ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ»
(Н И У « Б е л Г У »)

ИНСТИТУТ ИНЖЕНЕРНЫХ ТЕХНОЛОГИЙ И ЕСТЕСТВЕННЫХ НАУК

КАФЕДРА ИНФОРМАЦИОННЫХ И РОБОТОТЕХНИЧЕСКИХ СИСТЕМ

АНАЛИЗ, ИССЛЕДОВАНИЕ И РАЗРАБОТКА МЕТОДОВ СЖАТИЯ ТОМОГРАФИЧЕСКИХ ДАННЫХ

Магистерская диссертация
обучающегося по направлению подготовки
09.04.02 Информационные системы и технологии
очной формы обучения
группы 07001635
Богдановой Натальи Андреевны

Научный руководитель
к.т.н., доцент
Шамраева Е.О.

Рецензент
к.т.н., доцент
Зайцева Т.В.

БЕЛГОРОД 2018

РЕФЕРАТ

Анализ, исследование и разработка методов сжатия томографических данных. – Богданова Наталья Андреевна, магистерская диссертация, Белгород, Белгородский государственный национальный исследовательский университет (НИУ «БелГУ»), количество страниц 55, включая приложения 70, количество рисунков 39, количество таблиц 4, количество использованных источников 40.

КЛЮЧЕВЫЕ СЛОВА: томограмма, сжатие без потерь, модифицированный RLE-алгоритм, дельта-кодирование, адаптивный алгоритм Хаффмана, комплексный метод.

ОБЪЕКТ ИССЛЕДОВАНИЯ: процесс сжатия томографических данных.

ПРЕДМЕТ ИССЛЕДОВАНИЯ: комплексный метод сжатия томографических данных.

ЦЕЛЬ РАБОТЫ: уменьшение объема томографических данных за счет разработки комплексного метода сжатия без потерь.

ЗАДАЧИ ИССЛЕДОВАНИЯ: изучение проблемы сжатия томографических данных; исследование существующих методов сжатия информации без потерь; разработка метода сжатия томографических данных с учетом особенностей обрабатываемой информации и его описание с помощью диаграмм бизнес-процессов; реализация программного средства на основе разработанного метода сжатия томографических данных, тестирование его на наличие возможных ошибок и их устранение; описание порядка работы с программным средством в текстовой и иллюстративной форме; анализ результатов разработки.

МЕТОДЫ ИССЛЕДОВАНИЯ: системный анализ проблемы сжатия томографических данных, моделирование процессов комплексного метода сжатия без потерь, сравнение результатов сжатия томографических данных комплексным методом и другими существующими подходами.

ПОЛУЧЕННЫЕ РЕЗУЛЬТАТЫ: в работе предложен комплексный метод сжатия без потерь томографических данных, обеспечивающий уменьшение

объема занимаемой памяти, вследствие чего сокращается время и стоимость передачи информации по каналам связи в компьютерных сетях.

НАУЧНАЯ НОВИЗНА: разработка нового метода, который позволяет уменьшить объем томографических данных в большей степени, чем существующие подходы.

ОБЛАСТЬ ПРИМЕНЕНИЯ: использование программной реализации комплексного метода сжатия томографических данных возможно как в медицинских учреждениях (для хранения томограмм пациентов в оптимизированном виде), так и на персональных компьютерах пациентов (для передачи томографических данных по каналам связи в рамках онлайн-консультаций).

ПЕРСПЕКТИВЫ РАЗВИТИЯ: в качестве перспектив развития программного средства для сжатия томографических данных комплексным методом можно указать такие нереализованные возможности, как обработка непосредственно томограмм в формате DICOM, выполнение программы по действиям из контекстного меню при нажатии правой кнопки мыши.

СОДЕРЖАНИЕ

СОКРАЩЕНИЯ И ОБОЗНАЧЕНИЯ

ЛПР – лицо, принимающее решение

МАИ – метод анализа иерархий

ММАИ – мультипликативный метод анализа иерархий

МРТ – магнитно-резонансная томография

ВВЕДЕНИЕ

Одним из результатов информатизации общества является возникновение и повсеместное распространение такого понятия, как телемедицина [1].

Телемедицина – направление медицины, основанное на применении современных коммуникационных технологий для оказания удаленной медицинской помощи и проведения консультаций [2]. Соответственно, удаленное взаимодействие предполагает уменьшение объема передаваемых файлов, например, за счет сжатия информации.

Объектом исследования является процесс сжатия томографических данных.

В качестве предмета исследования выступает комплексный метод сжатия томографических данных.

Цель магистерской диссертации заключается в уменьшении объема томографических данных за счет разработки комплексного метода сжатия без потерь.

Для достижения поставленной цели необходимо выполнить следующие задачи:

- изучить проблемы сжатия томографических данных;
- исследовать существующие методы сжатия информации без потерь;
- разработать метод сжатия томографических данных с учетом особенностей обрабатываемой информации и описать его с помощью диаграмм бизнес-процессов;
- реализовать программное средство на основе разработанного метода сжатия томографических данных, протестировать его на наличие возможных ошибок и устранить их;
- описать порядок работы с программным средством в текстовой и иллюстративной форме;
- произвести анализ результатов разработки.

Научная новизна работы заключается в разработке нового метода, который позволяет уменьшить объем томографических данных в большей степени, чем существующие подходы.

В процессе развития телемедицины может возникнуть проблема, связанная с уменьшением скорости передачи информации из-за большого объема передаваемых данных [1]. Кроме того, серьезной проблемой является долгосрочное хранение результатов томографических исследований для оценки эффективности проводимых лечебных процедур и контроля состояния пациента в динамике [3]. Таким образом, необходимо уменьшать объем информации с технической точки зрения, но при этом сохранять «медицинскую» информацию, заключенную в томограммах, в чем заключается актуальность работы.

Структура магистерской диссертации включает введение, 3 раздела, заключение, список использованных источников и 1 приложение.

Первый раздел посвящен обобщению сведений о томографии и методах сжатия данных без потерь, определению цели и задач исследования.

Во втором разделе представлено вербальное моделирование комплексного метода сжатия томографических данных и проектирование его процессов в методологиях DFD, IDEF0, STD.

Третий раздел содержит функциональные и эксплуатационные требования к программному средству для сжатия томографических данных комплексным методом, описание реализации программного средства и порядка работы с ним, а также сравнительный анализ результатов сжатия.

В приложение вынесен исходный код.

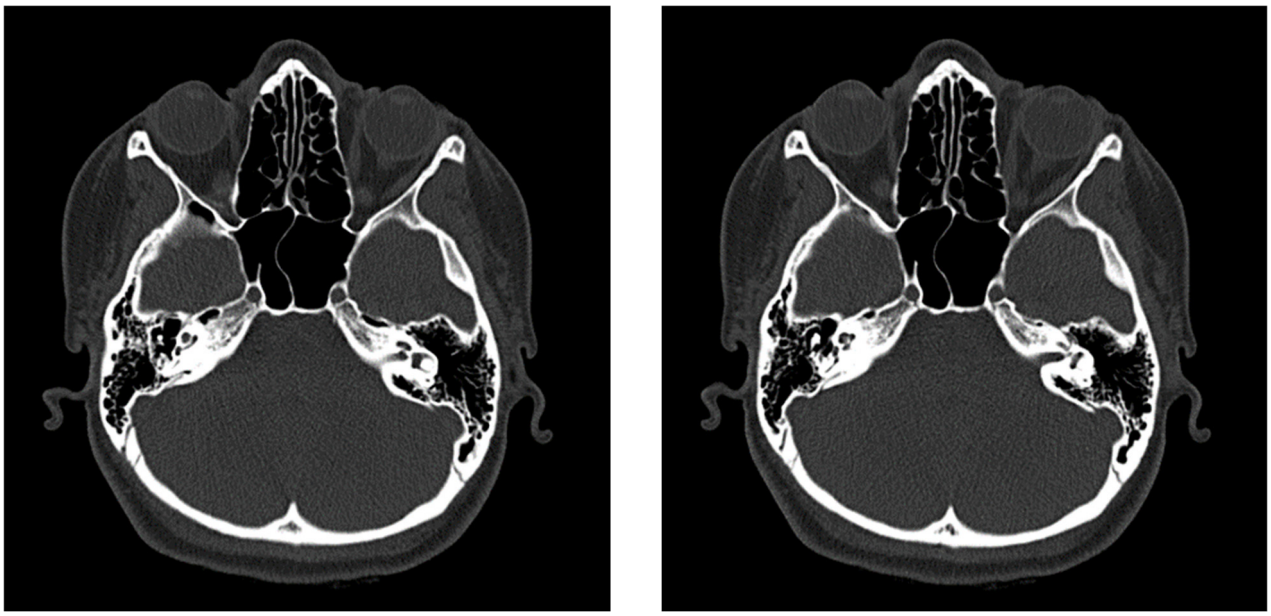
Общий объем магистерской диссертации – 70 страниц. Магистерская диссертация содержит 39 рисунков, 3 формулы, 4 таблицы, 40 использованных источников.

1 Изучение проблемы сжатия томографических данных и существующих подходов ее решения

1.1 Общие сведения о томографии

Годом основания магнитно-резонансной томографии (МРТ) принято считать 1973 год, когда профессор химии Пол Лотербур опубликовал в журнале Nature статью «Создание изображения с помощью индуцированного локального взаимодействия; примеры на основе магнитного резонанса». Позже Питер Мэнсфилд усовершенствовал математические алгоритмы получения изображения. За изобретение метода МРТ оба исследователя в 2003 году получили Нобелевскую премию по медицине [4].

Томограмма – рентгеновский снимок, полученный с помощью послойного изображения внутренней структуры объекта. Внешний вид томографических снимков представлен на рисунке 1.1,а – 1.1,г [5].



а б

Рисунок 1.1 – Томограммы: а) первый снимок; б) второй снимок; в) третий снимок; г) четвертый снимок

в г

Рисунок 1.1, лист 2

Специфичность томографических данных заключается в том, что, во-первых, размер всех томографических снимков составляет 512×512 пикселей, а во всех файлах снимки одинаково позиционированы; во-вторых, часть снимка состоит из длинных серий одинаковых (черных) пикселей; в-третьих, различия между двумя соседними томограммами незначительны [3].

Результаты одного томографического исследования могут занимать порядка 100 МБ памяти (размер одного снимка – от 500 кБ до 700 кБ). Это связано с тем, что для томограмм существует специальный формат – DICOM, который позволяет создавать, хранить, передавать и печатать отдельные кадры изображения, серии кадров, информацию о пациенте, исследовании, оборудовании, учреждениях, медицинском персонале, производящем обследование [6].

В связи с тем, что интерфейсы программ для работы с DICOM-файлами очень громоздки и практически не настраиваемы, вполне естественно конвертировать файл в распространенный графический формат *.BMP и производить компрессию над ним. При этом важно не допустить потери

медицинских данных, что уже исключает использование методов такого класса сжатия информации, как сжатие с потерями [6].

