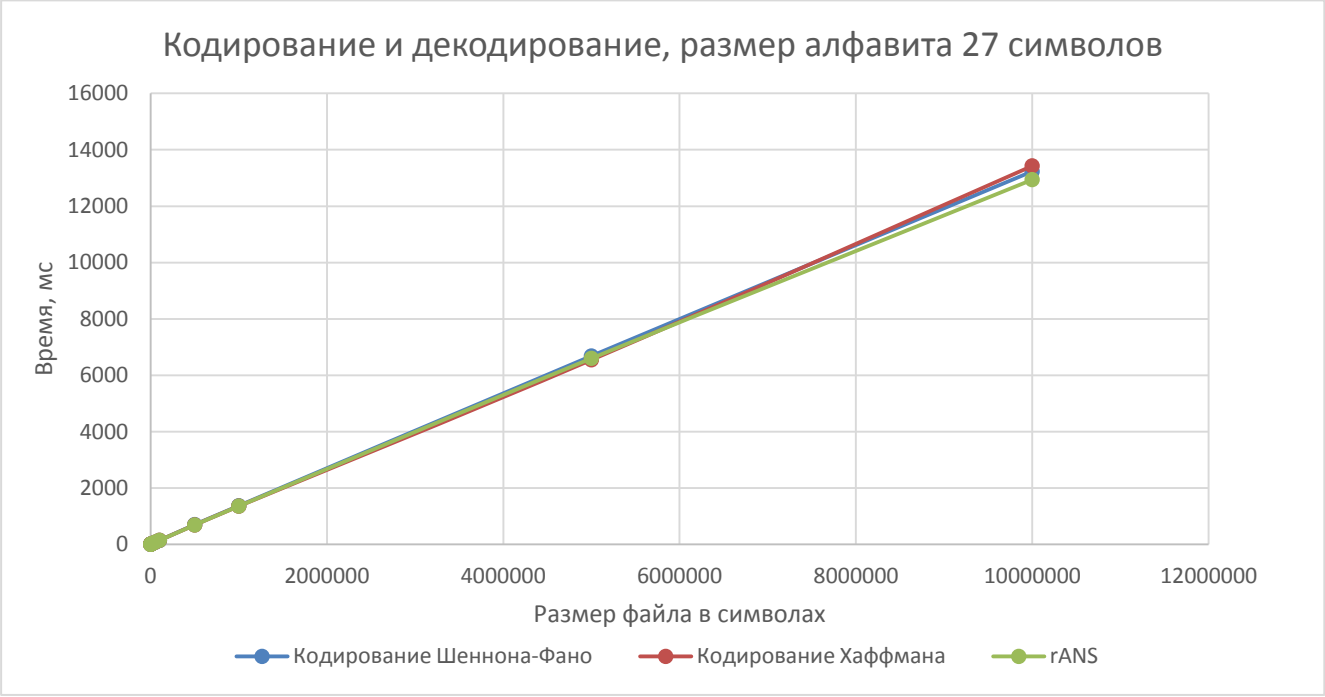

Рисунок 3.11. Сравнение объединенных результатов кодирования и декодирования по времени для алфавита из 27 символов

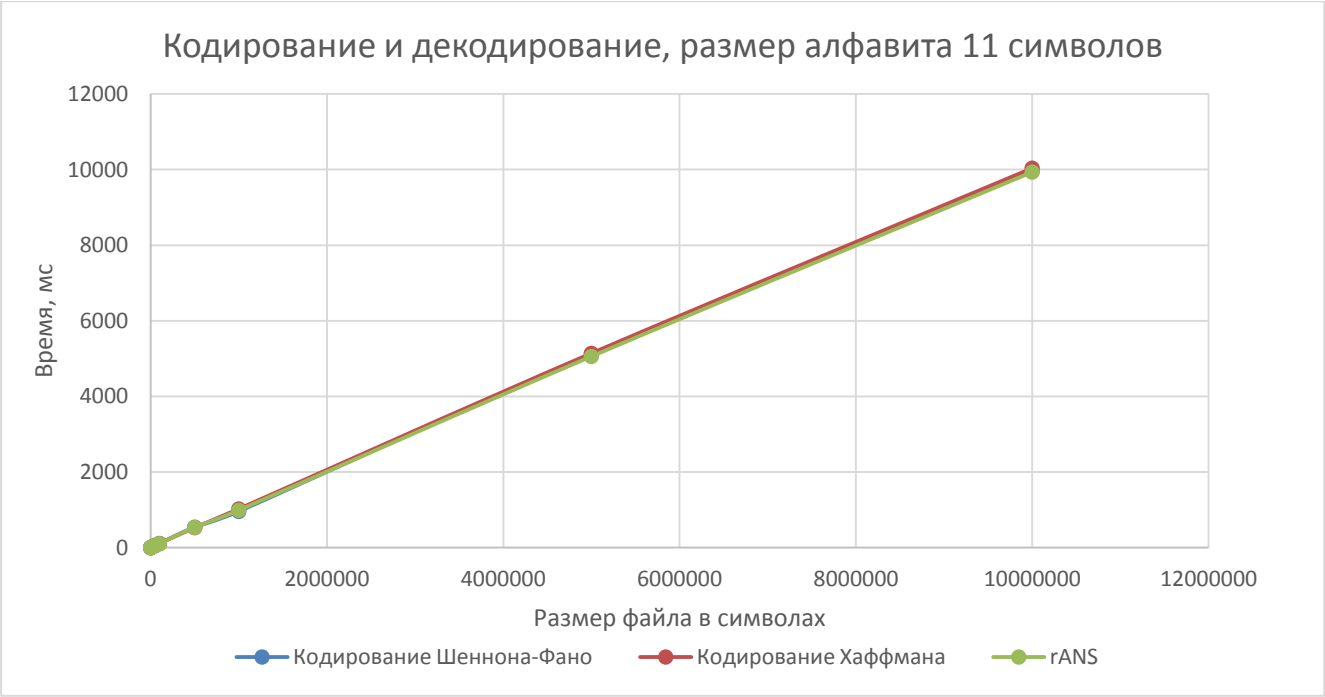

Рисунок 3.12. Сравнение объединенных результатов кодирования и декодирования по времени для алфавита из 11 символов

Проанализируем зависимость времени работы алгоритмов от размера алфавита. Для этого протестируем алгоритмы на файлах размером символов, но с различными размерами алфавита.

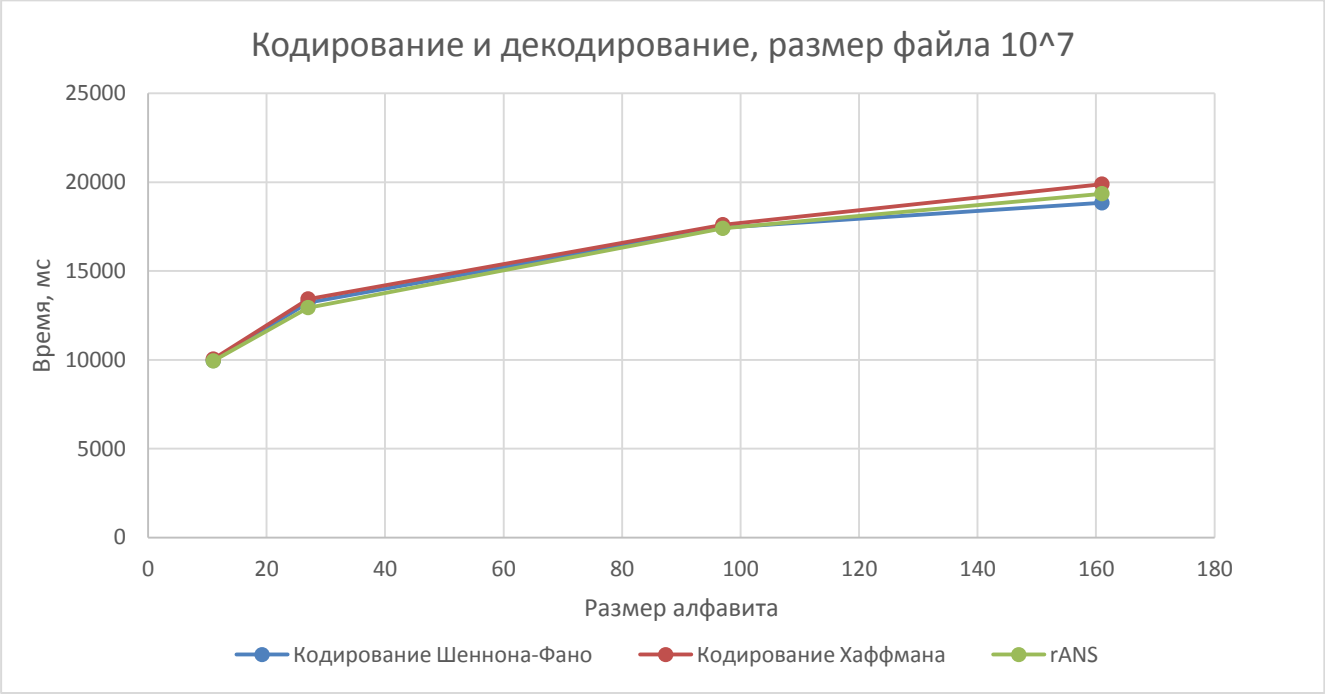

Рисунок 3.13. Сравнение объединенных результатов кодирования и декодирования по времени для файлов размера

Из проведенного анализа можно сделать вывод, что ANS работают так же быстро, как алгоритм Хаффмана и его модификации для различных алфавитов и размеров файлов, а также время работы эффективных энтропийных методов растет с увеличением размера алфавита, что подтверждает теоретические оценки зависимости времени работы алгоритмов от размера алфавита.

3.2 Сравнительный анализ по коэффициенту сжатия

На основе программных реализаций арифметического кодирования и потоковой версии rANS проведем их сравнительный анализ по коэффициенту сжатия. Коэффициент сжатия будем пересчитывать как среднее количество бит на символ. Протестируем методы на различных текстах с размерами алфавита 161, 97, 27 и 11 символов. Для каждого размера алфавита сгенерируем по 10 тестов с размерами файлов от 100 до 1000 символов с шагом 100. Такие размеры выбраны для того, чтобы время работы арифметического кодирования укладывалось в разумные рамки. Для каждого теста измерим энтропию текста, среднее количество бит на символ для методов rANS и арифметического кодирования без учета таблицы вероятностей и с учетом таблицы.

Кодирование без учета таблицы вероятностей может применяться, когда заранее известно почти точное распределение вероятностей, либо нет необходимости в максимально возможной степени сжатия и известно примерное распределение, как например при кодировании литературных текстов.

Кодирование с учетом передачи таблицы вероятностей используется, кода нельзя сказать ничего определенного о распределении вероятностей символов в кодируемых данных. Такая таблица строится при помощи дополнительного просмотра данных до начала процесса кодирования.

В процессе тестирования была построена таблица вероятностей, так как тесты были созданы при помощи использования генератора случайных чисел, и для таких данных ничего нельзя сказать о распределении вероятностей, что в основном характерно для данных научных и медицинских измерений. Результаты измерений и сравнительного анализа представлены в таблицах 3.9 – 3.12 и рисунках 3.14 – 3.17:

Таблица 3.9

Сравнение процессов кодирования на тестах с размером алфавита 161 символ

Количество символов в тесте	Энтропия	rANS (только код)	Арифметич. кодирование (только код)	rANS (с таблицей)	Арифметич. кодирование (с таблицей)
100	6.109	6.100	6.070	26.580	26.230
200	6.637	6.630	6.635	22.550	22.395
300	6.967	6.973	6.967	20.787	20.673
400	7.007	7.010	7.001	17.990	17.900
500	7.074	7.076	7.074	16.004	15.938
600	7.092	7.092	7.088	14.665	14.608
700	7.150	7.156	7.151	13.784	13.734
800	7.141	7.144	7.141	12.894	12.851
900	7.173	7.177	7.172	12.332	12.292
1000	7.205	7.209	7.204	11.945	11.900

Таблица 3.10

Сравнение процессов кодирования на тестах с размером алфавита 97 символов

Количество символов в тесте	Энтропия	rANS (только код)	Арифметич. кодирование (только код)	rANS (с таблицей)	Арифметич. кодирование (с таблицей)
100	5.607	5.620	5.611	21.300	20.971
200	6.191	6.190	6.194	18.231	18.072
300	6.371	6.381	6.372	15.713	15.597
400	6.408	6.415	6.408	13.595	13.508
500	6.477	6.482	6.478	12.306	12.238
600	6.486	6.492	6.487	11.343	11.287
700	6.483	6.488	6.483	10.853	10.803
800	6.490	6.494	6.491	10.244	10.211
900	6.528	6.530	6.529	10.059	10.022
1000	6.518	6.521	6.518	9.769	9.726

Таблица 3.11

Сравнение процессов кодирования на тестах с размером алфавита 27 символов

Количество символов в тесте	Энтропия	rANS (только код)	Арифметич. кодирование (только код)	rANS (с таблицей)	Арифметич. кодирование (с таблицей)
100	4.489	4.522	4.460	11.312	10.942
200	4.629	4.635	4.625	8.155	7.985
300	4.643	4.653	4.643	6.893	6.777
400	4.663	4.673	4.663	6.433	6.343
500	4.688	4.696	4.688	6.104	6.032
600	4.712	4.713	4.712	5.887	5.832
700	4.689	4.697	4.687	5.703	5.647
800	4.706	4.712	4.705	5.591	5.545
900	4.693	4.698	4.691	5.480	5.437
1000	4.694	4.699	4.689	5.411	5.361

Таблица 3.12

Сравнение процессов кодирования на тестах с размером алфавита 11 символов

Количество символов в тесте	Энтропия	rANS (только код)	Арифметич. кодирование (только код)	rANS (с таблицей)	Арифметич. кодирование (с таблицей)
100	3.162	3.170	3.151	6.135	5.791
200	3.361	3.365	3.369	4.965	4.853
300	3.406	3.423	3.407	4.487	4.367
400	3.363	3.375	3.365	4.175	4.085
500	3.355	3.365	3.356	4.005	3.932
600	3.372	3.382	3.372	3.928	3.865
700	3.364	3.366	3.364	3.823	3.776
800	3.385	3.398	3.385	3.803	3.750
900	3.382	3.388	3.378	3.752	3.707
1000	3.363	3.369	3.359	3.705	3.655

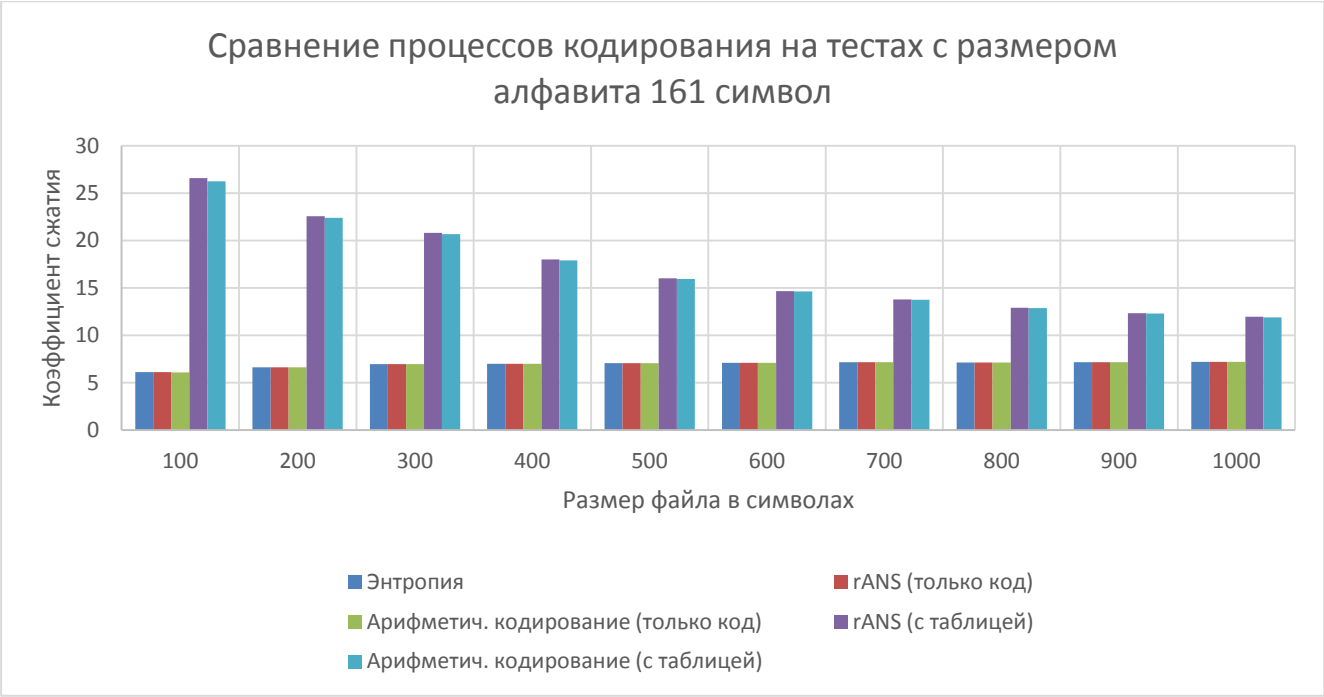

Рисунок 3.14. Сравнение результатов кодирования по коэффициенту сжатия для алфавита из 161 символа

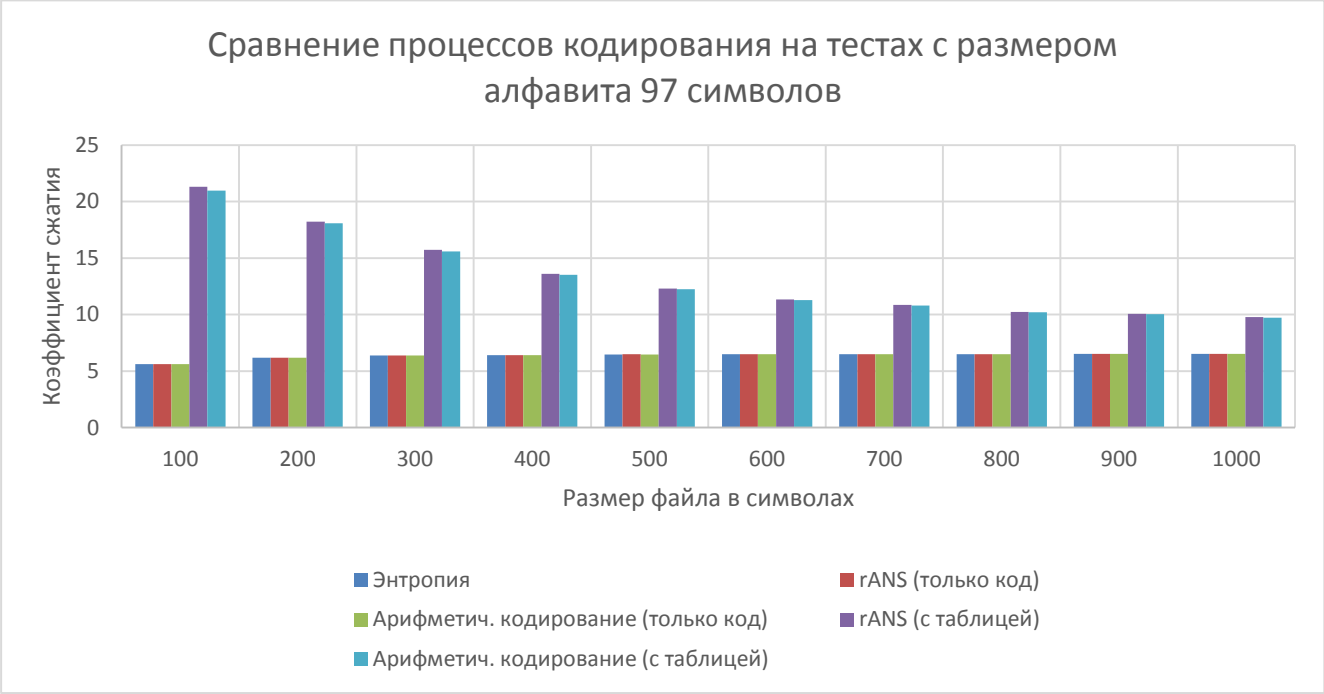

Рисунок 3.15. Сравнение результатов кодирования по коэффициенту сжатия для алфавита из 97 символов

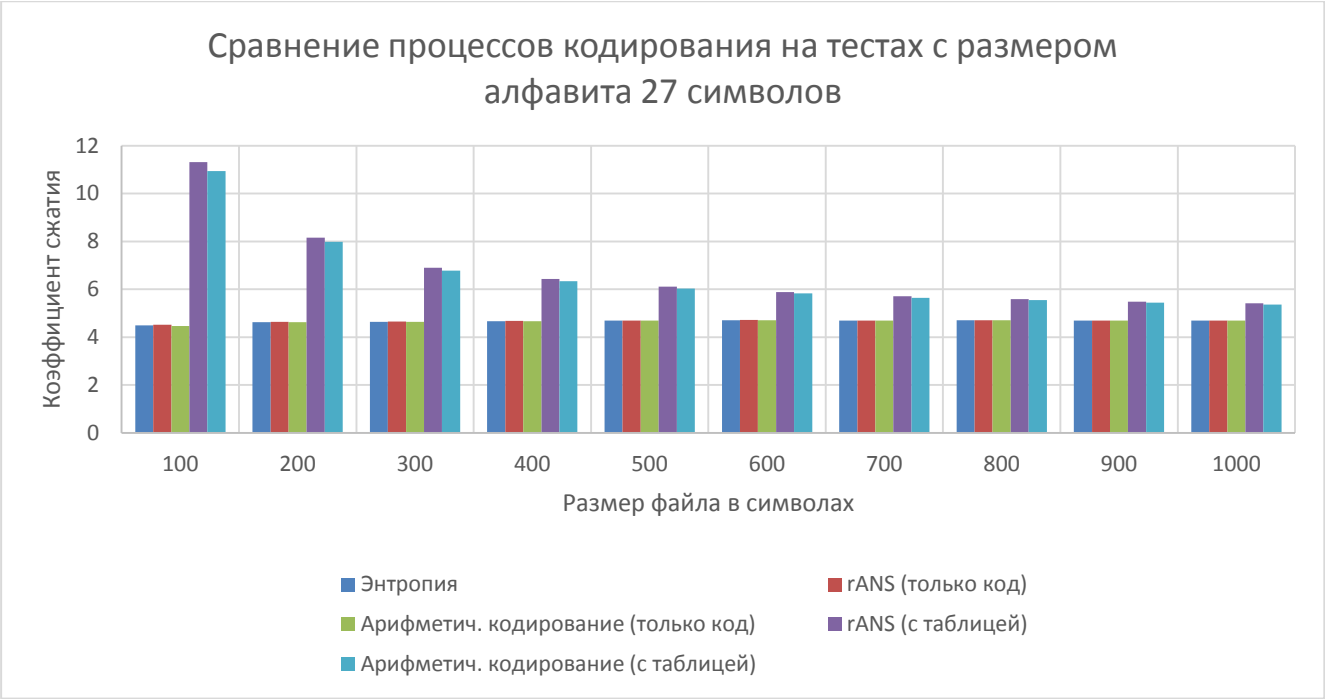

Рисунок 3.16. Сравнение результатов кодирования по коэффициенту сжатия для алфавита из 27 символов

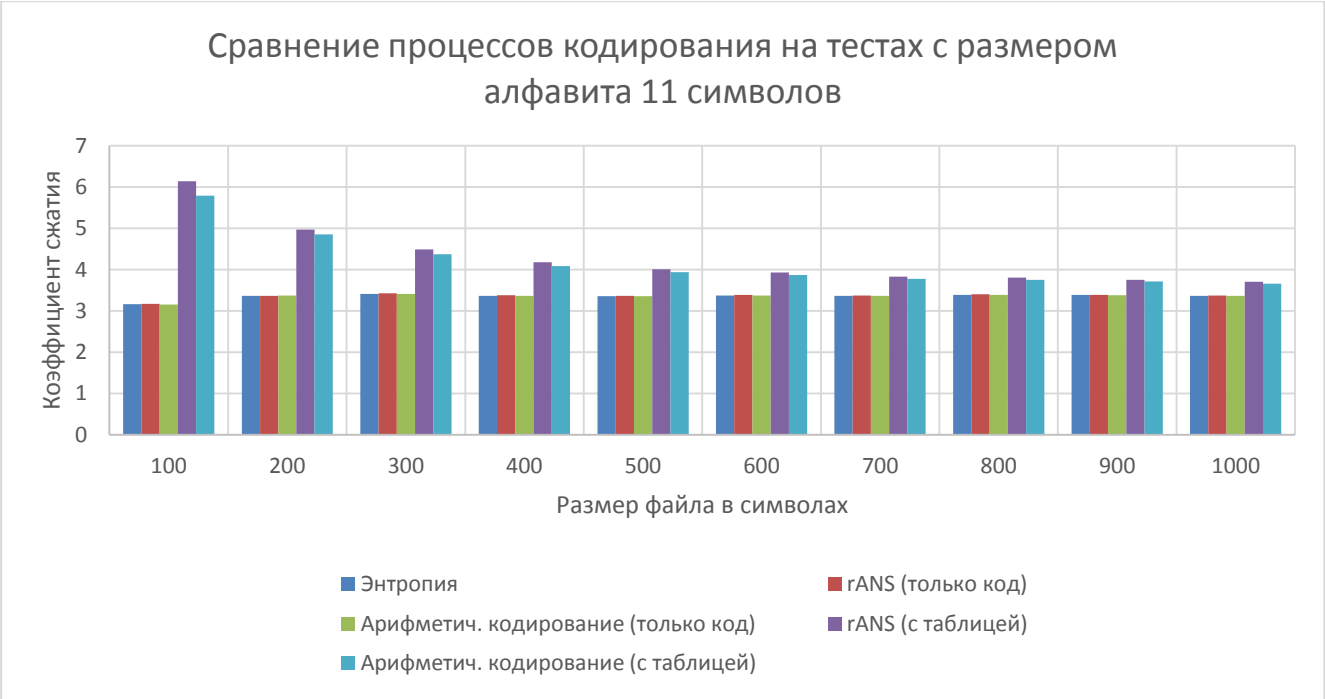

Рисунок 3.17. Сравнение результатов кодирования по коэффициенту сжатия для алфавита из 11 символов

По результатам измерений можно сделать следующие выводы. Кодирование без учета передачи таблицы вероятностей показывает для обоих методов результаты, максимально приближенные к энтропии текста. Такой результат получен из-за того, что при кодировании мы использовали точную таблицу

вероятностей, подсчитанную заранее. При использовании приближенных таблиц коэффициент сжатия будет больше, это зависит от точности используемой таблицы.

Кодирование с учетом таблицы вероятностей показывает сильно удаленные от энтропии результаты при больших алфавитах и маленьких размерах кодируемого файла, так как таблица требует передачи информации о каждом символе, и для маленьких файлов таблица получается больше кода. Но при увеличении размера файла и при уменьшении размера алфавита коэффициент даже с учетом таблицы вероятностей становится близким к энтропии у обоих методов, так как размер таблицы становится незначительным по сравнению с размером кода.

Таким образом, коэффициент сжатия rANS очень близок к коэффициенту сжатия арифметического кодирования, и оба этих метода кодируют информацию с количеством бит на символ, близким к энтропии.