1.2 Обзор методов сжатия данных без потерь

Сжатие данных без потерь – метод сжатия данных (видео, аудио, графики, документов, представленных в цифровом виде), при использовании которого закодированные данные однозначно могут быть восстановлены с точностью до бита [7], т.е. информация после восстановления будет идентична исходной до сжатия.

К наиболее распространенным методам сжатия информации без потерь, в частности изображений, относятся следующие:

а) алгоритм RLE. Данный алгоритм необычайно прост в реализации. Групповое кодирование – от английского Run Length Encoding (RLE) – один из самых старых и самых простых алгоритмов архивации графики. Изображение в нем (как и в нескольких алгоритмах, описанных ниже) вытягивается в цепочку байт по строкам растра. Само сжатие в RLE происходит за счет того, что в исходном изображении встречаются цепочки одинаковых байт. Замена их на пары <счетчик повторений, значение> уменьшает избыточность данных [8]. Пример работы алгоритма RLE представлен на рисунке 1.2 [9].

Рисунок 1.2 – Действие алгоритма RLE

б) алгоритм LZW. Название алгоритм получил по первым буквам фамилий его разработчиков – Lempel, Ziv и Welch. Сжатие в нем, в отличие от RLE, осуществляется уже за счет одинаковых цепочек байт. Ориентирован LZW на 8-битные изображения, построенные на компьютере. Сжимает за счет одинаковых подцепочек в потоке. LZW универсален – именно его варианты используются в обычных архиваторах [8]. Пример работы алгоритма LZW представлен на рисунке 1.3 [10].

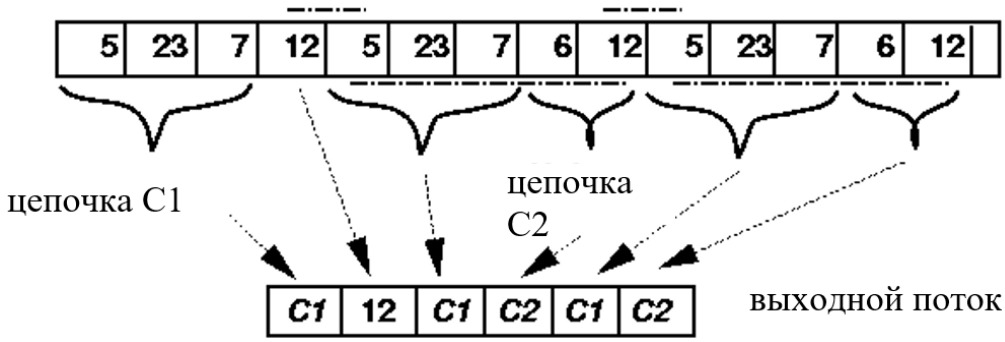


Рисунок 1.3 – Действие алгоритма LZW

в) алгоритм LZ77. Еще один представитель LZ-семейства. Основным отличием от алгоритма LZW является то, что замена повторного вхождения строки осуществляется за счет ссылки на одну из предыдущих позиций вхождения. LZ77 использует принцип «скользящего окна» (динамическая структура данных для хранения полученной ранее информации и организации вставки ссылок на значения в «скользящем окне», а не вставки самих значений). Пример работы алгоритма LZ77 представлен на рисунке 1.4 [11].

г) алгоритм Хаффмана (с фиксированной таблицей). Алгоритм начинается составлением списка символов алфавита в порядке убывания их вероятностей. Затем от корня строится дерево, листьями которого служат эти символы. Это делается по шагам, причем на каждом шаге выбираются два символа с наименьшими вероятностями, добавляются наверх частичного дерева, удаляются из списка и заменяются вспомогательным символом, представляющим эти два символа.

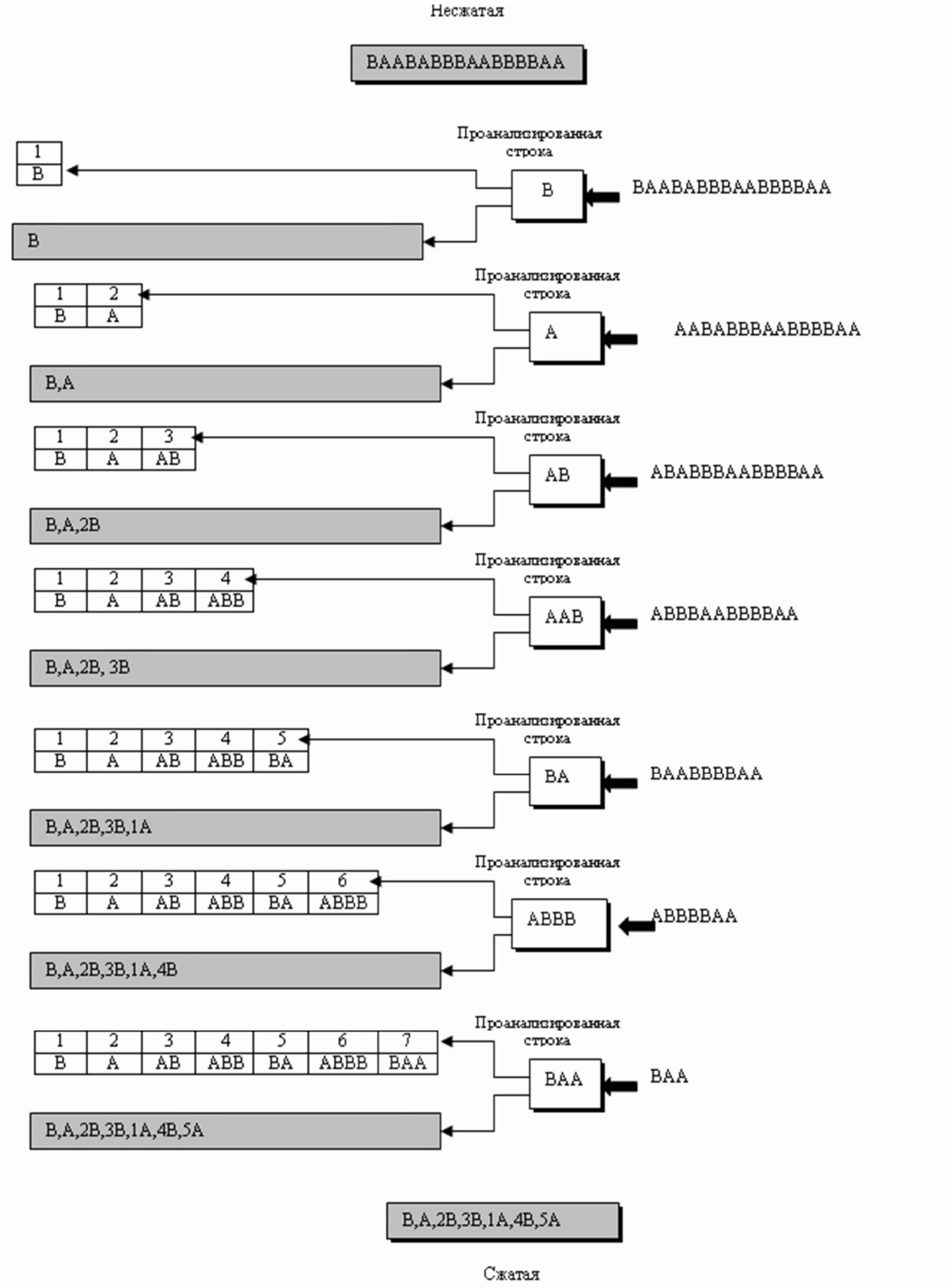


Рисунок 1.4 – Действие алгоритма LZ77

Вспомогательному символу приписывается вероятность, равная сумме вероятностей, выбранных на этом шаге символов. Когда список сокращается до

одного вспомогательного символа, представляющего весь алфавит, дерево объявляется построенным. Завершается алгоритм спуском по дереву и построением кодов всех символов. Метод Хаффмана с фиксированной таблицей предполагает, что частоты символов алфавита известны декодеру [12]. Пример работы алгоритма Хаффмана (с фиксированной таблицей) представлен на рисунке 1.5 [13]

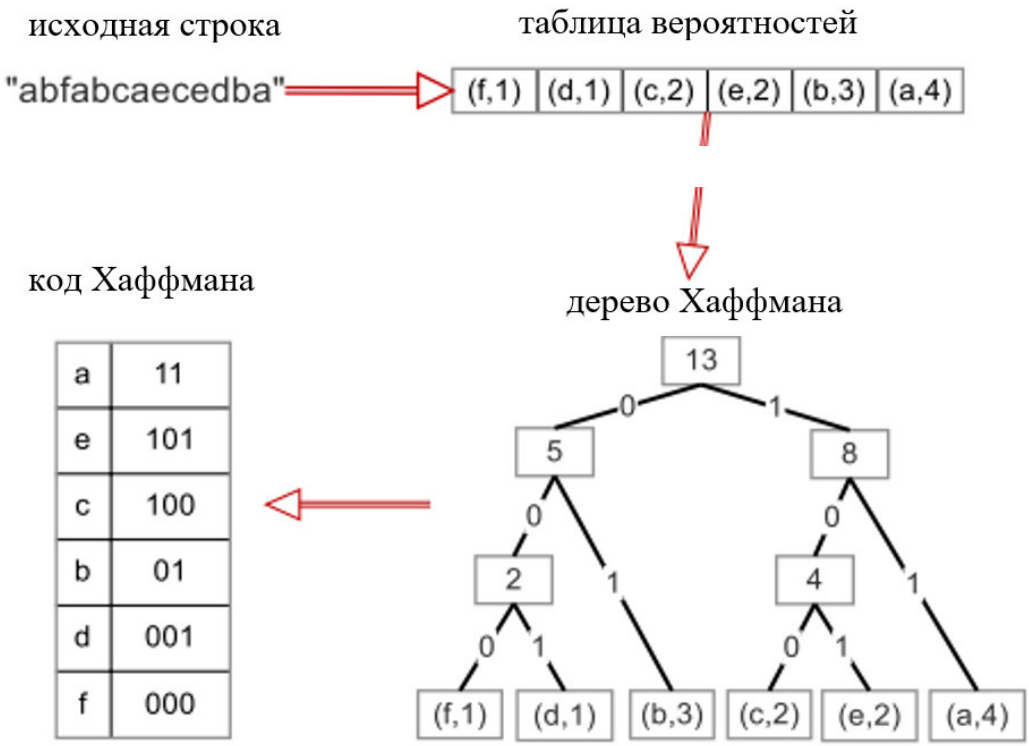


Рисунок 1.5 – Действие алгоритма Хаффмана с фиксированной таблицей

д) адаптивное кодирование Хаффмана (также называемое динамическое кодирование Хаффмана) – адаптивный метод, основанный на кодировании Хаффмана. Он позволяет строить кодовую схему в поточном режиме (без предварительного сканирования данных), не имея никаких начальных знаний из исходного распределения, что позволяет за один проход сжать данные. Строится так называемое дерево Хаффмана, удовлетворяющее следующему свойству: при изучении дерева по всем уровням слева направо и снизу вверх (от листьев к корню) частоты будут упорядочены по возрастанию (неубыванию), таким образом, нижний левый узел имеет наименьшую частоту, а верхний

правый (корень) – наибольшую. Преимуществом этого способа является возможность кодировать на лету [14]. Пример работы адаптивного алгоритма Хаффмана представлен на рисунке 1.6 [15].

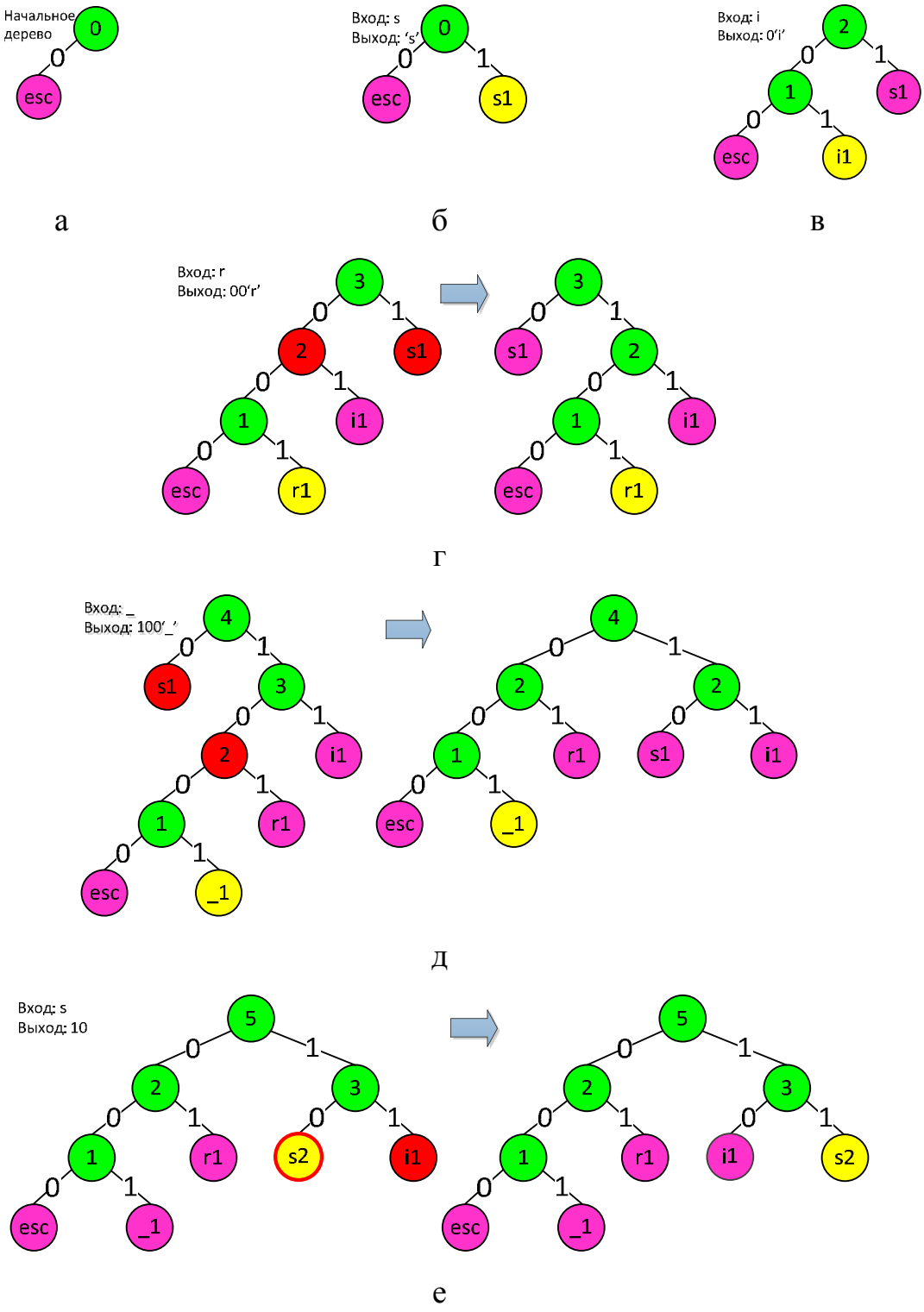


Рисунок 1.6 – Пример кодирования адаптивным алгоритмом Хаффмана фразы «sir_s»: а) шаг 1; б) шаг 2; в) шаг 3; г) шаг 4; д) шаг 5; е) шаг 6

е) арифметическое кодирование. Алгоритм создает неблоковые коды. В этом кодировании не существует однозначного соответствия между символами источника и кодовыми словами. Вся последовательность символов источника соотнесена с одним арифметическим кодовым словом, которое задает интервал вещественных чисел между 0 и 1. Каждый поступающий на обработку символ данных уменьшает этот интервал пропорционально вероятности своего появления (при этом число бит для записи этого интервала увеличивается) [16]. Пример работы арифметического кодирования представлен на рисунке 1.7.

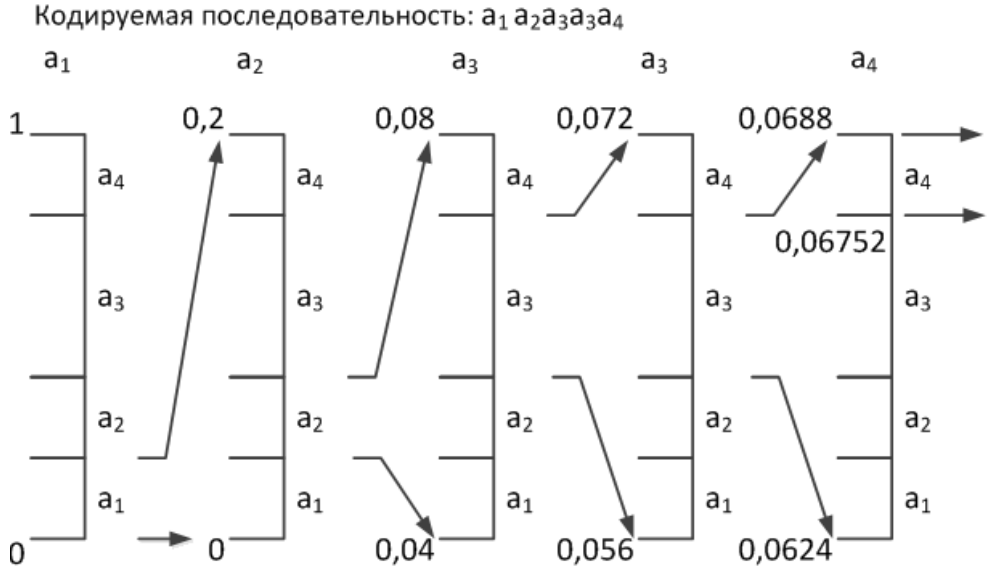


Рисунок 1.7 – Действие арифметического кодирования

ж) JBIG. Алгоритм разработан группой экспертов ISO (Joint Bi-level Experts Group) специально для сжатия однобитных черно-белых изображений. При этом алгоритм разбивает их на отдельные битовые плоскости. JBIG позволяет управлять такими параметрами, как порядок разбиения изображения на битовые плоскости, ширина полос в изображении, уровни масштабирования. Алгоритм построен на базе Q-кодировщика, патентом на который владеет IBM. Q-кодер, так же, как и алгоритм Хаффмана, использует для чаще появляющихся символов короткие цепочки, а для реже появляющихся – длинные. Однако, в отличие от него, в алгоритме используются и

последовательности символов [8]. Пример работы алгоритма JBIG представлен на рисунке 1.8 [17].

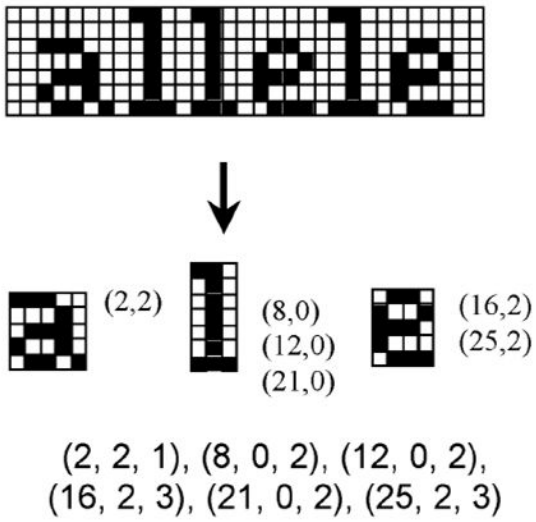


Рисунок 1.8 – Действие алгоритма JBIG

На сегодняшний день к наиболее популярным форматам архивации файлов, основанным на комбинациях перечисленных алгоритмов и их модификаций, относятся следующие:

а) ZIP. Популярный формат архивации файлов и сжатия данных без потерь. Наиболее часто в ZIP используется алгоритм сжатия Deflate (комбинация алгоритмов LZ77 и Хаффмана). Формат был создан в 1989 году Филом Кацем и реализован в программе PKZIP компании PKWARE в качестве замены формату архивов ARC Тома Хендерсона [18].

б) 7z. Свободный формат сжатия данных, поддерживающий несколько различных алгоритмов сжатия и шифрования данных. Формат 7z изначально был реализован в архиваторе 7-Zip [19]. Алгоритм основан на схеме сжатия данных по словарю, сходной с использованной в LZ77, что обеспечивает высокий коэффициент сжатия, а также использует статистические методы Хаффмана или арифметическое кодирование.

в) RAR. Проприетарный формат сжатия данных и условно бесплатная программа-архиватор. Формат разработан российским программистом

Евгением Рошалом (отсюда и название: Roshal ARchiver). Метод сжатия так и остается «закрытым» для пользователей PC и открытым для ZX Spectrum (только RAR v2) [20]. Для текстовых данных RAR v.4.x использует дающие высокий фактор сжатия, но медленные в упаковке и распаковке алгоритм Хаффмана или арифметическое кодирование, также применяемые в архиваторах 7-Zip и WinZip. Для архивов формата RAR5 эта возможность отсутствует.

В связи с тем, что существующие универсальные алгоритмы сжатия не учитывают совокупность особенностей томографических данных [21], использование перечисленных форматов архивации файлов может оказаться неэффективным.

1.3 Определение цели и задач исследования

Цель исследования заключается в разработке эффективного метода сжатия томографических данных, обеспечивающего:

- сокращение объема памяти, необходимой для хранения данных;
- сокращение времени передачи информации по каналам связи в компьютерных сетях;
- сокращение стоимости пользования каналами связи за счет сокращения времени передачи информации.

Кроме того, упаковка в один архивный файл группы файлов существенно упрощает их перенос с одного компьютера на другой, сокращает время копирования файлов на диски, позволяет защитить информацию от несанкционированного доступа, способствует защите от заражения компьютерными вирусами [22].

Для достижения цели исследования необходимо выполнить следующие задачи:

а) выявить особенности томографических данных;

б) с учетом выявленных особенностей разработать метод сжатия томографических данных, отвечающий таким требованиям, как:

1) устранение межэлементной избыточности (большая часть содержащейся информации является избыточной, поскольку значение каждого элемента изображения может быть предсказано по значениям его соседних [23]);

2) устранение межкадровой избыточности, возникающей из-за значительной корреляции (95-99%) двух соседних томограмм [3];

3) сжатие без потерь для сохранения «медицинской» информации;

4) адаптация под полутоновое изображение;

5) минимизация временных и ресурсных затрат;

в) теоретически и практически обосновать преимущество использования разработанного метода в сравнении с существующими подходами к сжатию томографических данных.