3.3 Пользовательский интерфейс для демонстрации результатов работы

После запуска приложения открывается следующая форма «рисунок 3.18»:

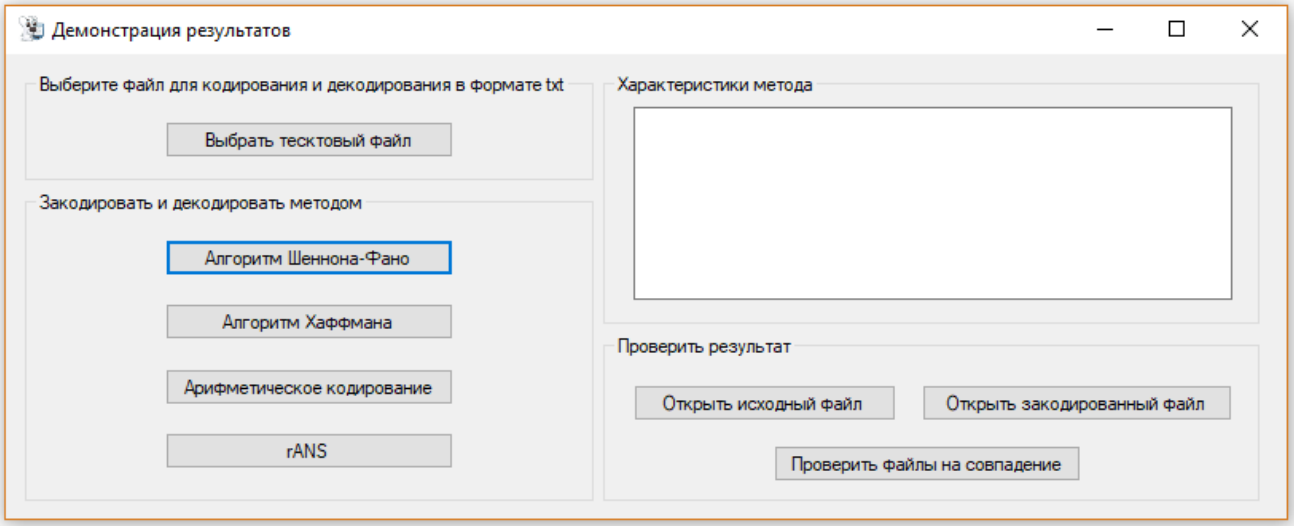

Рисунок 3.18. Главное окно демонстрации результатов работы

Для того чтобы выбрать текстовый файл для демонстрации работы методов, необходимо нажать кнопку «Выбрать текстовый файл». После нажатия этой кнопки открывается диалоговое окно для выбора файла в формате (*.txt) «рисунок 3.19»:

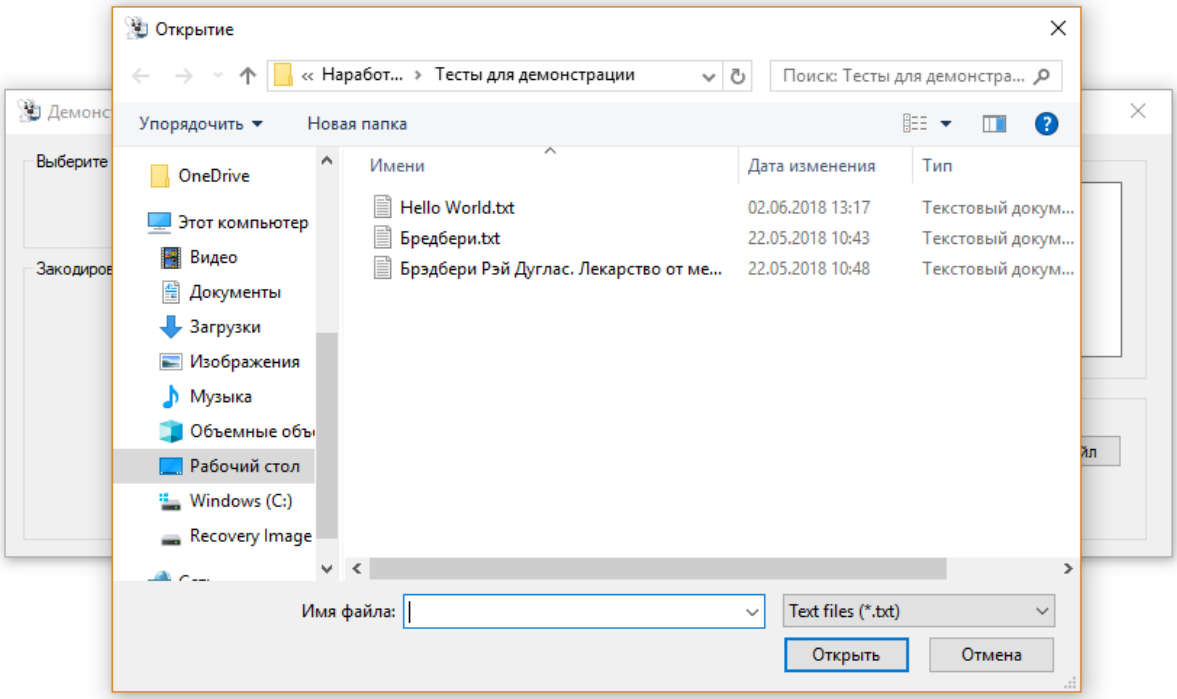

Рисунок 3.19. Диалоговое окно для выбора текстового файла в формате (*.txt)

После выбора текстового файла можно начать демонстрацию работы методов. На выбор представлено четыре метода – алгоритм Шеннона–Фано, Хаффмана, арифметическое кодирование и потоковая версия rANS. После выбора метода кодирования необходимо дать согласие на начало кодирования «рисунок 3.20»:

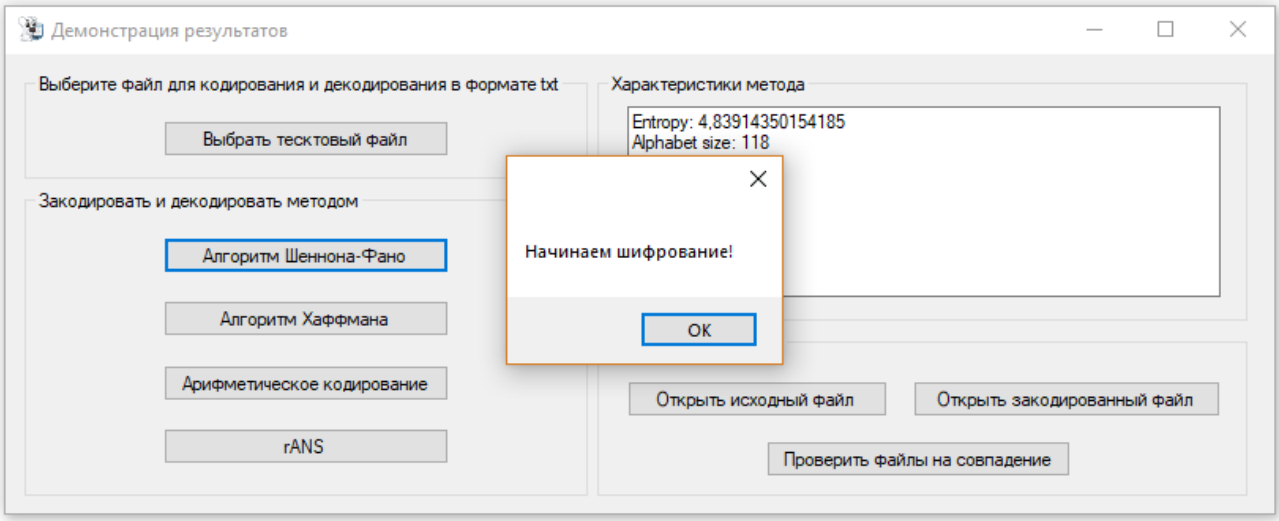

Рисунок 3.20. Окно для подтверждения процесса кодирования

После получения разрешения начинается процесс кодирования исходного текста и затем декодирование его обратно выбранным методом. В процессе происходит измерение времени работы, энтропии, размера алфавита, размера

кодирующей таблицы и коэффициентов сжатия с учетом таблицы и без учета таблицы. После завершения процесса выдается сообщение об окончании кодирования «рисунок 3.21»:

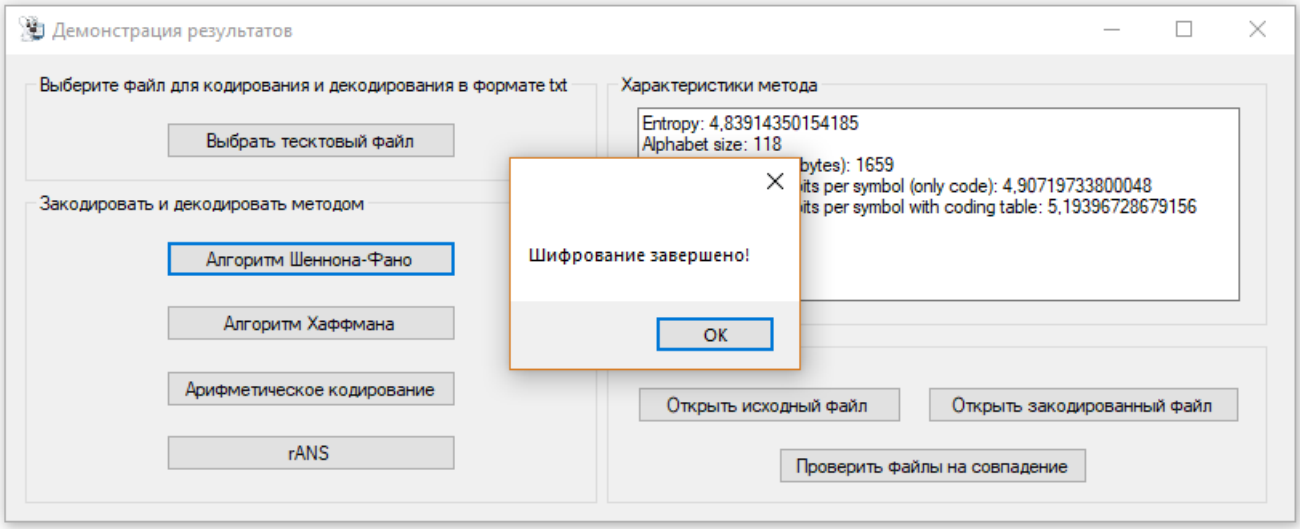

Рисунок 3.21. Завершение процесса кодирования

После завершения процесса в окне можно увидеть все характеристики отработавшего метода «рисунок 3.22»:

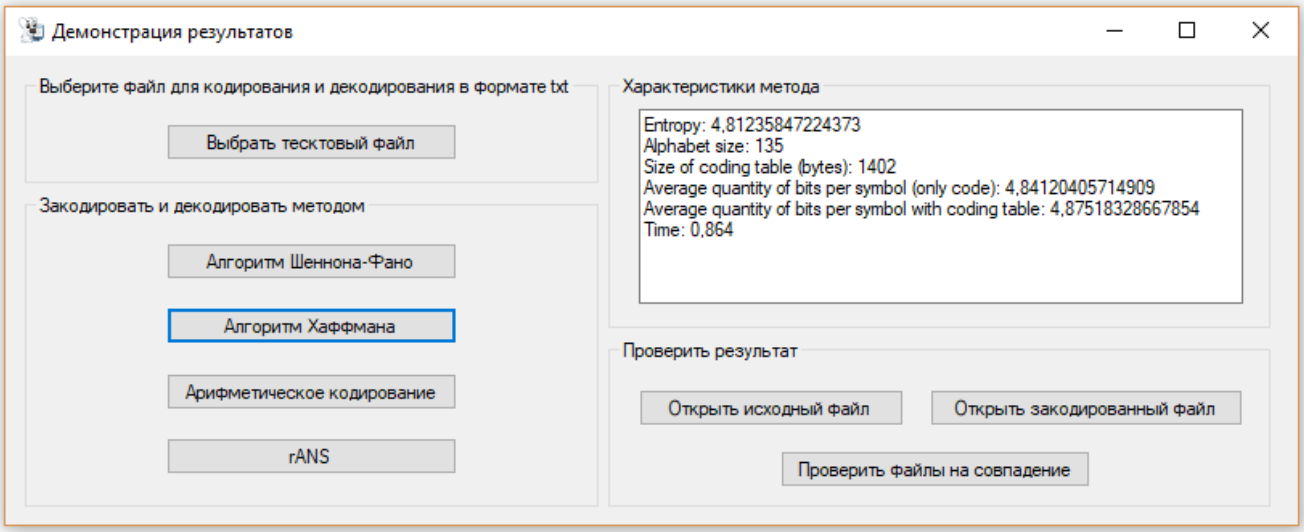

Рисунок 3.22. Характеристики метода

Полученные после кодирования и декодирования файлы можно открыть, нажав на соответствующие кнопки, а можно проверить на совпадение «рисунок 3.23»:

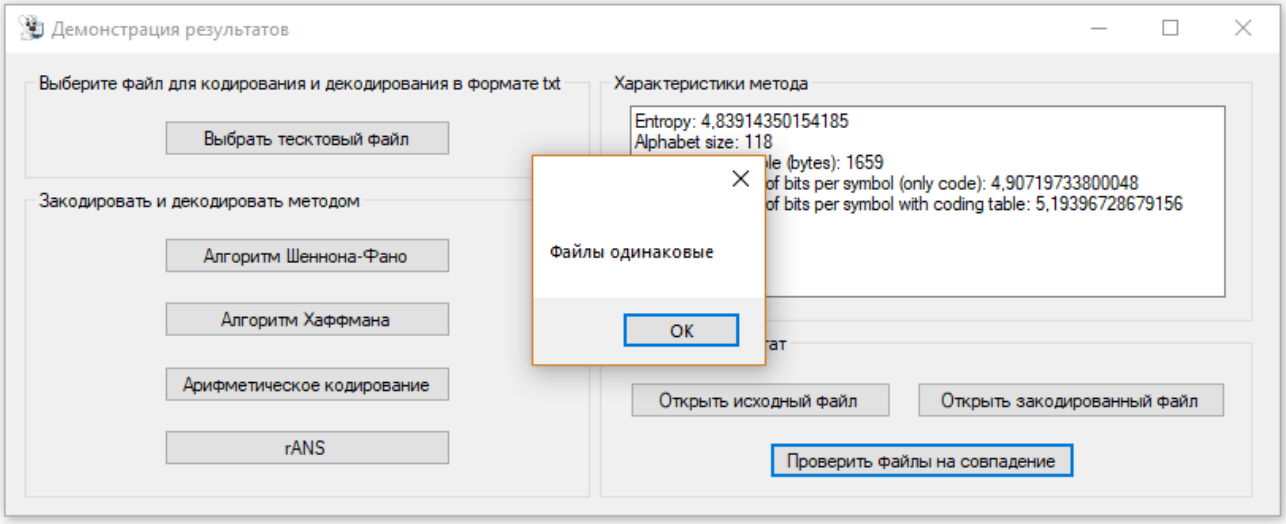

Рисунок 3.23. Проверка файлов на совпадение

Совпадение файлов говорит о том, что кодирование и декодирование взаимно однозначны, а значит работают корректно. В случае если какое-либо действие на форме было выбрано до того, как был указан файл для кодирования, будет выдано следующее сообщение об ошибке «рисунок 3.24»:

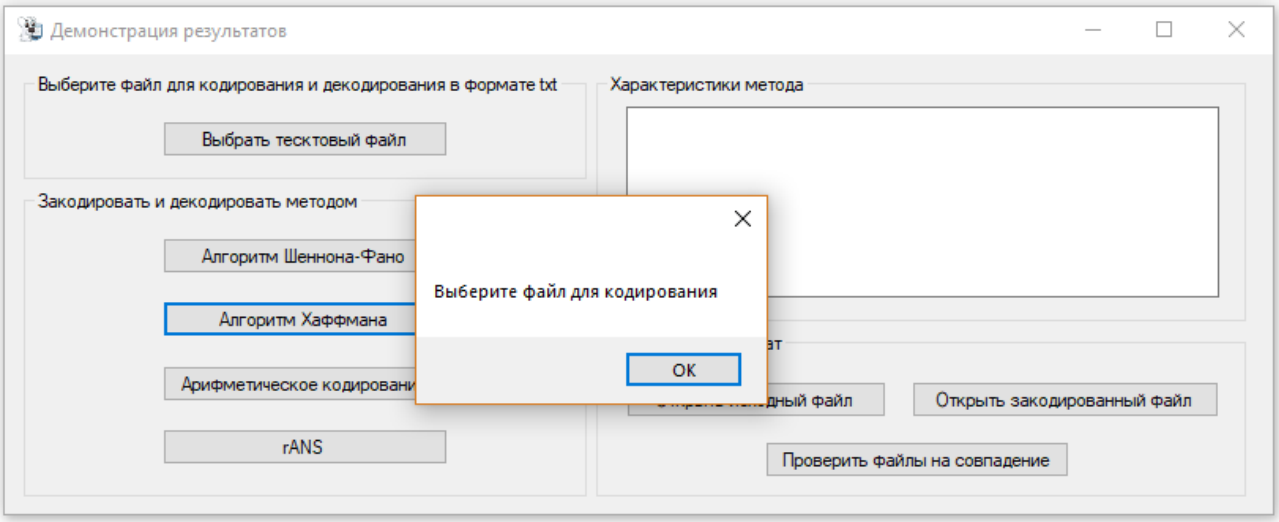

Рисунок 3.24. Сообщение о необходимости выбрать текстовый файл

3.4 Выводы по разделу

В разделе были проведены измерения времени работы и коэффициента сжатия методов энтропийного кодирования на основе различных экспериментальных данных и построены графики зависимостей времени работы и коэффициента сжатия от размера файла в символах, а также времени работы от размера алфавита. Также был реализован и представлен пользовательский интерфейс для демонстрации результатов работы методов.

ЗАКЛЮЧЕНИЕ

Данная работа посвящена исследованию асимметричных систем счисления как нового метода энтропийного кодирования информации.

В результате выполнения работы был проведен сравнительный анализ ANS с алгоритмами Хаффмана и Шеннона Фано по времени работы и с арифметическим кодированием по коэффициенту сжатия информации.

Для проведения сравнительного анализа эффективные методы энтропийного кодирования были реализованы на языке C++ в среде разработки Microsoft Visual Studio 2012. Для того чтобы измерить время работы методов, было сгенерировано по 10 тестов для каждого из четырех различных размеров алфавита: 161, 97, 27 и 11 символов. Данные размеры алфавитов выбраны не случайно. Таблица ASCII кодов состоит из 256 символов, из них 161 – используемые при написании текстов символы (латинские и русские буквы, знаки препинания и перевод строки), 97 – латинские буквы, знаки препинания и перевод строки, 27 – маленькие латинские буквы и перевод строки и 11 – цифры и перевод строки. Тестами являются файлы различного размера, наибольший размер – 10000000 символов. Для тестирования были выбраны текстовые файлы как наиболее простой и компактный вид входных данных, которые необходимо сжать без потерь. Для проведения сравнительного анализа по коэффициенту сжатия также были сгенерированы тесты различного размера и на различных алфавитах, однако размер файлов был существенно меньше из-за трудоемкости выполнения арифметического кодирования. Для демонстрации результатов работы методов был реализован пользовательский интерфейс.

В ходе сравнительного анализа был сделан вывод, что ANS комбинируют в себе скорость сжатия информации алгоритмом Хаффмана и коэффициент сжатия арифметического кодирования. Данный вывод подтверждает теоретические оценки. Эти свойства асимметричных систем делают их одним из наиболее эффективных методов сжатия информации без потерь на текущий момент.

Такие компании как Apple, Google, Facebook на данный момент успешно внедряют эту технологию в свои архиваторы и компрессоры, однако большинство из них защищены патентами. В будущем ANS получат широкое распространение и среди общедоступных методов сжатия данных.

В дальнейшем планируется рассмотреть приложения ANS в криптографии.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1 Ватолин, Д.С. Методы сжатия данных. Устройство архиваторов, сжатие изображений и видео / Д.С. Ватолин, А.С. Ратушняк, М.А. Смирнов [и др.] – Изд.-во: Диалог-МИФИ, 2002. – 384 с.

2 Сэломон, Д. Сжатие данных, изображений и звука / Д. Сэломон, пер. с англ. В.В.Чепыжова. – М.: Техносфера, 2004. – 368 с.

3 Уилсон, М. Американские ученые и изобретатели / М. Уилсон, пер. с англ. В.Б.Рамзеса; под ред. Н. Треневой. – М.: Знание, 1975. – с. 27–34. – 136 с.

4 Shannon, C. A mathematical theory of communication / C. Shannon // Bell System Technical Journal. – 1948. – Vol. 27, Iss. 3. – p. 379–423.

5 James, I. Claude Elwood Shannon 30 April 1916 – 24 February 2001 / James I., Stuart J.T. // Biographical Memoirs Fellows R. – The Royal Society, 2009. – Vol. 55, Iss. 0. – p. 257–265.

6 Кудряшов, Б.Д. Теория информации / Б.Д. Кудряшов. – Санкт-Петербург: СПбГУ ИТМО. – 2010. – 188 с.

7 Stephen, W. A New Kind of Science / W. Stephen. – Champaign, IL: Wolfram Media. – 2002. – 1069 p.

8 History of Lossless Data Compression Algorithms. – Дата обновления: 20.08.2014. URL: http://ethw.org/History_of_Lossless_Data_Compression_Algorithms (дата обращения: 15.04.2018).

9 Asymmetric Numeral Systems. – Дата обновления: 05.06.2018. URL: https://en.wikipedia.org/wiki/Asymmetric_numeral_systems (дата обращения: 15.04.2018).

10 Dialogic ADPCM. – Дата обновления: 13.01.2018. URL: https://en.wikipedia.org/wiki/Dialogic_ADPCM (дата обращения: 15.04.2018).

11 Pearlman, W.A. Digital Signal Compression: Principles and Practice / William A. Pearlman, Amir Said. – Cambridge University Press, ISBN 9780521899826, 2011. – 419 p.

12 Яглом, А. М. Вероятность и информация / А. М. Яглом, И. М. Яглом. – Наука, 1973. – 512 с.

13 Аршинов, М.Н. Коды и математика / М. Н. Аршинов, Л. Е. Садовский. – Коды и математика. – М.: Наука, 1983. – 144 с.

14 Камерон, П. Теория графов. Теория кодирования и блок-схемы / П. Камерон, Джю ван Линт – М.: Мир, 1980. – 144 с.

15 Касами, Т. Теория кодирования / Т. Касами. – М.: Мир, 1978. – 576 с.

16 Лидовский, В. В. Теория информации / В. В. Лидовский. – М.: Компания Спутник+, 2004. – 111 с.

17 Реньи, А. Трилогия о математике / А.Реньи. – М.: Мир, 1980. – 376 с.

18 Ромащенко, А. Заметки по теории кодирования / А. Е. Ромащенко, А. Ю. Румянцев, А. Шень. – М.: МЦНМО, 2011. – 80 с.

19 Сагалович, Ю. Введение в алгебраические коды / Ю. Л. Сагалович. – М.: МФТИ, 2007. – 262 с.
20 Арифметическое кодирование. – Дата обновления: 06.03.2018. URL: https://neerc.ifmo.ru/wiki/index.php?title= Арифметическое_кодирование (дата обращения: 20.04.2018).
21 Duda, J. Asymmetric numeral systems: entropy coding combining speed of Huffman coding with compression rate of arithmetic coding / Jarek Duda, Center for Science of Information, Purdue University, W.Lafayette, U.S.A., IN 47907. – arXiv:1311.2540v2 [cs.IT], 2014. – 24 p.
22 Duda, J. Asymmetric numeral systems / Jarek Duda, Jagiellonian University, Poland. – arXiv:0902.0271v5 [cs.IT], 2009. – 47 p.
23 Duda, J. The use of Asymmetric numeral systems as an accurate replacement of Huffman coding / J. Duda, K. Tahboub, N. J. Gadgil, E. J. Delp, Jagiellonian University, Poland. – 2015. – 5p.