Выводы по первому разделу: в настоящий момент активно ведутся работы по разработке методов и алгоритмов сжатия без потерь, которые обеспечили бы максимально возможный коэффициент сжатия, для архивации томографических данных [24] ввиду недостаточной эффективности существующих способов.

2 Разработка комплексного метода сжатия томографических данных

2.1 Вербальное моделирование комплексного метода

Современные медицинские программные приложения позволяют установить точный диагноз за короткое время, но зачастую требуют больших информационных ресурсов и затрат. Так, исследование томографических данных может быть оптимизировано за счет компрессии томограмм [25].

В связи с тем, что способы сжатия информации имеют свои достоинства и недостатки, а сами томограммы обладают определенными особенностями, выбор эффективного алгоритма для сжатия томографических данных – весьма актуальная проблема [26].

Зачастую в качестве инструмента системного подхода к решению проблем принятия решений используется метод анализа иерархий (МАИ). МАИ не предписывает лицу, принимающему решение (ЛПР), какого-либо «правильного» решения, а позволяет ему в интерактивном режиме найти такой вариант (альтернативу), который наилучшим образом согласуется с его пониманием сути проблемы и требованиями к ее решению [27].

Цель МАИ – определение приоритетов альтернатив в случае слабо структурированных и неструктурированных проблем [28].

В данном исследовании использована модификация МАИ – мультипликативный метод анализа иерархий (ММАИ), в основу которого положены два основных положения:

а) если ЛПР, определяет отношения (а не абсолютные значения) двух элементов соответствующего уровня иерархии, то более логично перемножать такие отношения, чем суммировать значения, полученные из сравнений;

б) переход от вербальных сравнений к числам должен происходить на основе некоторых предположений о поведении человека при сравнительных измерениях [29].

То есть ЛПР оценивает стимулы на одномерной оси желательности S_j относительно S_k по каждому из соответствующих критериев [30] в соответствии с таблицей 2.1.

Таблица 2.1 – Шкала относительной важности критериев по Ф.Лутсма

Значение	Уровень относительной важности критериев
-8	S_j намного менее желательно, чем S_k
-6	S_j гораздо менее желательно, чем S_k
-4	S_j (определенно) менее желательно, чем S_k
-2	S_j несколько менее желательно, чем S_k
0	S_j так же желательно, как и S_k (в равной степени желательно)
+2	S_j несколько более желательно, чем S_k
+4	S_j (определенно) более желательно, чем S_k
+6	S_j гораздо более желательно, чем S_k
+8	S_j намного более желательно, чем S_k

На первом этапе выбора посредством ММАИ алгоритма для сжатия томографических данных определена иерархическая структура проблемы (цель – критерии – альтернативы), что представлено на рисунке 2.1.

Сущность представленных в структуре проблемы альтернатив заключается в следующем:

- RLE-алгоритм заменяет повторяющиеся символы (серии) на один символ и число его повторов;

- метод Хаффмана позволяет строить кодовую схему в поточном режиме (без предварительного сканирования данных), не имея никаких начальных знаний из исходного распределения, что позволяет за один проход сжать данные [14];

- комплексный метод заключается в сжатии без потерь на основе модифицированного RLE-алгоритма, метода дельта-кодирования и адаптивного метода Хаффмана [26].

Для дальнейшего анализа следует обратить внимание на особенности комплексного метода.

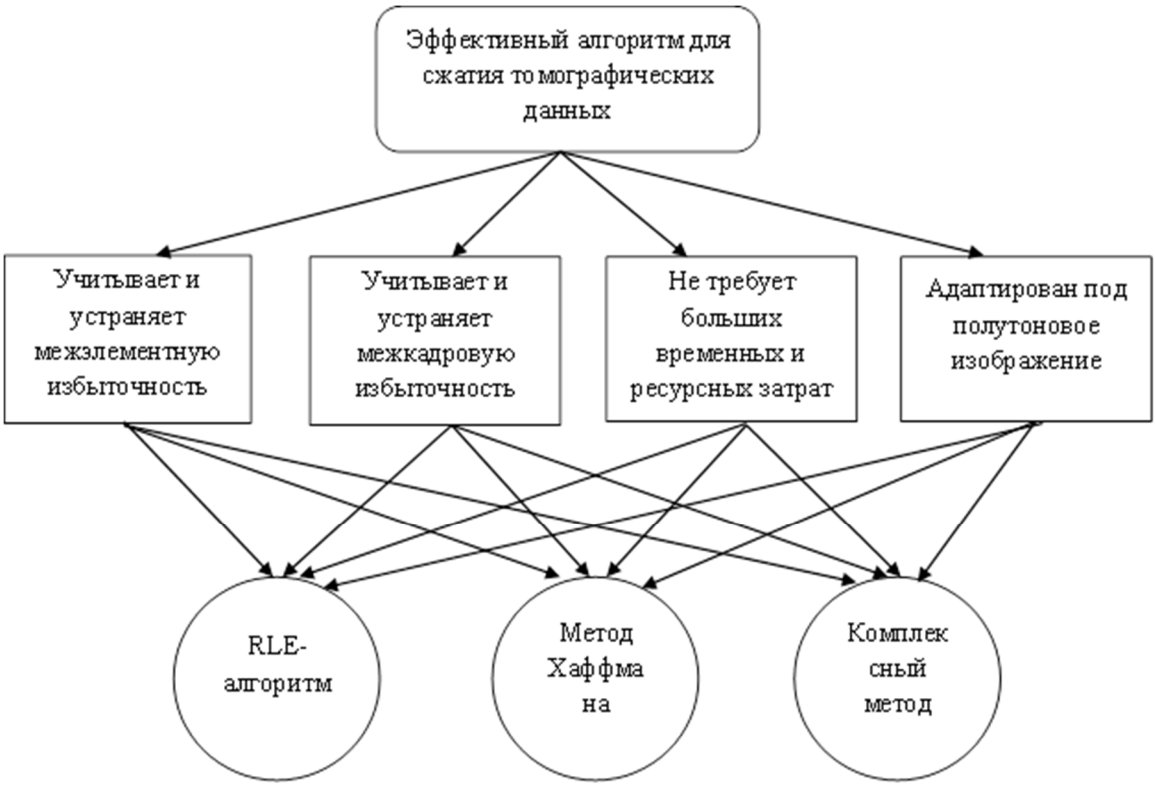


Рисунок 2.1 – Структура проблемы выбора алгоритма

По причине того, что матрица изображения, полученного в процессе МРТ, состоит из пикселей, которые на экране монитора выглядят как участки различных оттенков серого, как показано на рисунке 2.2 [5], использование стандартного RLE-алгоритма непродуктивно: сжатие за счет объединения больших блоков с нулевой яркостью (черные области на снимке) нивелируется кодированием единичных серых точек (точки в области мозгового вещества на снимке).

В соответствии с вышеизложенным, модификация алгоритма кодирования длин серий (RLE) заключается в следующем: значения элементов

вдоль строки изображения f(x,0),f(x,1),…,f(x,511) преобразуются в соответствии с формулой (2.1):

$$RLE_m = \begin{cases} (0, w_1), (0, w_2), ..., (0, w_i), \text{если } g_i = 0; \\ g_1, g_2, ..., g_i, \text{в остальных случаях} \end{cases} \quad (2.1)$$

где w_i – длина серии, пиксель;

g_i – значение яркости на отрезке (серии) i, бит.

То есть в серии объединяются только пиксели с «нулевой» яркостью, что позволяет устранить межэлементную избыточность на томограмме [31].

Рисунок 2.2 – Разнообразие оттенков серого на снимке

Для работы с единой структурой данных стандартный RLE-алгоритм модифицирован под работу с однобайтовыми указателями, для чего на начальном этапе:

- размер томограммы уменьшается до 510×510 путем удаления по ее периметру по одному пикселю (обычно с «нулевой» яркостью);

- томограмма по вертикали и горизонтали делится пополам и дальнейшее преобразование осуществляется над снимками размером 255×255 [32].

Более того, пиксели с «нулевой» яркостью, как правило, располагаются по углам снимка, поэтому преобразование данных модифицированным алгоритмом RLE необходимо осуществлять от внешнего края к внутреннему в каждой части поделенного снимка, что представлено на рисунке 2.3.

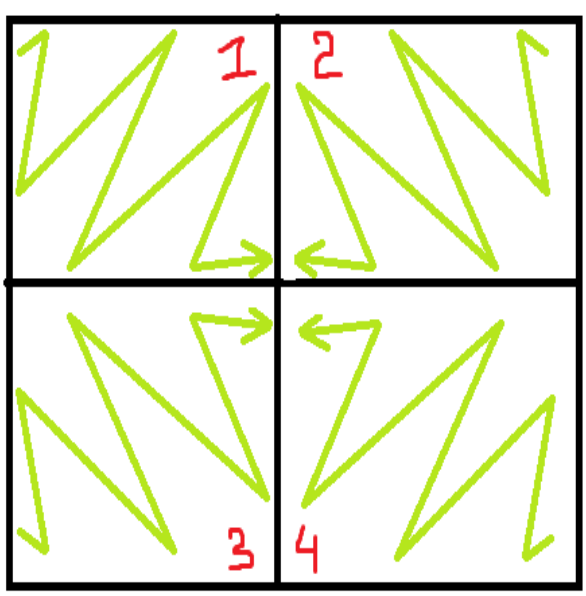


Рисунок 2.3 – Считывание данных

В качестве примера действия модифицированного алгоритма RLE как составляющей комплексного метода сжатия томографических данных приведен рисунок 2.4 [31].

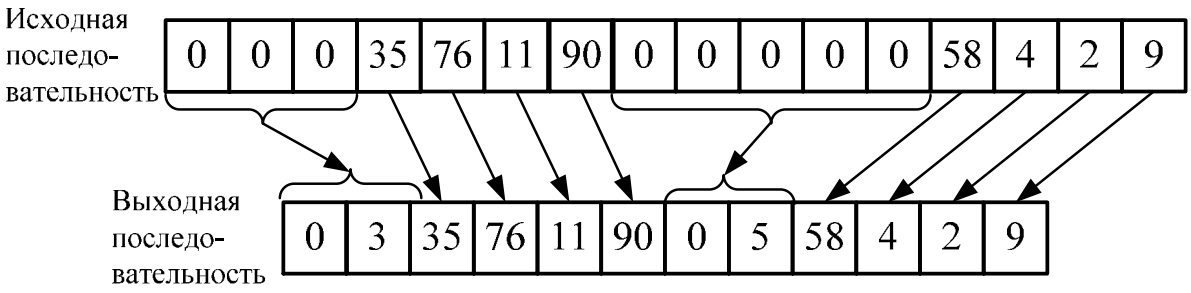


Рисунок 2.4 – Действие модифицированного алгоритма RLE

В комплексном методе модифицированный алгоритм RLE предназначен для оптимизации так называемого базового снимка – первого в последовательности снимков с наименее выраженной и не превышающей 10% межкадровой избыточностью.