Файл MyForm.h

```
#pragma once
#define _CRT_SECURE_NO_WARNINGS
#include <algorithm>
#include <cstdio>
#include <string>
#include <ctime>
#include <vector>
#include "ArifmCoding.h"
#include "flowrANS.h"
#include "Huffman.h"
#include "ReadInit.h"
#include "Shennon-Fano.h"
#include "Structures.h"

namespace Visualization {

	using namespace System;
	using namespace System::ComponentModel;
	using namespace System::Collections;
	using namespace System::Windows::Forms;
	using namespace System::Data;
	using namespace System::Drawing;
	using namespace System::Runtime::InteropServices;
	using namespace System::IO;
	using namespace System::Text;

	const int ALPHABET_SIZE = 256;
	public ref class MyForm : public System::Windows::Forms::Form
	{
	public:
		MyForm(void)
		{
			InitializeComponent();
		}

	protected:
		~MyForm()
		{
			if (components)
			{
				delete components;
			}
		}

	private: System::Windows::Forms::OpenFileDialog^  openFileDialog1;
	private: System::Windows::Forms::Button^  ChooseFile;
	private: System::Windows::Forms::Button^  rANS;

	private: System::Windows::Forms::Button^  ArifmCode;

	private: System::Windows::Forms::Button^  Huffman;
	private: System::Windows::Forms::Button^  ShennonFano;
	private: System::Windows::Forms::GroupBox^  groupBox1;
	private: System::Windows::Forms::GroupBox^  groupBox2;
```

```
        internal: System::Windows::Forms::GroupBox^  groupBox3;
        private: System::Windows::Forms::Button^  OpenOutput;
        internal:
        private:
        private: System::Windows::Forms::Button^  OpenInput;
        private: System::Windows::Forms::Button^  CheckFiles;
        internal:
        private: System::Windows::Forms::GroupBox^  groupBox4;
        private: System::Windows::Forms::TextBox^  textBox1;


        protected:

        private:
                char* StrToChar(String^ str) {
                        return (char*)(void*)Marshal::StringToHGlobalAnsi(str);
                }
                System::ComponentModel::Container ^components;

#pragma region Windows Form Designer generated code
                void InitializeComponent(void)
                {
                        System::ComponentModel::ComponentResourceManager^  resources = (gcnew
System::ComponentModel::ComponentResourceManager(MyForm::typeid));
                        this->openFileDialog1 = (gcnew System::Windows::Forms::OpenFileDialog());
                        this->ChooseFile = (gcnew System::Windows::Forms::Button());
                        this->rANS = (gcnew System::Windows::Forms::Button());
                        this->ArifmCode = (gcnew System::Windows::Forms::Button());
                        this->Huffman = (gcnew System::Windows::Forms::Button());
                        this->ShennonFano = (gcnew System::Windows::Forms::Button());
                        this->groupBox1 = (gcnew System::Windows::Forms::GroupBox());
                        this->groupBox2 = (gcnew System::Windows::Forms::GroupBox());
                        this->groupBox3 = (gcnew System::Windows::Forms::GroupBox());
                        this->CheckFiles = (gcnew System::Windows::Forms::Button());
                        this->OpenOutput = (gcnew System::Windows::Forms::Button());
                        this->OpenInput = (gcnew System::Windows::Forms::Button());
                        this->groupBox4 = (gcnew System::Windows::Forms::GroupBox());
                        this->textBox1 = (gcnew System::Windows::Forms::TextBox());
                        this->groupBox1->SuspendLayout();
                        this->groupBox2->SuspendLayout();
                        this->groupBox3->SuspendLayout();
                        this->groupBox4->SuspendLayout();
                        this->SuspendLayout();
                        //
                        // openFileDialog1
                        //
                        this->openFileDialog1->Filter = L"Text files (*.txt)|*.txt";
                        //
                        // ChooseFile
                        //
                        this->ChooseFile->Location = System::Drawing::Point(88, 30);
                        this->ChooseFile->Name = L"ChooseFile";
                        this->ChooseFile->Size = System::Drawing::Size(181, 23);
                        this->ChooseFile->TabIndex = 8;
                        this->ChooseFile->Text = L"Выбрать тесктовый файл";
                        this->ChooseFile->UseVisualStyleBackColor = true;
                        this->ChooseFile->Click += gcnew System::EventHandler(this,
&MyForm::ChooseFile_Click);
                        //
                        // rANS
                        //
                        this->rANS->Location = System::Drawing::Point(88, 151);
                        this->rANS->Name = L"rANS";
```

```
                    this->rANS->Size = System::Drawing::Size(181, 23);
                    this->rANS->TabIndex = 3;
                    this->rANS->Text = L"rANS";
                    this->rANS->UseVisualStyleBackColor = true;
                    this->rANS->Click += gcnew System::EventHandler(this,
&MyForm::rANS_Click);
                    //
                    // ArifmCode
                    //
                    this->ArifmCode->Location = System::Drawing::Point(88, 111);
                    this->ArifmCode->Name = L"ArifmCode";
                    this->ArifmCode->Size = System::Drawing::Size(181, 23);
                    this->ArifmCode->TabIndex = 2;
                    this->ArifmCode->Text = L"Арифметическое кодирование";
                    this->ArifmCode->UseVisualStyleBackColor = true;
                    this->ArifmCode->Click += gcnew System::EventHandler(this,
&MyForm::ArifmCode_Click);
                    //
                    // Huffman
                    //
                    this->Huffman->Location = System::Drawing::Point(88, 70);
                    this->Huffman->Name = L"Huffman";
                    this->Huffman->Size = System::Drawing::Size(181, 23);
                    this->Huffman->TabIndex = 1;
                    this->Huffman->Text = L"Алгоритм Хаффмана";
                    this->Huffman->UseVisualStyleBackColor = true;
                    this->Huffman->Click += gcnew System::EventHandler(this,
&MyForm::Huffman_Click);
                    //
                    // ShennonFano
                    //
                    this->ShennonFano->Location = System::Drawing::Point(88, 30);
                    this->ShennonFano->Name = L"ShennonFano";
                    this->ShennonFano->Size = System::Drawing::Size(181, 23);
                    this->ShennonFano->TabIndex = 0;
                    this->ShennonFano->Text = L"Алгоритм Шеннона-Фано";
                    this->ShennonFano->UseVisualStyleBackColor = true;
                    this->ShennonFano->Click += gcnew System::EventHandler(this,
&MyForm::ShennonFano_Click);
                    //
                    // groupBox1
                    //
                    this->groupBox1->Controls->Add(this->ShennonFano);
                    this->groupBox1->Controls->Add(this->Huffman);
                    this->groupBox1->Controls->Add(this->ArifmCode);
                    this->groupBox1->Controls->Add(this->rANS);
                    this->groupBox1->Location = System::Drawing::Point(12, 86);
                    this->groupBox1->Name = L"groupBox1";
                    this->groupBox1->Size = System::Drawing::Size(357, 195);
                    this->groupBox1->TabIndex = 9;
                    this->groupBox1->TabStop = false;
                    this->groupBox1->Text = L"Закодировать и декодировать методом";
                    //
                    // groupBox2
                    //
                    this->groupBox2->Controls->Add(this->ChooseFile);
                    this->groupBox2->Location = System::Drawing::Point(12, 12);
                    this->groupBox2->Name = L"groupBox2";
                    this->groupBox2->Size = System::Drawing::Size(357, 68);
                    this->groupBox2->TabIndex = 10;
                    this->groupBox2->TabStop = false;
                    this->groupBox2->Text = L"Выберите файл для кодирования и декодирования в
формате txt";
```

```
                    //
                    // groupBox3
                    //
                    this->groupBox3->Controls->Add(this->CheckFiles);
                    this->groupBox3->Controls->Add(this->OpenOutput);
                    this->groupBox3->Controls->Add(this->OpenInput);
                    this->groupBox3->Location = System::Drawing::Point(375, 176);
                    this->groupBox3->Name = L"groupBox3";
                    this->groupBox3->Size = System::Drawing::Size(412, 105);
                    this->groupBox3->TabIndex = 11;
                    this->groupBox3->TabStop = false;
                    this->groupBox3->Text = L"Проверить результат";
                    //
                    // CheckFiles
                    //
                    this->CheckFiles->Location = System::Drawing::Point(107, 69);
                    this->CheckFiles->Name = L"CheckFiles";
                    this->CheckFiles->Size = System::Drawing::Size(193, 23);
                    this->CheckFiles->TabIndex = 2;
                    this->CheckFiles->Text = L"Проверить файлы на совпадение";
                    this->CheckFiles->UseVisualStyleBackColor = true;
                    this->CheckFiles->Click += gcnew System::EventHandler(this,
&MyForm::CheckFiles_Click);
                    //
                    // OpenOutput
                    //
                    this->OpenOutput->Location = System::Drawing::Point(200, 31);
                    this->OpenOutput->Name = L"OpenOutput";
                    this->OpenOutput->Size = System::Drawing::Size(195, 23);
                    this->OpenOutput->TabIndex = 1;
                    this->OpenOutput->Text = L"Открыть закодированный файл";
                    this->OpenOutput->UseVisualStyleBackColor = true;
                    this->OpenOutput->Click += gcnew System::EventHandler(this,
&MyForm::OpenOutput_Click);
                    //
                    // OpenInput
                    //
                    this->OpenInput->Location = System::Drawing::Point(19, 31);
                    this->OpenInput->Name = L"OpenInput";
                    this->OpenInput->Size = System::Drawing::Size(165, 23);
                    this->OpenInput->TabIndex = 0;
                    this->OpenInput->Text = L"Открыть исходный файл";
                    this->OpenInput->UseVisualStyleBackColor = true;
                    this->OpenInput->Click += gcnew System::EventHandler(this,
&MyForm::OpenInput_Click);
                    //
                    // groupBox4
                    //
                    this->groupBox4->Controls->Add(this->textBox1);
                    this->groupBox4->Location = System::Drawing::Point(375, 12);
                    this->groupBox4->Name = L"groupBox4";
                    this->groupBox4->Size = System::Drawing::Size(412, 158);
                    this->groupBox4->TabIndex = 12;
                    this->groupBox4->TabStop = false;
                    this->groupBox4->Text = L"Характеристики метода";
                    //
                    // textBox1
                    //
                    this->textBox1->BackColor = System::Drawing::SystemColors::Window;
                    this->textBox1->Location = System::Drawing::Point(19, 21);
                    this->textBox1->Multiline = true;
                    this->textBox1->Name = L"textBox1";
                    this->textBox1->ReadOnly = true;
```

```
                    this->textBox1->Size = System::Drawing::Size(376, 121);
                    this->textBox1->TabIndex = 0;
                    //
                    // MyForm
                    //
                    this->AutoScaleDimensions = System::Drawing::SizeF(6, 13);
                    this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;
                    this->ClientSize = System::Drawing::Size(804, 292);
                    this->Controls->Add(this->groupBox4);
                    this->Controls->Add(this->groupBox3);
                    this->Controls->Add(this->groupBox2);
                    this->Controls->Add(this->groupBox1);
                    this->Icon = (cli::safe_cast<System::Drawing::Icon^>(resources-
>GetObject(L"$this.Icon")));
                    this->MaximumSize = System::Drawing::Size(820, 330);
                    this->MinimumSize = System::Drawing::Size(820, 330);
                    this->Name = L"MyForm";
                    this->Text = L"Демонстрация результатов";
                    this->groupBox1->ResumeLayout(false);
                    this->groupBox2->ResumeLayout(false);
                    this->groupBox3->ResumeLayout(false);
                    this->groupBox4->ResumeLayout(false);
                    this->groupBox4->PerformLayout();
                    this->ResumeLayout(false);

            }
#pragma endregion
        private: System::Void ChooseFile_Click(System::Object^  sender, System::EventArgs^  e) {
            Stream^ s;
            this->DialogResult = System::Windows::Forms::DialogResult::OK;
            if (this->openFileDialog1->ShowDialog() ==
System::Windows::Forms::DialogResult::OK) {
                    if (openFileDialog1->OpenFile() == nullptr)
                            MessageBox::Show("Ошибка открытия файла");
                    else
                    {
                            FILE *f_input = NULL, *f_output = NULL;
                            errno_t err;
                            err = fopen_s(&f_input, StrToChar(openFileDialog1->FileName),
"rt");
                            if (err != 0) {
                                    MessageBox::Show("Ошибка открытия файла");
                                    getchar();
                                    return;
                            }
                            err = fopen_s(&f_output, "save_copy.txt", "w");
                            if (err != 0) {
                                    MessageBox::Show("Ошибка открытия файла");
                                    getchar();
                                    return;
                            }
                            // Чтение исходного файла
                            unsigned char c1;
                            while (!feof(f_input)) {
                                    fscanf(f_input, "%c", &c1);
                                    fprintf(f_output, "%c", c1);
                            }
                            fclose(f_input);
                            fclose(f_output);
                    }
            }
        }
```

```
	private: System::Void ShennonFano_Click(System::Object^  sender, System::EventArgs^  e)
{
		if (openFileDialog1->FileName == "")
			MessageBox::Show("Выберите файл для кодирования");
		else {
			FILE *f_input = NULL, *f_output = NULL;
			string message = "";	// Текст, который необходимо закодировать
			int size = 0;	//Размер файла
			int f = ReadCoding(f_input, f_output, message, size,"save_copy.txt");
			if (f == 0)
			{
				MessageBox::Show("Ошибка открытия файла");
				return;
			}
			textBox1->Clear();
			symbol probs[ALPHABET_SIZE]; // Массив, представляющий символы, их
вероятности и коды
			int diff_alpha = 0; // Количество присутствующих в тексте различных
символов
			textBox1->Text = textBox1->Text + Initialization(message, size, probs,
diff_alpha, ALPHABET_SIZE);
			MessageBox::Show("Начинаем шифрование!");

			clock_t t_bg = clock();
			//Сортируем символы по убыванию вероятностей
			sort(probs, probs + ALPHABET_SIZE);
			//Выполнение кодирования алгоритмом Шеннона-Фано
			CodingShannon(probs, 0, diff_alpha - 1, size, ALPHABET_SIZE);
			//Кодирование исходного текста при помощи построенных кодов
			textBox1->Text = textBox1->Text + PackShannon(f_output, message, probs,
size, diff_alpha, ALPHABET_SIZE);
			//Декодирование
			_fcloseall();
			size = 0;  // Размер исходного файла
			int alpha = 0; // Размер алфавита закодированного текста
			f = ReadDecoding(f_input, f_output, alpha, size);
			unsigned char *char_table = new unsigned char[alpha]; //Таблица символов
			Vertex* root = new Vertex; //Корень дерева
								//Восстанавливаем дерево
Шеннона-Фано по кодам
			RebuildTreeShannon(f_input, root, alpha, char_table);

			//Декодируем исходное сообщение
			DecodeShannon(f_input, f_output, root, char_table);
			Clean_TreeShannon(root);
			delete(char_table);

			double t_end = ((double)clock() - t_bg) / CLOCKS_PER_SEC;
			textBox1->Text = textBox1->Text + "Time: " + t_end.ToString() +
Environment::NewLine;
			MessageBox::Show("Шифрование завершено!");
			fclose(f_input);
			fclose(f_output);
		}
	}
	private: System::Void Huffman_Click(System::Object^  sender, System::EventArgs^  e) {
		if (openFileDialog1->FileName == "")
			MessageBox::Show("Выберите файл для кодирования");
		else {
			FILE *f_input = NULL, *f_output = NULL;
			string message = "";	// Текст, который необходимо закодировать
			int size = 0;	//Размер файла
			int f = ReadCoding(f_input, f_output, message, size, "save_copy.txt");
```

```
                    if (f == 0)
                    {
                            MessageBox::Show("Ошибка открытия файла");
                            return;
                    }
                    textBox1->Clear();
                    symbol probs[ALPHABET_SIZE]; // Массив, представляющий символы, их
вероятности и коды
                    int diff_alpha = 0; // Количество присутствующих в тексте различных
символов
                    textBox1->Text = textBox1->Text + Initialization(message, size, probs,
diff_alpha, ALPHABET_SIZE);

                    MessageBox::Show("Начинаем шифрование!");

                    clock_t t_bg = clock();
                    //Выполнение кодирования алгоритмом Хаффмана
                    CodingHuffman(probs, diff_alpha, ALPHABET_SIZE);
                    //Кодирование исходного текста при помощи построенных кодов
                    textBox1->Text = textBox1->Text + PackHuffman(f_output, message, probs,
size, diff_alpha, ALPHABET_SIZE);

                    //Декодирование
                    _fcloseall();
                    size = 0;  // Размер исходного файла
                    int alpha = 0; // Размер алфавита закодированного текста
                    f = ReadDecoding(f_input, f_output, alpha, size);
                    unsigned char *char_table = new unsigned char[alpha]; //Таблица символов
                    Vertex* root = new Vertex; //Корень дерева
                                                                    //Восстанавливаем дерево
Хаффмана по кодам
                    RebuildTreeHuffman(f_input, root, alpha, char_table);
                    //Декодируем исходное сообщение
                    DecodeHuffman(f_input, f_output, root, char_table);

                    Clean_TreeHuffman(root);
                    delete(char_table);
                    double t_end = ((double)clock() - t_bg) / CLOCKS_PER_SEC;
                    textBox1->Text = textBox1->Text + "Time: " + t_end.ToString() +
Environment::NewLine;
                    MessageBox::Show("Шифрование завершено!");
                    fclose(f_input);
                    fclose(f_output);
            }
        }


        private: System::Void ArifmCode_Click(System::Object^  sender, System::EventArgs^  e) {
            if (openFileDialog1->FileName == "")
                    MessageBox::Show("Выберите файл для кодирования");
            else {
                    FILE *f_input = NULL, *f_output = NULL;
                    string message = "";      // Текст, который необходимо закодировать
                    int size = 0;    //Размер файла
                    int f = ReadCoding(f_input, f_output, message, size, "save_copy.txt");
                    if (f == 0)
                    {
                            MessageBox::Show("Ошибка открытия файла");
                            return;
                    }
                    textBox1->Clear();
                    symbol probs[ALPHABET_SIZE]; // Массив, представляющий символы, их
вероятности и коды
```

```
                    int diff_alpha = 0; // Количество присутствующих в тексте различных
символов
                    textBox1->Text = textBox1->Text + Initialization(message, size, probs,
diff_alpha, ALPHABET_SIZE);

                    MessageBox::Show("Начинаем шифрование!");

                    clock_t t_bg = clock();
                    //Сжатие исходного текста при помощи арифметического кодирования
                    textBox1->Text = textBox1->Text + PackArifm(f_output, message, probs,
size, diff_alpha, ALPHABET_SIZE);

                    //Декодирование
                    _fcloseall();
                    size = 0;  // Размер исходного файла
                    int alpha = 0; // Размер алфавита закодированного текста
                    f = ReadDecoding(f_input, f_output, alpha, size);
                    //Декодируем исходное сообщение
                    DecodeArifm(f_input, f_output, alpha, probs, ALPHABET_SIZE);
                    double t_end = ((double)clock() - t_bg) / CLOCKS_PER_SEC;
                    textBox1->Text = textBox1->Text + "Time: " + t_end.ToString() +
Environment::NewLine;
                    MessageBox::Show("Шифрование завершено!");
                    fclose(f_input);
                    fclose(f_output);
            }
      }
      private: System::Void rANS_Click(System::Object^  sender, System::EventArgs^  e) {
            if (openFileDialog1->FileName == "")
                    MessageBox::Show("Выберите файл для кодирования");
            else {
                    FILE *f_input = NULL, *f_output = NULL;
                    string message = "";      // Текст, который необходимо закодировать
                    int size = 0;    //Размер файла
                    int f = ReadCoding(f_input, f_output, message, size, "save_copy.txt");
                    if (f == 0)
                    {
                            MessageBox::Show("Ошибка открытия файла");
                            return;
                    }
                    textBox1->Clear();
                    symbol probs[ALPHABET_SIZE]; // Массив, представляющий символы, их
вероятности и коды
                    int diff_alpha = 0; // Количество присутствующих в тексте различных
символов
                    textBox1->Text = textBox1->Text + Initialization(message, size, probs,
diff_alpha, ALPHABET_SIZE);

                    MessageBox::Show("Начинаем шифрование!");

                    clock_t t_bg = clock();
                    //Сжатие исходного текста при помощи rANS
                    textBox1->Text = textBox1->Text + PackFlowrANS(f_output, message, probs,
diff_alpha, size, ALPHABET_SIZE);

                    //Декодирование
                    _fcloseall();
                    size = 0;  // Размер исходного файла
                    int alpha = 0; // Размер алфавита закодированного текста
                    f = ReadDecoding(f_input, f_output, alpha, size);
                    //Декодируем исходное сообщение
                    DecoderFlowANS(f_input, f_output, alpha, size, ALPHABET_SIZE);
                    double t_end = ((double)clock() - t_bg) / CLOCKS_PER_SEC;
```

```
                    textBox1->Text = textBox1->Text + "Time: " + t_end.ToString() +
Environment::NewLine;
                    MessageBox::Show("Шифрование завершено!");
                    fclose(f_input);
                    fclose(f_output);
            }
        }

        private: System::Void CheckFiles_Click(System::Object^  sender, System::EventArgs^  e) {
            if (openFileDialog1->FileName == "")
                    MessageBox::Show("Выберите файл для кодирования");
            else {
                    FILE *f_input = NULL, *f_output = NULL;
                    errno_t err;
                    err = fopen_s(&f_input, "save_copy.txt", "rt");
                    if (err != 0) {
                            MessageBox::Show("Ошибка открытия файла");
                            getchar();
                            return ;
                    }
                    err = fopen_s(&f_output, "output_decoding.txt", "rt");
                    if (err != 0) {
                            MessageBox::Show("Ошибка открытия файла");
                            getchar();
                            return ;
                    }
                    // Чтение исходного файла
                    unsigned char c1,c2;
                    while ((!feof(f_input))&& (!feof(f_output))) {
                            fscanf(f_input, "%c", &c1);
                            fscanf(f_output, "%c", &c2);
                            if (feof(f_output) || feof(f_input))
                            {
                                    if (feof(f_output) != feof(f_input))
                                    {
                                            fclose(f_input);
                                            fclose(f_output);
                                            MessageBox::Show("Файлы разные");
                                            return;
                                    }
                            }
                            if (c1 != c2)
                            {
                                    fclose(f_input);
                                    fclose(f_output);
                                    MessageBox::Show("Файлы разные");
                                    return;
                            }
                    }
                    fclose(f_input);
                    fclose(f_output);
                    MessageBox::Show("Файлы одинаковые");
            }
        }
        private: System::Void OpenInput_Click(System::Object^  sender, System::EventArgs^  e) {
            if (openFileDialog1->FileName == "")
                    MessageBox::Show("Выберите файл для кодирования");
            else {
                    system("save_copy.txt");
            }
        }
        private: System::Void OpenOutput_Click(System::Object^  sender, System::EventArgs^  e) {
            if (openFileDialog1->FileName == "")
```

```
                MessageBox::Show("Выберите файл для кодирования");
            else {
                system("output_decoding.txt");
            }
        }
        };
}
```