Алгоритм нахождения разницы между базовым и текущим снимком представлен на рисунке 2.5.

Рисунок 2.5 – Расчет расхождения снимка с принятым за базовый

Межкадровую избыточность позволяет устранить метод дельта-кодирования. Использующийся обычно для компрессии видеоинформации, этот способ представления данных в виде разницы (дельты) между последовательными данными вместо самих данных, применим также и к изображениям [33], в частности, к томограммам, снимки которых имеют незначительные изменения относительно предшествующих и последующих изображений, что представлено на рисунке 2.6.

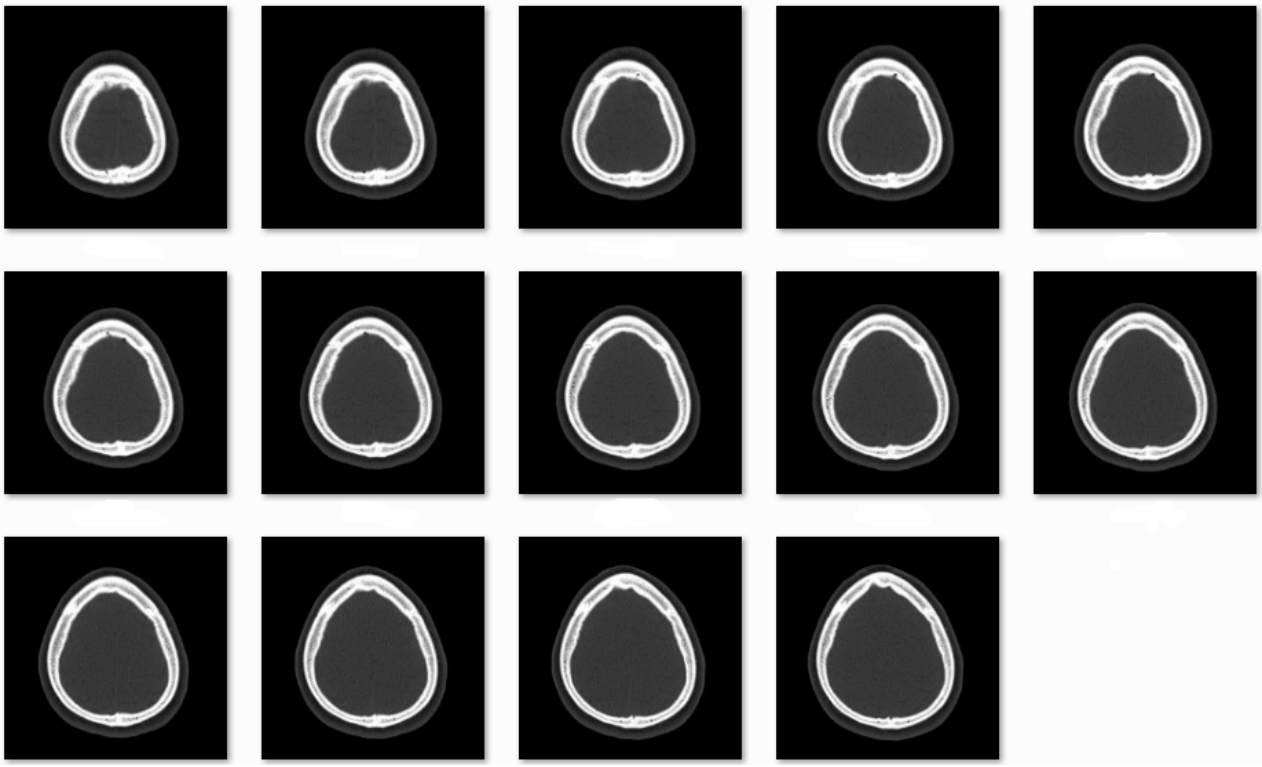


Рисунок 2.6 – Последовательный ряд снимков в томограмме

В качестве примера дельта-кодирования как составляющей комплексного метода сжатия томографических данных приведен рисунок 2.7 [31].

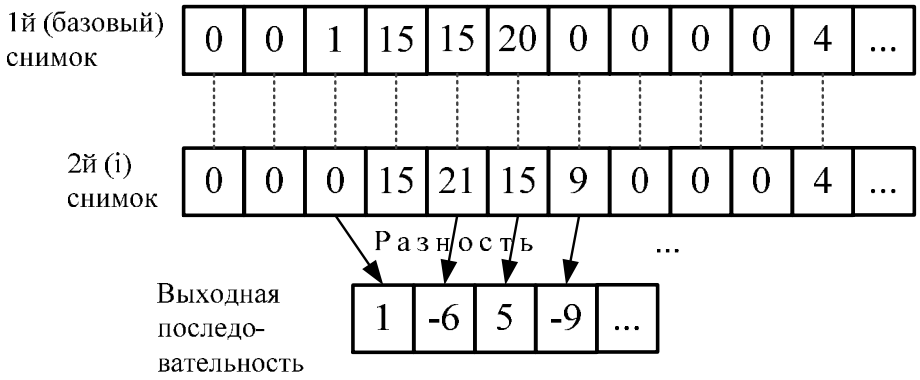


Рисунок 2.7 – Дельта-кодирование томографических данных

Стоит отметить, что в исследовании ММАИ дельта-кодирование не представлено в виде отдельной альтернативы ввиду своего использования по отношению не к единичному изображению, а к множеству.

Заключительным этапом комплексного метода является выполнение адаптивного алгоритма Хаффмана.

В соответствии с вышеизложенным определены спецификации процессов в виде блок-схем алгоритмов, выполненных с использованием MS Office Visio 2010 и представленных на рисунках 2.8 – 2.9.

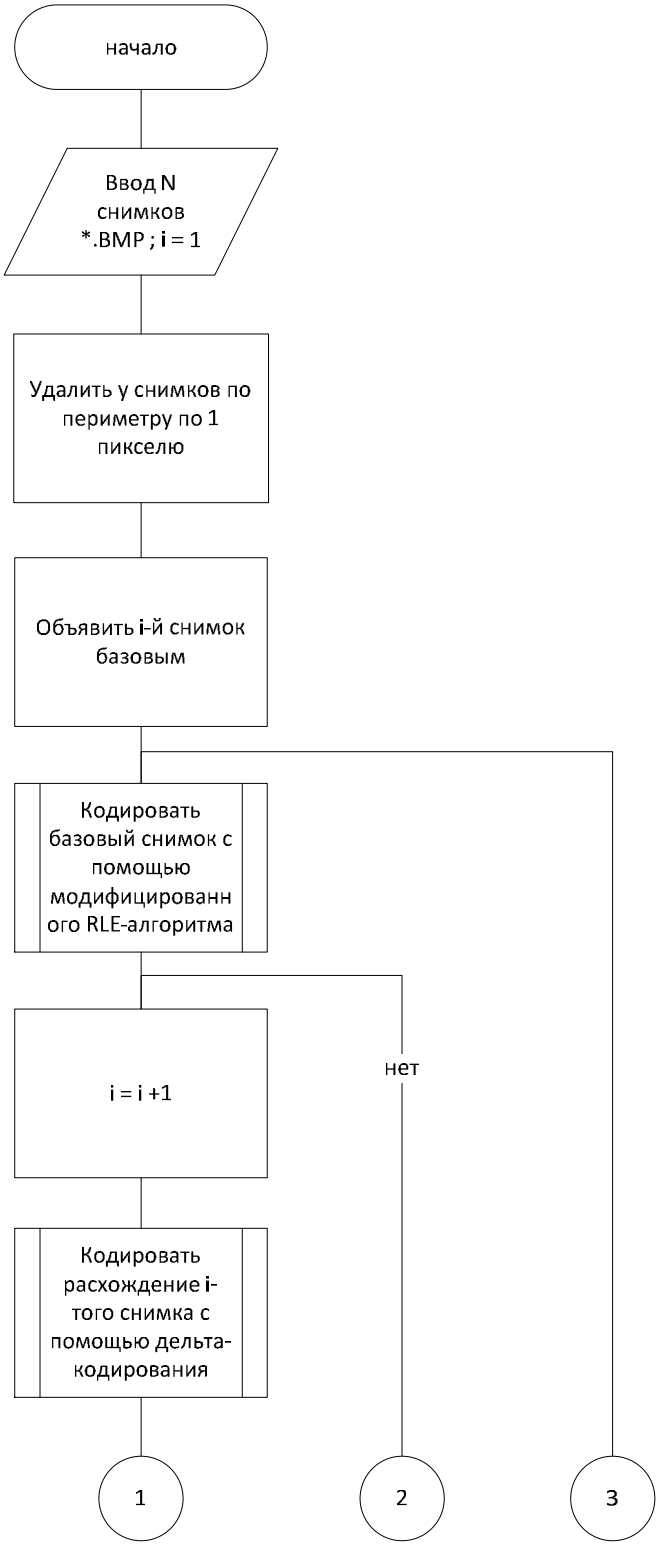


Рисунок 2.8 – Основной алгоритм сжатия томографических данных комплексным методом без потерь (часть 1)

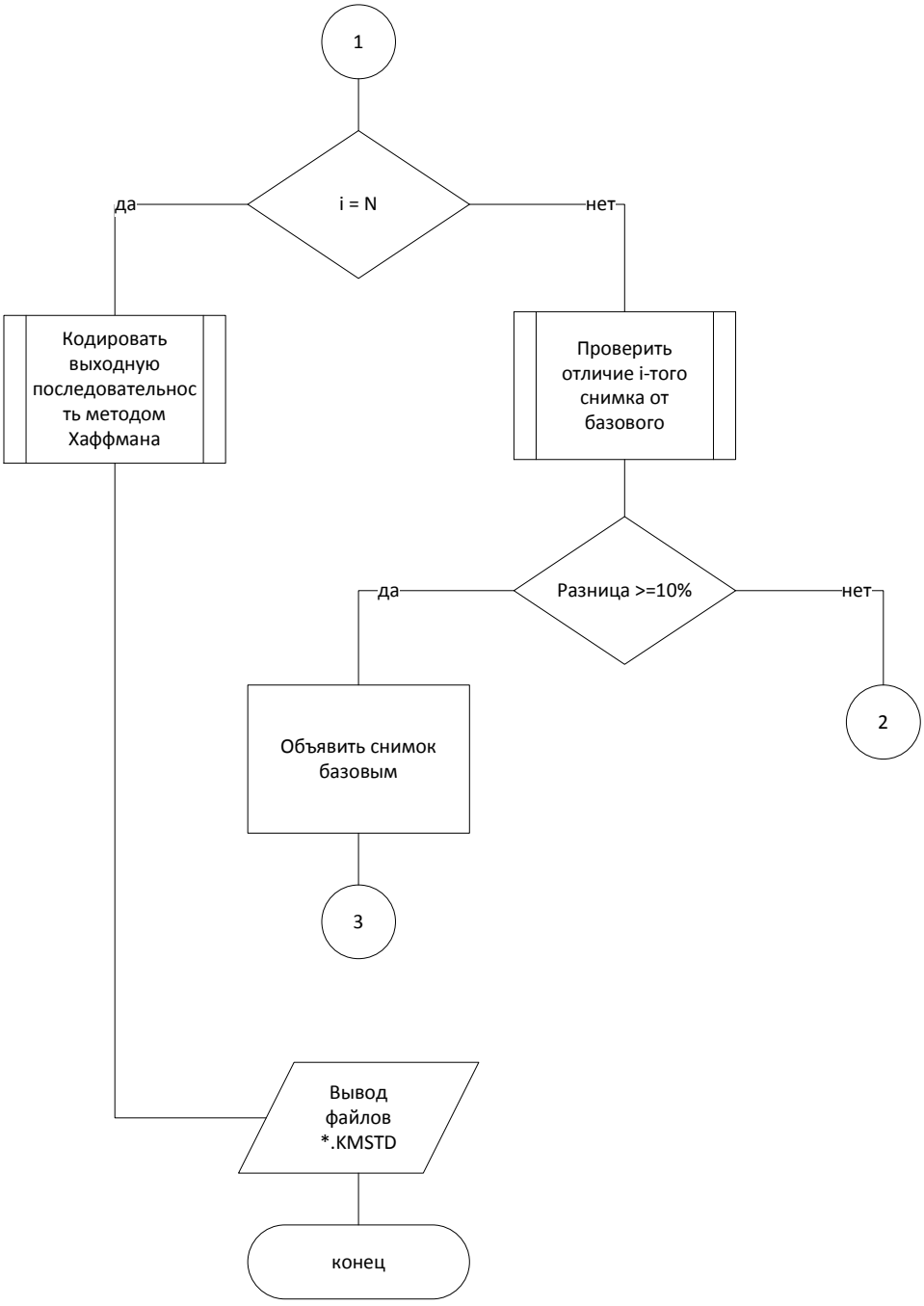


Рисунок 2.9 – Основной алгоритм сжатия томографических данных комплексным методом без потерь (часть 2)

В продолжение исследования ММАИ, осуществлен расчет нормализованных векторов локальных приоритетов для каждого критерия путем выполнения операции деления значения оценки собственного вектора на сумму оценок векторов:

- критерия 1 (учитывает и устраняет межэлементную избыточность) составил 0,4678;

- для критерия 2 (учитывает и устраняет межкадровую избыточность) – 0,4678;

- для критерия 3 (не требует больших временных и ресурсных затрат) – 0,0012;

- для критерия 4 (адаптирован под полутоновое изображение) – 0,0633 [26].

Результаты расчетов с учетом взвешенных векторов приоритетов критериев и альтернатив представлены в таблице 2.2.

Таблица 2.2 – Иерархический синтез для расчета глобальных приоритетов

Альтернативы	Критерии				Глобальные приоритеты
	К1	К2	К3	К4	
	Численное значение вектора приоритета				
	0,4678	0,4678	0,0012	0,0633	
А1	0,4999	0,0003	0,9796	0,1189	0,2427
А2	0,0002	0,0003	0,0179	0,0022	0,0004
А3	0,4999	0,9993	0,0024	0,8789	0,7569

Исходя из данных таблицы 2.2, можно сделать следующие выводы:

- по критерию 1 RLE-алгоритм и комплексный метод одинаково более приоритетны, чем метод Хаффмана;

- по критерию 2 RLE-алгоритм и метод Хаффмана одинаково менее приоритетны, чем комплексный метод;

- по критерию 3 комплексный метод значительно уступает другим альтернативам;

- по критерию 4 минимальный приоритет у метода Хаффмана.

По результатам расчетов построена диаграмма, представленная на рисунке 2.10.

Рисунок 2.10 – Результаты выбора ММАИ эффективного алгоритма для сжатия томографических данных

На диаграмме видно, что наиболее эффективным алгоритмом для сжатия томографических данных является комплексный метод, наименее – метод Хаффмана.

Несмотря на необходимость больших временных и ресурсных затрат, реализацию программного средства для сжатия и восстановления томограмм, рекомендуется проводить на основе комплексного метода.

2.2 Проектирование процессов комплексного метода

В последнее время значительно вырос интерес к CASE-технологиям (Computer-Aided Software/System Engineering) и CASE-средствам, позволяющим максимально систематизировать и автоматизировать все этапы разработки программного обеспечения [34].

AllFusion Process Modeler 7 – это средство функционального моделирования, реализующее 3 вида методологий (IDEF0, IDEF3, DFD), каждая из которых решает свои специфические задачи [35]. С использованием данного

CASE-средства определены диаграммы потоков данных (методология DFD) и функциональные диаграммы (методология IDEF0).

Для построения DFD-диаграмм использованы следующие компоненты: внешние сущности, процессы, хранилища и потоки данных. Так, контекстная диаграмма содержит внешнюю сущность «Пользователь ПК», процесс «Обработка томографических данных комплексным методом без потерь», хранилище «Папка на ПК (родительская)» и потоки данных «Запрос на сжатие», «Запрос на восстановление», «Файлы в формате *.BMP (до обработки)», «Файлы в формате *.KMSTD (до обработки)», «Файлы в формате *.KMSTD (после обработки)», «Файлы в формате *.BMP (после обработки)».

Трехуровневая модель сжатия томографических данных комплексным методом, выполненная в методологии DFD, представлена на рисунках 2.11 – 2.14.

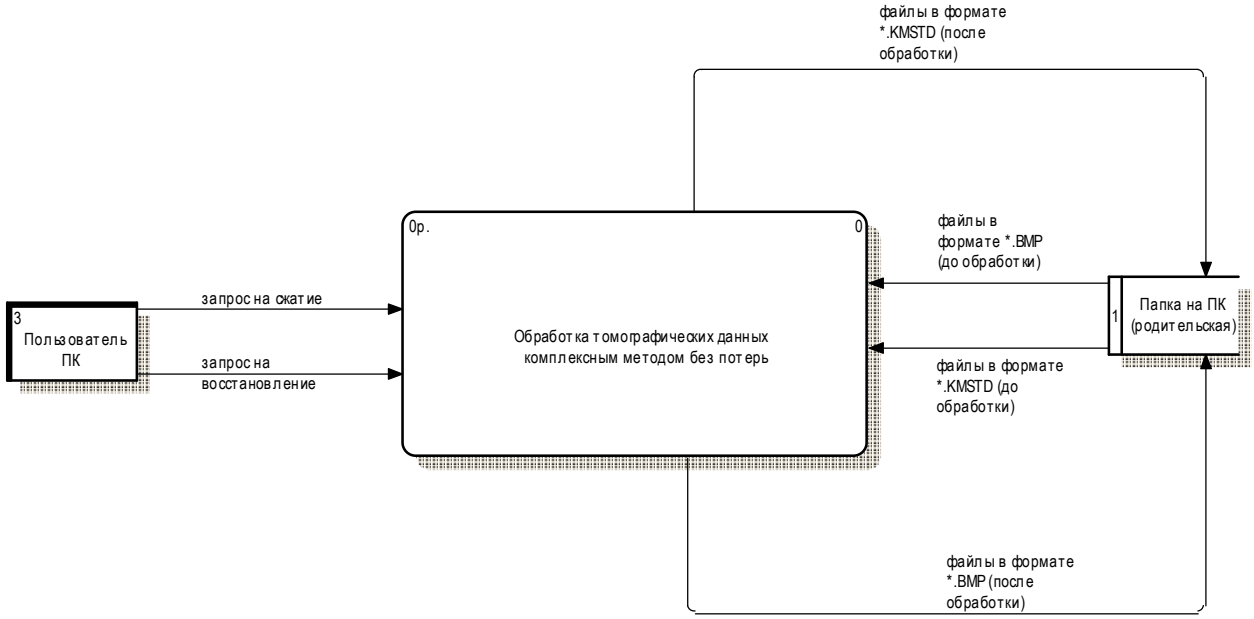


Рисунок 2.11 – Контекстная диаграмма потоков данных «Обработка томографических данных комплексным методом без потерь»

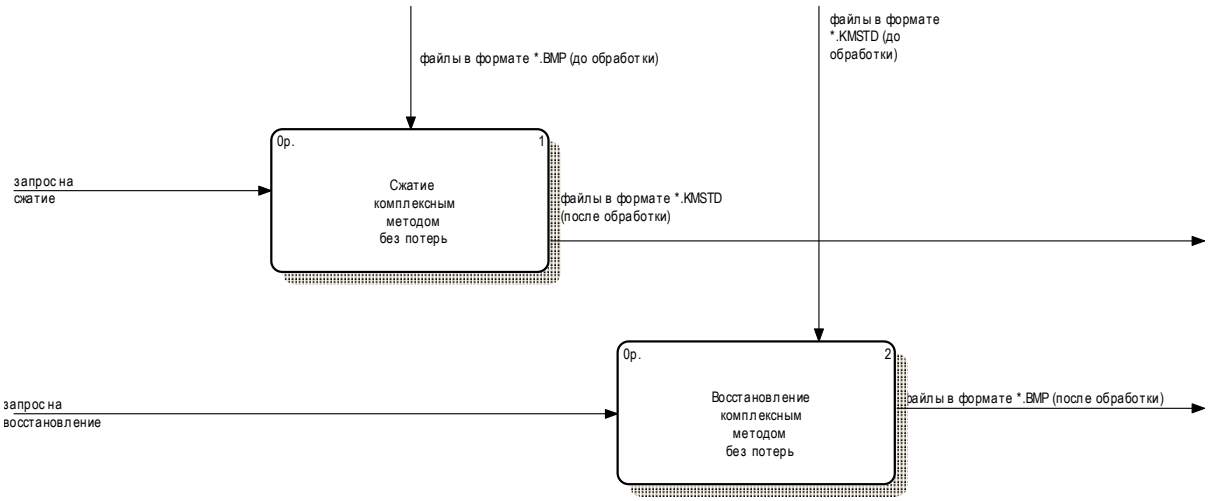


Рисунок 2.12 – Детализация диаграммы потоков данных «Обработка томографических данных комплексным методом без потерь»

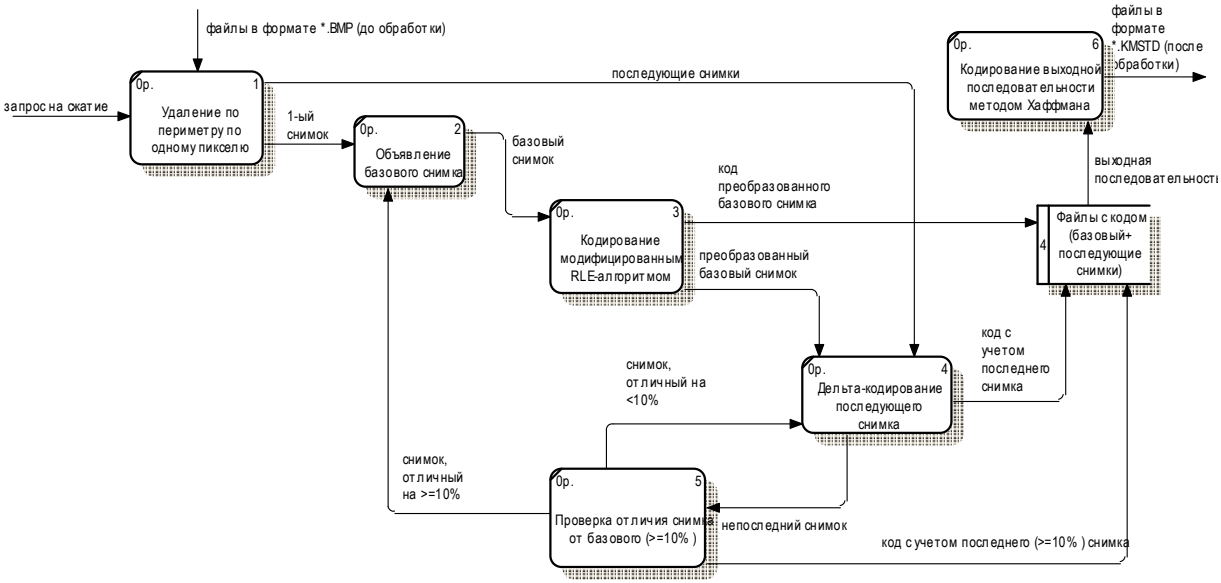


Рисунок 2.13 – Диаграмма декомпозиции «Сжатие комплексным методом»

Посредством DFD-диаграмм продемонстрировано, как каждый процесс преобразует свои входные данные в выходные, а также выявлены отношения между этими процессами.