Файл MyForm.cpp

```
#include <io.h>
#include <stdio.h>
#include <fcntl.h>
#include <share.h>
#include <stdlib.h>
#include <vcclr.h>
#include <algorithm>
#include <cstdio>
#include <string>
#include <ctime>
#include <vector>
#include "MyForm.h"

using namespace System;
using namespace System::Windows::Forms;

[STAThreadAttribute]
void Main(array<String^>^ args) {
        Application::EnableVisualStyles();
        Application::SetCompatibleTextRenderingDefault(false);
        Visualization::MyForm form;
        Application::Run(%form);
}
```

Файл Structures.h

```
#pragma once
#ifndef __STRUCTURES_H__
#define __STRUCTURES_H__

#include <vector>
#include "LongArifm.h"
using namespace std;

// Структура для символа для методов Хаффмана и Шеннона-Фано
struct symbol {
        unsigned char ch;           // Символ
        int prob;          // Сколько раз символ встретился в исходном тексте
        vector<unsigned char> code; // Код
        symbol() { ch = ' '; prob = 0; }
        ~symbol() { code.clear(); }
};

bool operator<(const symbol &s1, const symbol &s2) {
        return (s1.prob > s2.prob);
}

//Структура для дерева Шеннона-Фано и Хаффмана
struct Vertex {
        int flag;             // Является ли вершина терминальной
        Vertex *Left, *Right; // Указатели на левого и правого сына
```

```
	Vertex() { flag = -1; Left = Right = NULL; }
};

//Структура для дерева Хаффмана
struct elem_tree {
	unsigned char ch;		// Символ
	int prob;		// Сколько раз символ встретился в исходном тексте
	int pos;		// Индекс вершины в дереве
	int pos_left, pos_right; // Индексы левого и правого сына в дереве
	elem_tree() { pos = pos_left = pos_right = -1; ch = ' '; prob = 0; }
	elem_tree(unsigned char ch, int prob, int p1, int p2, int p3) :
		ch(ch), prob(prob), pos(p1), pos_left(p2), pos_right(p3) {}
};

bool operator<(const elem_tree &a, const elem_tree &b) {
	return (a.prob < b.prob || (a.prob == b.prob && a.pos < b.pos));
}

//Струткура для хранения дроби
struct fraction {
	tlong num, denom;
	fraction() { num = 0; denom = 1; }
	fraction(const tlong &x1, const tlong &x2) {
		num = x1;
		denom = x2;
		tlong c = gcd(num, denom);
		num /= c;
		denom /= c;
	}
};

bool operator >= (const fraction &f1, const fraction &f2) {
	return ((f1.num * f2.denom) >= (f1.denom * f2.num));
}

bool operator <= (const fraction &f1, const fraction &f2) {
	return ((f1.num * f2.denom) <= (f1.denom * f2.num));
}

bool operator < (const fraction &f1, const fraction &f2) {
	return ((f1.num * f2.denom) < (f1.denom * f2.num));
}

bool operator > (const fraction &f1, const fraction &f2) {
	return ((f1.num * f2.denom) > (f1.denom * f2.num));
}

fraction operator + (const fraction &f1, const fraction &f2) {
	return fraction(f1.num * f2.denom + f2.num * f1.denom, f1.denom * f2.denom);
}

fraction operator - (const fraction &f1, const fraction &f2) {
	return fraction(f1.num * f2.denom - f2.num * f1.denom, f1.denom * f2.denom);
}

fraction operator * (const fraction &f1, const fraction &f2) {
	return fraction(f1.num * f2.num, f1.denom * f2.denom);
}

fraction operator / (const fraction &f1, const fraction &f2) {
	return fraction(f1.num * f2.denom, f2.num * f1.denom);
}
```

```
//Структура, задающая подотрезок отрезка [0;1), соответсвтующего конкретному символу на основе частотного анализа
struct Segment {
	fraction Left, Right;
	char ch;
	Segment() {}
};

inline int CountBitNum(int x) {
	int ans = 0;
	while (x > 0) {
		ans++;
		x /= 10;
	}
	return ans;
}

int CountAllBitsProbs(int cnt_alpha, symbol* a, int size) {
	int ans = CountBitNum(size) + 1;
	ans += CountBitNum(cnt_alpha) + 1;
	for (int i = 0; i < cnt_alpha; i++) {
		ans++;
		if (a[i].ch != '\n')
			ans++;
		ans += CountBitNum(a[i].prob) + 1;
	}
	return ans;
}

int CountAllBitsCodes(int cnt_alpha, symbol* a, int size) {
	int ans = CountBitNum(size) + 1;
	ans += CountBitNum(cnt_alpha) + 1;
	for (int i = 0; i < cnt_alpha; i++) {
		ans++;
		if (a[i].ch != '\n')
			ans++;
		ans += (int)a[i].code.size() + 1;
	}
	return ans;
}
    #endif
```

Файл ReadInit.h

```
#pragma once
#ifndef __READ_WRITE_H__
#define __READ_WRITE_H__

#include <cstdio>
using namespace std;
using namespace System;

int ReadCoding(FILE* &f_input, FILE* &f_output, string &message, int &size,const char * filename) {
	// Файлы с исходным текстом и закодированным текстом
	errno_t err;
	err = fopen_s(&f_input, filename, "rt");
	if (err != 0) {
		getchar();
		return 0;
	}
	err = fopen_s(&f_output, "output_coding.txt", "w");
	if (err != 0) {
```

```
		getchar();
		return 0;
	}
	// Чтение исходного файла
	unsigned char c;
	while (!feof(f_input)) {
		fscanf(f_input, "%c", &c);
		if (feof(f_input))
			break;
		message += c;
		size++;
	}
	return 1;
}

int ReadNum(FILE* &f_input) {
	int ans = 0;
	unsigned char c;
	fscanf(f_input, "%c", &c);
	while (c >= '0' && c <= '9') {
		ans = ans * 10 + c - '0';
		fscanf(f_input, "%c", &c);
	}
	return ans;
}

int ReadDecoding(FILE* &f_input, FILE* &f_output, int &alpha, int &size) {
	// Файлы с исходным текстом и закодированным текстом
	errno_t err;
	err = fopen_s(&f_input, "output_coding.txt", "rt");
	if (err != 0) {
		getchar();
		return 0;
	}
	err = fopen_s(&f_output, "output_decoding.txt", "w");
	if (err != 0) {
		getchar();
		return 0;
	}
	unsigned char c;
	size = ReadNum(f_input);
	alpha = ReadNum(f_input);
	return 1;
}

String^ Initialization(string &message, int &size, symbol* probs, int &diff_alpha, const int
ALPHABET_SIZE) {
	//Инициализация массива символов
	for (int i = 0; i < ALPHABET_SIZE; i++) {
		probs[i].prob = 0;
		probs[i].ch = (unsigned char)i;
	}
	//Подсчёт вероятности
	for (int i = 0; i < size; i++)
		probs[unsigned char(message[i])].prob++;

	for (int i = 0; i < ALPHABET_SIZE; i++)
		if (probs[i].prob != 0)
			diff_alpha++;
	//Подсчет энтропии
	double entropy = 0.0;
	for (int i = 0; i < ALPHABET_SIZE; i++)
		if (probs[i].prob != 0) {
```

```
                        double p = (double)probs[i].prob / (double)size;
                        entropy -= p * log(p) / log(2.0);
                }

        String^ answer;
        answer = "Entropy: " + entropy.ToString() + Environment::NewLine;
        answer = answer + "Alphabet size: " + diff_alpha.ToString() + Environment::NewLine;
        return answer;
}

#endif
```

Файл Shennon-Fano.h

```
#pragma once
#ifndef __SHENNON_FANO_H__
#define __SHENNON_FANO_H__

#include <cstdio>
#include "Structures.h"
using namespace std;
using namespace System;

void CodingShannon(symbol* a, int left, int right, int sum, const int ALPHABET_SIZE) {
        if (left >= right) return;
        int sum_left = 0;    // Сумма элементов в левой части таблицы
        int merge_c = 0;     // Номер крайнего правого элемента левой части таблицы
        int merge_prob = 0; // Сумма вероятностей в левой части таблицы
                                        //Рассматриваем все вероятности
        for (int i = left; i <= right; i++) {
                sum_left += a[i].prob;
                if (a[i].prob > ((sum + 1) >> 1)) {
                        a[i].code.push_back('0');
                        merge_c = i;
                        merge_prob = a[i].prob;
                        continue;
                }
                if (sum_left <= ((sum + 1) >> 1)) {
                        a[i].code.push_back('0');
                        merge_prob = sum_left;
                        merge_c = i;
                }
                else
                        a[i].code.push_back('1');
        }

        CodingShannon(a, left, merge_c, merge_prob, ALPHABET_SIZE);
        CodingShannon(a, merge_c + 1, right, sum - merge_prob, ALPHABET_SIZE);
}

String^ PackShannon(FILE* f, const string &message, symbol* a, int size, int cnt_alpha, const
int ALPHABET_SIZE) {
        // Создание массива для быстрого доступа к символам
        int *alpha_positions = new int[ALPHABET_SIZE];
        for (int i = 0; i < ALPHABET_SIZE; i++)
                alpha_positions[a[i].ch] = i;

        int table_size = CountAllBitsCodes(cnt_alpha, a, size);
        fprintf(f, "%d\n", size);
        //Вывод таблицы кодов символов
        fprintf(f, "%d\n", cnt_alpha);
        for (int i = 0; i < cnt_alpha; i++) {
```

```
		fprintf(f, "%c", a[i].ch);
		if (a[i].ch != '\n')
			fprintf_s(f, "\n");
		for (int j = 0; j < (int)a[i].code.size(); j++)
			fprintf_s(f, "%c", a[i].code[j]);
		fprintf_s(f, "\n");
	}

	double aver = 0.0;
	// Кодирование сообщения
	for (int i = 0; i < size; i++) {
		int pos = alpha_positions[unsigned char(message[i])];
		aver += a[pos].code.size();
		for (int j = 0; j < (int)a[pos].code.size(); j++)
			fprintf_s(f, "%c", a[pos].code[j]);
	}
	delete alpha_positions;
	String^ answer;
	answer = "Size of coding table (bytes): " + table_size.ToString() +
Environment::NewLine;
	answer = answer + "Average quantity of bits per symbol (only code): " + (aver /
((double)size)).ToString() + Environment::NewLine;
	answer = answer + "Average quantity of bits per symbol with coding table: " +
		((aver + (double)table_size * 8.0) / ((double)size)).ToString() +
Environment::NewLine;
	return answer;
}

void RebuildTreeShannon(FILE* f, Vertex* &root, int alpha, unsigned char* &char_table) {
	int cnt = 0;
	for (int q = 0; q < alpha; q++) {
		Vertex *tree = root;
		unsigned char c, f_c;
		fscanf(f, "%c\n", &f_c);
		fscanf(f, "%c", &c);
		while (c != '\n') {
			if (c == '0') {
				if (tree->Left == NULL)
					tree->Left = new Vertex;
				tree = tree->Left;
			}
			else if (c == '1') {
				if (tree->Right == NULL)
					tree->Right = new Vertex;
				tree = tree->Right;
			}
			fscanf(f, "%c", &c);
		}
		tree->flag = cnt;
		char_table[cnt++] = f_c;
	}
}

void DecodeShannon(FILE *f_in, FILE *f_out, Vertex* &root, unsigned char *char_table) {
	Vertex *tree = root;
	unsigned char c;
	while (!feof(f_in)) {
		if (tree->flag != -1) {
			fprintf(f_out, "%c", char_table[tree->flag]);
			tree = root;
		}
		else {
			fscanf(f_in, "%c", &c);
```

```
                        if (c == '0')
                                tree = tree->Left;
                        else
                                tree = tree->Right;
                }
        }
}

void Clean_TreeShannon(Vertex* &root) {
        if (root->Left != NULL) Clean_TreeShannon(root->Left);
        if (root->Right != NULL) Clean_TreeShannon(root->Right);
        delete root;
}

#endif
```

Файл Huffman.h

```
#pragma once
#ifndef __HUFFMAN_H__
#define __HUFFMAN_H__

#include <set>
#include <cstdio>
#include <vector>
#include "Structures.h"
using namespace std;
using namespace System;

void dfsHuffman(symbol* a, elem_tree *tree, int *alph, int v, vector<unsigned char> &x) {
        if (tree[v].pos_left == -1 && tree[v].pos_right == -1) {
                int pos = alph[tree[v].ch];
                a[pos].code = x;
                return;
        }
        if (tree[v].pos_left != -1) {
                x.push_back('0');
                dfsHuffman(a, tree, alph, tree[v].pos_left, x);
                x.pop_back();
        }
        if (tree[v].pos_right != -1) {
                x.push_back('1');
                dfsHuffman(a, tree, alph, tree[v].pos_right, x);
                x.pop_back();
        }
}

void CodingHuffman(symbol* a, int cnt_alpha, const int ALPHABET_SIZE) {
        // Создание массива для быстрого доступа к символам
        int *alpha_positions = new int[ALPHABET_SIZE];
        for (int i = 0; i < ALPHABET_SIZE; i++)
                alpha_positions[a[i].ch] = i;

        //Построение дерева Хаффмана
        set<elem_tree> st;
        int uk_tree = 0;
        elem_tree *tree = new elem_tree[2 * cnt_alpha];
        for (int i = 0; i < ALPHABET_SIZE; i++)
                if (a[i].prob != 0) {
                        tree[uk_tree] = elem_tree(a[i].ch, a[i].prob, uk_tree, -1, -1);
                        st.insert(tree[uk_tree]);
                        uk_tree++;
```

```
		}

	while ((int)st.size() > 1) {
		set<elem_tree>::iterator it1 = st.begin();
		elem_tree x1 = *it1;
		st.erase(it1);
		set<elem_tree>::iterator it2 = st.begin();
		elem_tree x2 = *it2;
		st.erase(it2);
		elem_tree new_elem = elem_tree(x1.ch, x1.prob + x2.prob, uk_tree, x1.pos,
x2.pos);
		tree[uk_tree++] = new_elem;
		st.insert(new_elem);
	}
	uk_tree--;

	// Построение кодов символов по дереву Хаффмана
	vector<unsigned char> x;
	dfsHuffman(a, tree, alpha_positions, uk_tree, x);
	delete tree;
	delete alpha_positions;
}

String^ PackHuffman(FILE* f, const string &message, symbol* a, int size, int cnt_alpha, const
int ALPHABET_SIZE) {
	int table_size = CountAllBitsCodes(cnt_alpha, a, size);
	fprintf(f, "%d\n", size);
	//Вывод таблицы кодов символов
	fprintf(f, "%d\n", cnt_alpha);
	for (int i = 0; i < ALPHABET_SIZE; i++)
		if (a[i].prob != 0) {
			fprintf(f, "%c", a[i].ch);
			if (a[i].ch != '\n')
				fprintf_s(f, "\n");
			for (int j = 0; j < (int)a[i].code.size(); j++)
				fprintf_s(f, "%c", a[i].code[j]);
			fprintf_s(f, "\n");
		}

	double aver = 0.0;
	// Кодирование сообщения
	for (int i = 0; i < size; i++) {
		int pos = unsigned char(message[i]);
		aver += a[pos].code.size();
		for (int j = 0; j < (int)a[pos].code.size(); j++)
			fprintf_s(f, "%c", a[pos].code[j]);
	}
	String^ answer;
	answer = "Size of coding table (bytes): " + table_size.ToString() +
Environment::NewLine;
	answer = answer + "Average quantity of bits per symbol (only code): " + (aver /
((double)size)).ToString() + Environment::NewLine;
	answer = answer + "Average quantity of bits per symbol with coding table: " +
		((aver + (double)table_size * 8.0) / ((double)size)).ToString() +
Environment::NewLine;
	return answer;
}

void RebuildTreeHuffman(FILE* f, Vertex* &root, int alpha, unsigned char* &char_table) {
	int cnt = 0;
	for (int q = 0; q < alpha; q++) {
		Vertex *tree = root;
		unsigned char c, f_c;
```

```
		fscanf(f, "%c\n", &f_c);
		fscanf(f, "%c", &c);
		while (c != '\n') {
			if (c == '0') {
				if (tree->Left == NULL)
					tree->Left = new Vertex;
				tree = tree->Left;
			}
			else if (c == '1') {
				if (tree->Right == NULL)
					tree->Right = new Vertex;
				tree = tree->Right;
			}
			fscanf(f, "%c", &c);
		}
		tree->flag = cnt;
		char_table[cnt++] = f_c;
	}
}

void DecodeHuffman(FILE *f_in, FILE *f_out, Vertex* &root, unsigned char *char_table) {
	Vertex *tree = root;
	unsigned char c;
	while (!feof(f_in)) {
		if (tree->flag != -1) {
			fprintf(f_out, "%c", char_table[tree->flag]);
			tree = root;
		}
		else {
			fscanf(f_in, "%c", &c);
			if (c == '0')
				tree = tree->Left;
			else
				tree = tree->Right;
		}
	}
}

void Clean_TreeHuffman(Vertex* &root) {
	if (root->Left != NULL) Clean_TreeHuffman(root->Left);
	if (root->Right != NULL) Clean_TreeHuffman(root->Right);
	delete root;
}

#endif
```

Файл LongArifm.h

```
#pragma once
#ifndef __LONG_CODING_H__
#define __LONG_CODING_H__

#include <algorithm>
#include <vector>
using namespace std;
using namespace System;
typedef long long ll;
const int LONG_BASE = 1000000000;		// Наибольшее число в разряде плюс один
const int BASE_DEG = 9;			// Храним по 9 цифр

struct tlong {
    int len;
```

```
vector<int> s;
tlong():len(1) {
    s.resize(2);
    s[1] = 0;
}
tlong(const tlong& t) {
    len = t.len;
    s.resize(len + 1);
    s = t.s;
}
// x < LONG_BASE
tlong(int x) {
    len = 1;
    s.resize(2);
    s[1] = x;
}
   ~tlong() {}

tlong& trim() {
    while (len > 1 && s[len] == 0)
        len--;
    s.resize(len + 1);
    return *this;
}

void resize(int newlen) {
    len = newlen;
    //Индексация с "1"
    s.resize(newlen + 1);
}

tlong& operator = (const tlong &x) {
    resize(x.len);
    s = x.s;
    return *this;
}

tlong& operator = (ll x) {   // x < LONG_BASE * LONG_BASE
    resize(2);
    s[1] = (int)(x % LONG_BASE);
    s[2] = (int)(x / LONG_BASE);
    if (s[2] == 0) {
        len--;
        s.pop_back();
    }
    return *this;
}
// x < LONG_BASE
tlong& operator = (int x) {
    resize(1);
    s[1] = x;
    return *this;
}

tlong& operator = (const char *st) {
    int l = (int)strlen(st);
    resize(l / BASE_DEG + 1);
    s[len] = 0;
    for (int i = 0; i < l % BASE_DEG; i++)
        s[len] = s[len] * 10 + (st[i] - '0');
    for (int i = 1; i <= l / BASE_DEG; i++) {
        s[len - i] = 0;
        for (int j = (i - 1) * BASE_DEG; j < i * BASE_DEG; j++)
```

```
                s[len - i] = s[len - i] * 10 + (st[j + l % BASE_DEG] - '0');
        }
        trim();
        return *this;
    }
};

tlong const lzero;
tlong const lone = 1;

int compare(const tlong &a, const tlong &b, int sp = 0) {
    if (!b.s[b.len] + !a.s[a.len])
        return !b.s[b.len] - !a.s[a.len];
    if (a.len > b.len + sp)
        return 1;
    if (a.len < b.len + sp)
        return -1;
    for (long i = a.len; i > sp; i--)
        if (a.s[i] > b.s[i - sp])
            return 1;
        else if (a.s[i] < b.s[i - sp])
            return -1;
    for (long i = sp; i >= 1; i--)
        if (a.s[i] > 0) return 1;
    return 0;
}

inline bool operator < (const tlong& a, const tlong &b) {
    return compare(a, b) < 0;
}

inline bool operator > (const tlong& a, const tlong &b) {
    return compare(a, b) > 0;
}

inline bool operator == (const tlong& a, const tlong &b) {
    return compare(a, b) == 0;
}

inline bool operator != (const tlong &a, const tlong &b) {
    return (!(compare(a, b) == 0));
}

inline bool operator >= (const tlong &a, const tlong &b) {
    return (!(compare(a, b) < 0));
}

inline bool operator <= (const tlong &a, const tlong &b) {
    return (!(compare(a, b) == 1));
}

void long_add(const tlong &a, const tlong &b, tlong &c, ll p = 0) {
    int a_len = a.len, b_len = b.len;
    int k = min(a_len, b_len);
    c.resize(max(a_len, b_len) + 1);
    for (int i = 1; i <= k; i++) {
        p += a.s[i] + b.s[i];
        c.s[i] = (int)(p % LONG_BASE);
        p /= LONG_BASE;
    }
    for (int i = k + 1; i <= a_len; i++) {
        p += a.s[i];
        c.s[i] = (int)(p % LONG_BASE);
```

```
        p /= LONG_BASE;
    }
    for (int i = k + 1; i <= b_len; i++) {
        p += b.s[i];
        c.s[i] = (int)(p % LONG_BASE);
        p /= LONG_BASE;
    }
    c.s[c.len] = (int)p;
    c.trim();
}

// Нормирует ненормированные величины в разрядах после *+
inline void lnorm(tlong& a) {
    long_add(a, lzero, a);
}

tlong operator + (const tlong &a, const tlong &b) {
    tlong c;
    long_add(a, b, c);
    return c;
}

// b < LONG_BASE
tlong operator + (const tlong &a, int b) {
    tlong c;
    long_add(a, lzero, c, b);
    return c;
}

inline tlong& operator += (tlong& a, const tlong& b) {
    tlong c = a;
    long_add(c, b, a);
    return a;
}

// b < LONG_BASE
inline tlong& operator += (tlong& a, int b) {
    tlong c = a;
    long_add(c, lzero, a, b);
    return a;
}

void long_sub(const tlong &a, const tlong &b, tlong &c, int sp = 0) {
    int z = 0, i;
    c.resize(a.len);
    for (i = sp + 1; i <= a.len; i++) {
        if (i - sp > b.len)
            c.s[i] = a.s[i] - z;
        else
            c.s[i] = a.s[i] - b.s[i - sp] - z;
        z = 0;
        if (c.s[i] < 0) {
            z = 1;
            c.s[i] += LONG_BASE;
        }
    }
    c.trim();
}

// b < LONG_BASE
void long_sub(const tlong &a, int b, tlong &c, int sp = 0) {
    c = a;
    int z = 0, i;
```

```
    for (i = sp + 1; i <= a.len; i++) {
        if (i - sp > 1)
            c.s[i] -= z;
        else
            c.s[i] -= b + z;
        z = 0;
        if (c.s[i] < 0) {
            z = 1;
            c.s[i] += LONG_BASE;
        }
    }
    c.trim();
}

tlong operator - (const tlong &a, const tlong &b) {
    tlong c;
    long_sub(a, b, c);
    return c;
}

// b < LONG_BASE
tlong operator - (const tlong &a, int b) {
    tlong c;
    long_sub(a, b, c);
    return c;
}

inline tlong& operator -= (tlong& a, const tlong& b) {
    tlong c = a;
    long_sub(c, b, a);
    return a;
}

// b < LONG_BASE
inline tlong& operator -= (tlong& a, int b) {
    tlong c = a;
    long_sub(c, b, a);
    return a;
}

void long_mul_add(const tlong &a, const tlong &b, tlong &c) {
    int a_len = a.len, b_len = b.len, c_len = c.len;
    c.resize(max(c_len, a_len + b_len + 1));
    ll p;
    for (int i = 1; i <= a_len; i++) {
        p = 0;
        for (int j = 1; j <= b_len; j++) {
            p += c.s[i + j - 1] + (ll)a.s[i] * (ll)b.s[j];
            c.s[i + j - 1] = (int)(p % LONG_BASE);
            p /= LONG_BASE;
        }
        c.s[i + b_len] += (int)p;
    }
    lnorm(c);
}

inline void long_mul(const tlong &a, tlong b, tlong &c) {
    c = lzero;
    long_mul_add(a, b, c);
}

// k < LONG_BASE
void long_mul(const tlong &a, int k, tlong &c) {
```

```
    int a_len = a.len, c_len = a_len + 1;
    ll p = 0;
    c.resize(a_len + 1);
    for (int i = 1; i <= a_len; i++) {
        p += (ll)a.s[i] * (ll)k;
        c.s[i] = (int)(p % LONG_BASE);
        p /= LONG_BASE;
    }
    c.s[c_len] = (int)p;
    c.trim();
}

tlong operator * (const tlong &a, const tlong &b) {
    tlong c;
    long_mul(a, b, c);
    return c;
}

// b < LONG_BASE
tlong operator * (const tlong &a, int b) {
    tlong c;
    long_mul(a, b, c);
    return c;
}

inline tlong& operator *= (tlong& a, const tlong& b) {
    tlong c = a;
    long_mul(c, b, a);
    return a;
}

// b < LONG_BASE
inline tlong& operator *= (tlong& a, int b) {
    tlong c = a;
    long_mul(c, b, a);
    return a;
}

// k < LONG_BASE
void long_div(const tlong &a, int k, tlong &q, int &rem) {
    ll p = 0, dd;
    q.resize(a.len);
    for (int i = q.len; i >= 1; i--) {
        dd = p * LONG_BASE + (ll)a.s[i];
        q.s[i] = (int)(dd / (ll)k);
        p = dd % k;
    }
    rem = (int)p;
    q.trim();
}

int long_div_find(tlong &ost, const tlong &b, int sp) {
    int up, down, mid;
    tlong c;
    down = 0;
    up = LONG_BASE;
    while (up > down) {
        mid = (up + down) / 2;
        long_mul(b, mid, c);
        if (compare(ost, c, sp) < 0)
            up = mid;
        else
            down = mid + 1;
```

```
    }
    down--;
    long_mul(b, down, c);
    long_sub(ost, c, ost, sp);
    return down;
}

void long_div_make(const tlong &a, const tlong &b, tlong &q, tlong &ost) {
    int sp = a.len - b.len;
    ost = a;
    if (compare(a, b, sp) == -1)
        sp--;
    q.resize(sp + 1);
    while (sp >= 0) {
        q.s[sp + 1] = long_div_find(ost, b, sp);
        sp--;
    }
}

void long_div(const tlong &a, const tlong &b, tlong &q, tlong &ost) {
    q = 0;
    ost = 0;
    switch (compare(a,b)) {
        case 1: long_div_make(a, b, q, ost);
                break;
        case -1: ost = a;
                 break;
        case 0: q = 1;
                break;
    }
}

tlong operator / (const tlong &a, const tlong &b) {
    tlong c, ost;
    long_div(a, b, c, ost);
    return c;
}

tlong operator % (const tlong &a, const tlong &b) {
    tlong c, ost;
    long_div(a, b, c, ost);
    return ost;
}

inline tlong& operator /= (tlong& a, const tlong& b) {
    tlong q, rem;
    long_div(a, b, q, rem);
    return a = q;
}

inline tlong& operator %= (tlong& a, const tlong& b) {
    tlong q, rem;
    long_div(a, b, q, rem);
    return a = rem;
}

tlong gcd(tlong a, tlong b) {
    tlong res, ost, c;
    while (a.s[a.len] && b.s[b.len])
        if (compare(a, b, 0) >= 0) {
            long_div(a, b, res, ost);
            a = ost;
        }
```

```
        else {
            long_div(b, a, res, ost);
            b = ost;
        }
    long_add(a, b, c);
    return c;
}
#endif
```