Для построения IDEF0-диаграмм, отражающих взаимосвязи функций разрабатываемого программного обеспечения, использованы следующие компоненты: блоки функций, стрелки входа, выхода, управления и механизма.

Сжатие томографических данных комплексным методом обеспечивают следующие входные параметры:

- путь к папке с данными формата *.BMP (томографические данные до сжатия комплексным методом);

- путь к папке с данными формата *.KMSTD (томографические данные после сжатия комплексным методом).

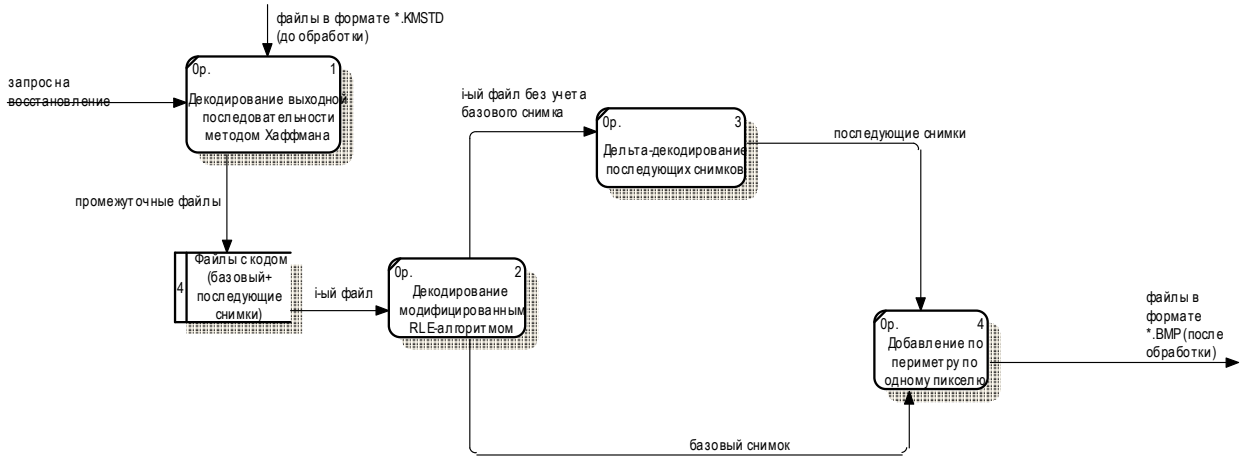


Рисунок 2.14 – Диаграмма декомпозиции «Восстановление комплексным методом без потерь»

Результатом функционирования программного средства являются следующие выходные параметры:

- сжатые данные;

- восстановленные данные.

Ресурсами, с помощью которых выполняется сжатие томографических данных комплексным методом, являются следующие механизмы:

- пользователь ПК;

- программное приложение.

К влияющим на сжатие томографических данных условиям, правилам, стратегиям, стандартам, относятся следующие параметры управления:

- требования по эксплуатации операционной системы;

- алгоритмы программного приложения.

Трехуровневая модель сжатия томографических данных, выполненная в методологии IDEF0, представлена на рисунках 2.15 – 2.18.

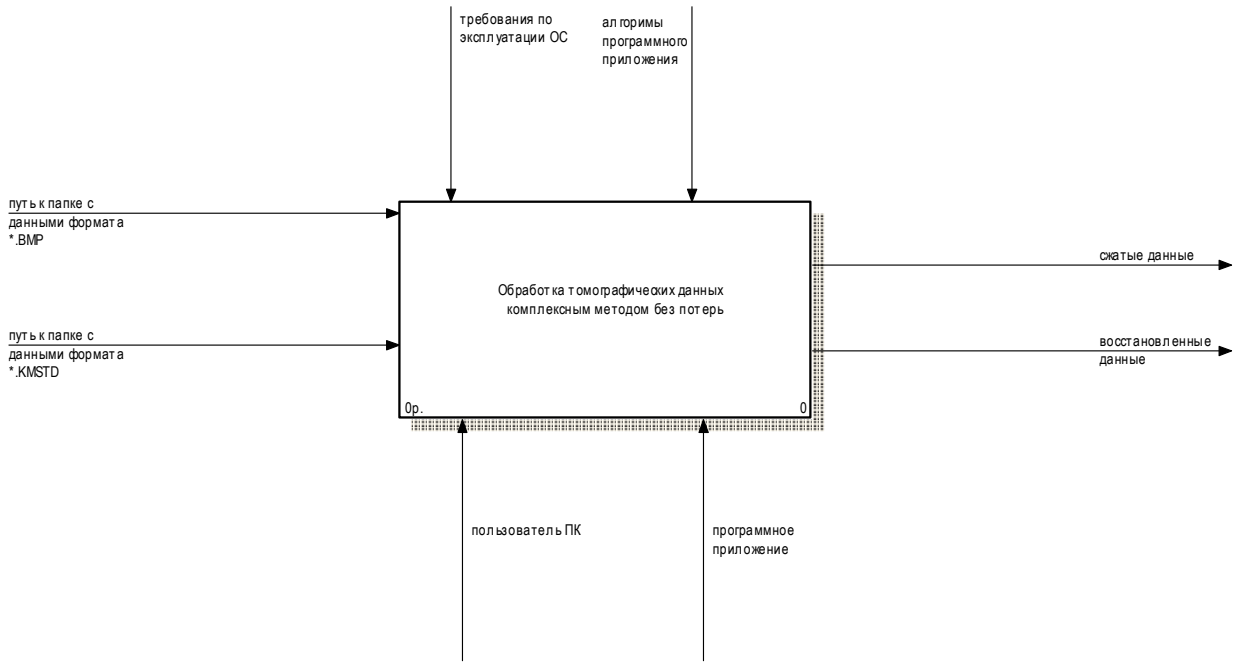


Рисунок 2.15 – Контекстная функциональная диаграмма «Обработка томографических данных комплексным методом без потерь»

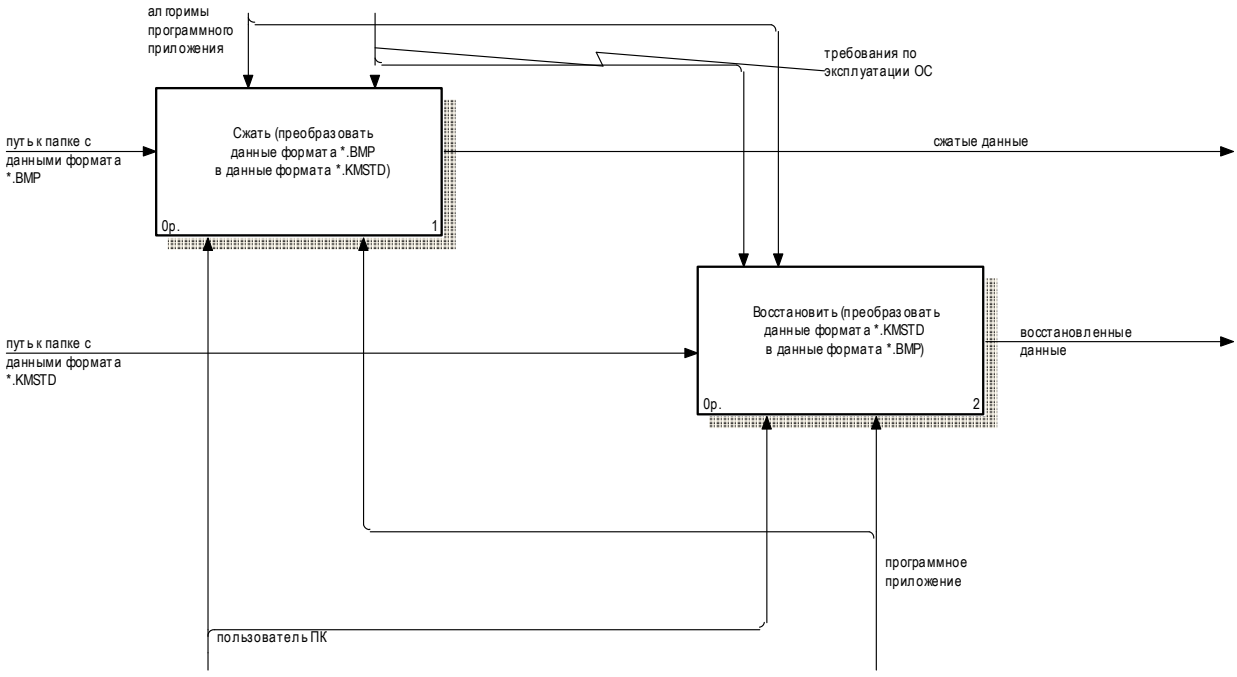


Рисунок 2.16 – Детализация функциональной диаграммы «Обработка томографических данных комплексным методом без потерь»