Файл ArifmCoding.h

```
#pragma once
#ifndef __ARIFM_CODING_H__
#define __ARIFM_CODING_H__

#include <cstdio>
#include "LongArifm.h"
#include "Structures.h"
using namespace std;
using namespace System;
typedef long long ll;

String^ PackArifm(FILE* f, const string &message, symbol * a, int size, int cnt_alpha, const
int ALPHABET_SIZE) {
	int table_size = CountAllBitsProbs(cnt_alpha, a, size);
	fprintf(f, "%d\n", size);
	//Вывод таблицы кодов символов
	fprintf(f, "%d\n", cnt_alpha);
	for (int i = 0; i < ALPHABET_SIZE; i++)
		if (a[i].prob != 0) {
			fprintf(f, "%c", a[i].ch);
			if (a[i].ch != '\n')
				fprintf_s(f, "\n");
			fprintf_s(f, "%d\n", a[i].prob);
		}
	//Инициализация отрезков на основе частотного анализа
	Segment* segm = new Segment[ALPHABET_SIZE];
	fraction st = fraction(0, 1);
	for (int i = 0; i < ALPHABET_SIZE; i++)
		if (a[i].prob != 0) {
			segm[i].Left = st;
			segm[i].Right = st + fraction(a[i].prob, size);
			st = segm[i].Right;
		}

	//Процесс кодирования
	fraction left = fraction(0, 1);
	fraction right = fraction(1, 1);
	for (int i = 0; i < size; i++) {
		int pos = unsigned char(message[i]);
		fraction newLeft = left + (right - left) * segm[pos].Left;
		fraction newRight = left + (right - left) * segm[pos].Right;
		left = newLeft;
		right = newRight;
	}
	//Процесс получения кода по отрезку
	vector<int> ans;
	while (true) {
		tlong c1 = (left.num * 2) / left.denom;
		left.num = (left.num * 2) % left.denom;
		tlong c2 = (right.num * 2) / right.denom;
		right.num = (right.num * 2) % right.denom;
```

```
		if (c1.s[1] == c2.s[1])
			ans.push_back((int)c1.s[1]);
		else {
			ans.push_back(1);
			break;
		}
	}
	for (int i = 0; i < (int)ans.size(); i++)
		fprintf_s(f, "%d", ans[i]);
	double aver = (double)ans.size();	// Среднее количество бит на символ
	String^ answer;
	answer = "Size of coding table (bytes): " + table_size.ToString() +
Environment::NewLine;
	answer = answer +"Average quantity of bits per symbol (only code): " + (aver /
((double)size)).ToString() + Environment::NewLine;
	answer = answer + "Average quantity of bits per symbol with coding table: " +
		((aver + (double)table_size * 8.0) / ((double)size)).ToString() +
Environment::NewLine;
	return answer;
}

void DecodeArifm(FILE *f_in, FILE *f_out, int alpha, symbol* a, const int ALPHABET_SIZE) {
	//Считываем таблицу вероятностей
	for (int q = 0; q < alpha; q++) {
		int d = 0;
		unsigned char c, f_c;
		fscanf(f_in, "%c", &f_c);
		if (f_c != '\n')
			fscanf(f_in, "\n");
		fscanf(f_in, "%c", &c);
		while (c >= '0' && c <= '9') {
			d = d * 10 + c - '0';
			fscanf(f_in, "%c", &c);
		}
		a[f_c].ch = f_c;
		a[f_c].prob = d;
	}
	//Инициализация отрезков на основе частотного анализа
	int size = 0;
	for (int i = 0; i < ALPHABET_SIZE; i++)
		size += a[i].prob;
	Segment* segm = new Segment[ALPHABET_SIZE];
	int uk = 0;
	fraction st = fraction(0, 1);
	for (int i = 0; i < ALPHABET_SIZE; i++)
		if (a[i].prob != 0) {
			segm[uk].Left = st;
			segm[uk].Right = st + fraction(a[i].prob, size);
			segm[uk].ch = a[i].ch;
			st = segm[uk++].Right;
		}
	//Процесс перевода кода в дробь
	vector<int> ans;
	unsigned char c;
	while (!feof(f_in)) {
		fscanf(f_in, "%c", &c);
		if (feof(f_in))
			break;
		ans.push_back(c - '0');
	}

	fraction number = fraction(0, 1);
	tlong step = 2;
```

```
	for (int i = 0; i < (int)ans.size(); i++) {
		number = number + fraction(ans[i], step);
		step *= 2;
	}

	//Процесс декодирования с бинарным поиском
	for (int i = 0; i < size; i++) {
		int down = 0;
		int up = uk;
		for (int j = 0; j < 10; j++) {
			int mid = (up + down) / 2;
			if (segm[mid].Left > number)
				up = mid;
			else
				down = mid + 1;
		}
		for (int j = max(0, down - 2); j < min(uk, down + 3); j++)
			if (number >= segm[j].Left && number < segm[j].Right) {
				fprintf(f_out, "%c", segm[j].ch);
				number = (number - segm[j].Left) / (segm[j].Right - segm[j].Left);
				break;
			}
	}
}

#endif
```

Файл flowrANS.h

```
#pragma once
#ifndef __FLOW_rANS_H__
#define __FLOW_rANS_H__

#include <cstdio>
#include "Structures.h"
using namespace std;
using namespace System;
const int MAX_SIZE = 100000000;
typedef long long ll;

String^ PackFlowrANS(FILE* f, const string &message, symbol* probs, int cnt_alpha, int size,
const int ALPHABET_SIZE) {
	int table_size = CountAllBitsProbs(cnt_alpha, probs, size);
	fprintf(f, "%d\n", size);
	//Вывод таблицы вероятностей символов
	fprintf(f, "%d\n", cnt_alpha);
	for (int i = 0; i < ALPHABET_SIZE; i++)
		if (probs[i].prob > 0) {
			fprintf(f, "%c", probs[i].ch);
			if (probs[i].ch != '\n')
				fprintf(f, "\n");
			fprintf(f, "%d\n", probs[i].prob);
		}

	//Инициализация массивов для кодирования
	symbol *a = new symbol[cnt_alpha];
	int uk = 0;
	int *pos = new int[ALPHABET_SIZE];
	for (int i = 0; i < ALPHABET_SIZE; i++)
		if (probs[i].prob > 0) {
			pos[i] = uk;
			a[uk++] = probs[i];
		}
```

```
	int *b = new int[cnt_alpha];
	b[0] = 0;
	for (int i = 1; i < cnt_alpha; i++)
		b[i] = b[i - 1] + a[i - 1].prob;

	ll L = (ll)size * (ll)ALPHABET_SIZE;
	//Процесс кодирования
	unsigned char *ans = new unsigned char[MAX_SIZE];
	uk = 0;
	int s;
	ll x = 0, x_max;
	for (int i = size - 1; i >= 0; i--) {
		s = pos[unsigned char(message[i])];
		x_max = (2LL * ALPHABET_SIZE) * a[s].prob;
		while (x >= x_max) {
			ans[uk++] = (unsigned char)(x & 1) + '0';
			x >>= 1;
		}
		x = (x / a[s].prob) * size + b[s] + (x % a[s].prob);
	}
	while (x > 0) {
		ans[uk++] = (unsigned char)(x & 1) + '0';
		x >>= 1;
	}
	double aver = uk;	// Среднее количество бит на символ
	for (int i = uk - 1; i >= 0; i--)
		fprintf(f, "%c", ans[i]);
	String^ answer;
	answer = "Size of coding table (bytes): " + table_size.ToString() +
Environment::NewLine;
	answer = answer + "Average quantity of bits per symbol (only code): " + (aver /
((double)size)).ToString() + Environment::NewLine;
	answer = answer + "Average quantity of bits per symbol with coding table: " +
		((aver + (double)table_size * 8.0) / ((double)size)).ToString() +
Environment::NewLine;
	return answer;
}

void DecoderFlowANS(FILE *f_in, FILE *f_out, int cnt_alpha, int size, const int ALPHABET_SIZE)
{
	//Считывание таблицы вероятностей
	symbol *a = new symbol[cnt_alpha];
	int *pos = new int[ALPHABET_SIZE];
	for (int i = 0; i < cnt_alpha; i++) {
		unsigned char c, f_c;
		int d = 0;
		fscanf(f_in, "%c", &c);
		if (c != '\n')
			fscanf(f_in, "\n");
		fscanf(f_in, "%c", &f_c);
		while (f_c != '\n') {
			d = d * 10 + (f_c - '0');
			fscanf(f_in, "%c", &f_c);
		}
		a[i].ch = c;
		a[i].prob = d;
		pos[c] = i;
	}
	//Инициализация массивов для декодирования
	int *b = new int[cnt_alpha];
	b[0] = 0;
	for (int i = 1; i < cnt_alpha; i++)
		b[i] = b[i - 1] + a[i - 1].prob;
```

```
	unsigned char *mas_s = new unsigned char[size];
	int uk = 0;
	for (int i = 0; i < cnt_alpha; i++)
		for (int j = 0; j < a[i].prob; j++)
			mas_s[uk++] = a[i].ch;

	//Декодирование
	ll L = (ll)size * (ll)ALPHABET_SIZE;
	ll x = 0;
	unsigned char c;
	int cnt = 0, f = 0, md, s, t;
	while (cnt < size) {
		if (x >= L && x < (L << 1LL) || f) {
			md = x % size;
			s = mas_s[md];
			t = pos[s];
			fprintf(f_out, "%c", (unsigned char)s);
			x = (x / size) * a[t].prob + md - b[t];
			cnt++;
		}
		while (!feof(f_in) && x < L) {
			fscanf(f_in, "%c", &c);
			if (feof(f_in))
				break;
			x = (x << 1LL) + (c - '0');
		}
		if (x < L)
			f = 1;
	}
}

#endif
```