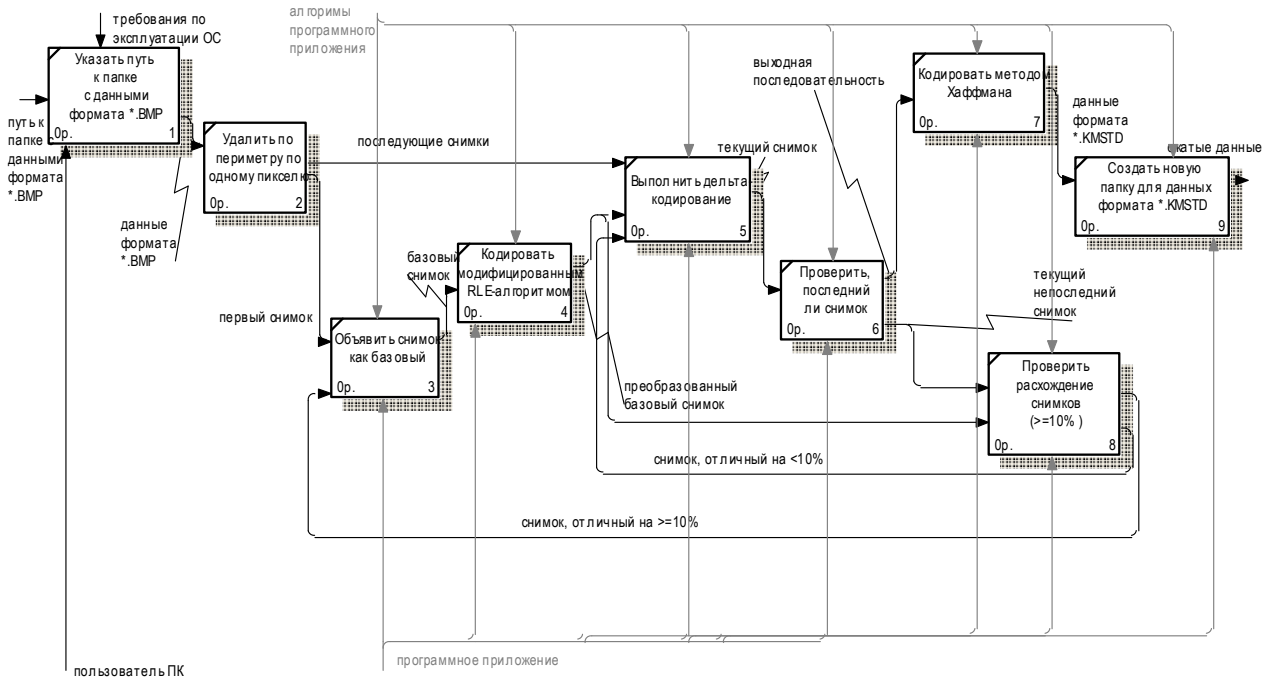


Рисунок 2.17 – Диаграмма декомпозиции «Сжать (преобразовать данные формата *.BMP в данные формата *.KMSTD)»

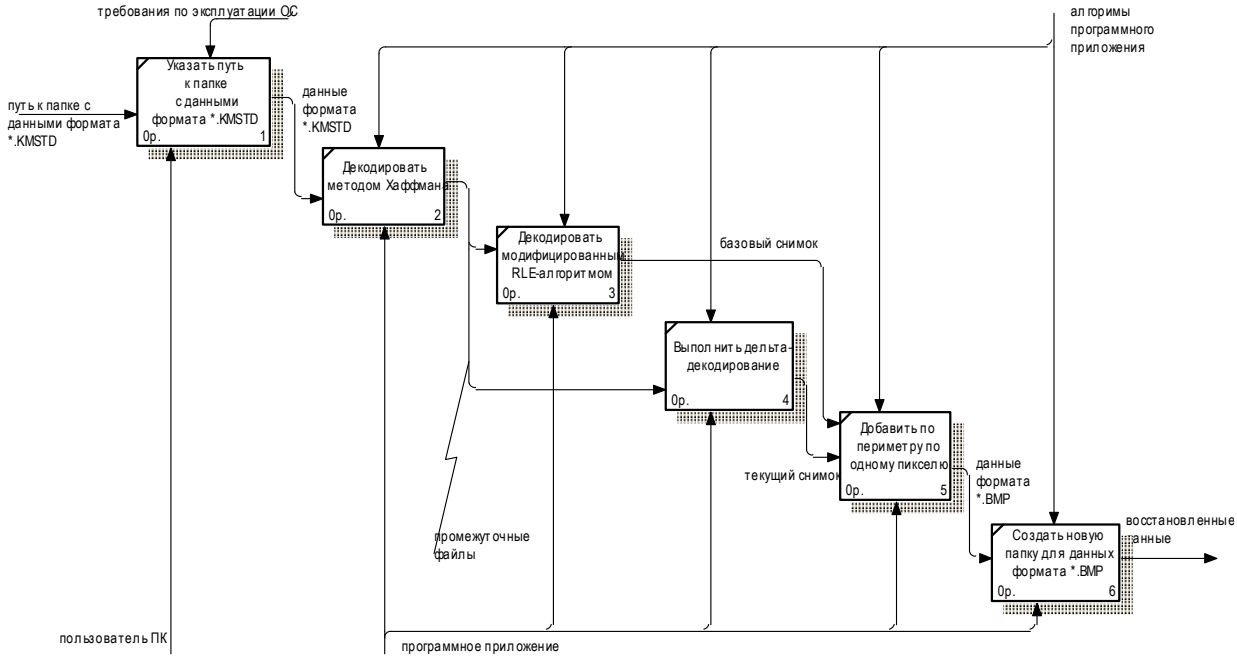


Рисунок 2.18 – Диаграмма декомпозиции «Восстановить (преобразовать данные формата *.KMSTD в данные формата*.BMP)»

Для создания схем и работы с векторной графикой предпочтительно использовать MS Office Visio 2010 [36], посредством чего определена диаграмма переходов состояний (STD), представленная на рисунке 2.19.

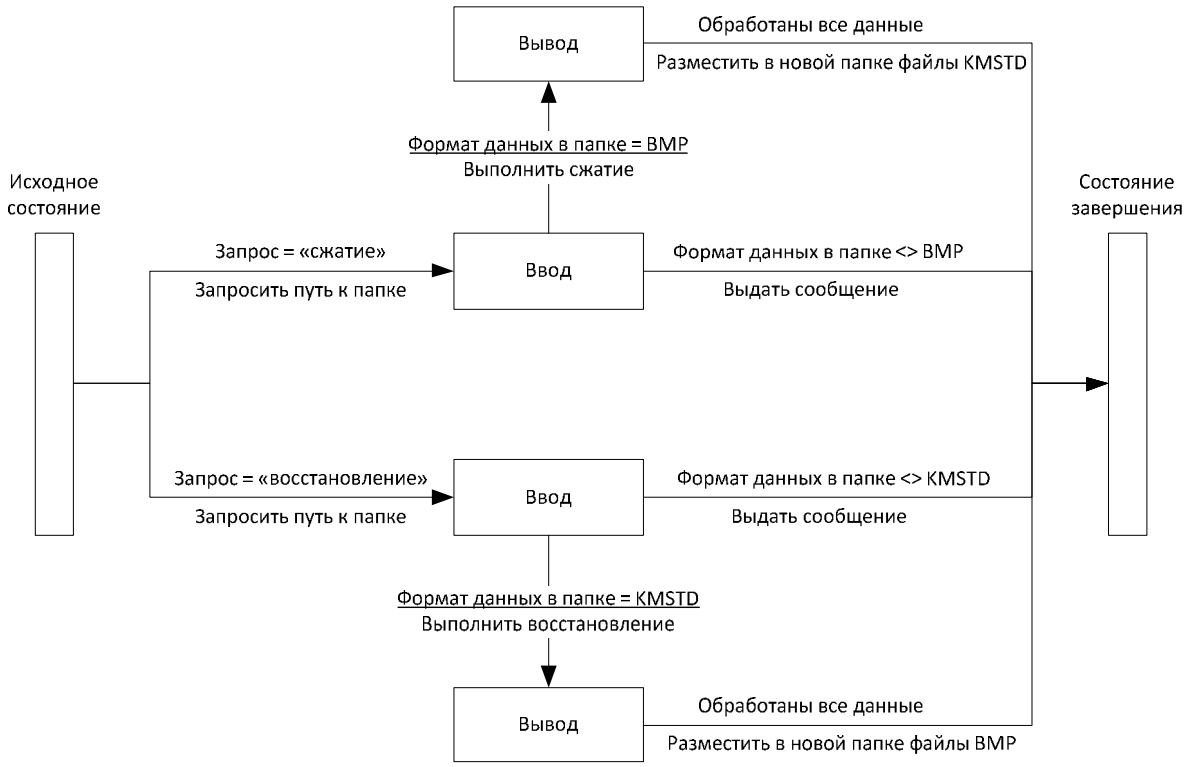


Рисунок 2.19 – Диаграмма переходов состояний «Обработка томографических данных комплексным методом без потерь»

Для построения STD-диаграммы определены узлы (состояния динамической системы) и дуги (переходы системы из одного состояния в другое). С помощью STD-диаграммы продемонстрировано поведение системы при получении управляющих воздействий.

Выводы по второму разделу: на основании вербального моделирования комплексного метода сжатия томографических данных и проектирования его процессов возможна последующая программная реализация.

3 Программная реализация комплексного метода сжатия томографических данных

3.1 Формулировка требований к программному средству

Функциональные требования определяют функциональность программного обеспечения, которую разработчики должны построить, чтобы пользователи смогли выполнить свои задачи в рамках бизнес-требований [37].

Функциональные требования к разрабатываемому программному средству:

- необходимо реализовать программное средство для сжатия и восстановления томографических данных комплексным методом посредством языка программирования – Java;
- в процессе разработки допускается использование сторонних стандартных библиотек;
- программное средство должно предоставлять пользователю возможность выбора операции обработки данных: сжатие или восстановление;
- программное средство должно предоставлять пользователю возможность указания пути размещения данных к обработке;
- максимальное сжатие томографических данных без потерь информации необходимо обеспечить за счет алгоритмов комплексного метода;
- сжатые или восстановленные данные необходимо сохранять в новой папке и размещать ее в папке, указанной в качестве родительской;
- при отсутствии подходящих по формату для обработки данных не выполнять операции по сжатию или восстановлению.

Эксплуатационные требования определяют характеристики разрабатываемого программного обеспечения, проявляемые в процессе его использования, например: правильность, универсальность, надежность, проверяемость, точность результатов, защищенность, программная

совместимость, аппаратная совместимость, эффективность, адаптируемость, повторная входимость, реентерабельность [38].

Эксплуатационные требования к разрабатываемому программному средству:

- программное средство должно обеспечить возможность сжатия томографических данных в формате *.BMP;
- при необходимости пользователь должен самостоятельно преобразовать томографические данные в другом формате в формат *.BMP посредством сторонних программных средств;
- программное средство должно обеспечить возможность восстановления томографических данных в формате *. KMSTD;
- программное средство должно осуществлять сжатие томографических данных без потерь;
- программное средство должно обеспечить правильность восстановления томографических данных;
- программное средство должно работать под управлением операционной системы Windows.

3.2 Разработка компонентов программного средства

Разработка программного средства для сжатия томографических данных комплексным методом выполнена с использованием технологии Java. Программы на Java транслируются в байт-код Java, выполняемый виртуальной машиной Java (JVM) – программой, обрабатывающей байтовый код и передающей инструкции оборудованию как интерпретатор [39].

Окно программного средства «Codec» содержит графические компоненты, представленные в таблице 3.1.

Таблица 3.1 – Использованные графические компоненты

Компонент	Количество	Наименование компонентов	Описание работы компонентов
JButton	2	Выбрать	Кнопка для перехода в окно «Проводника» с последующим выбором обрабатываемых файлов
		Обработать	Кнопка для запуска процесса сжатия томографических данных комплексным методом или восстановления сжатых комплексным методом томографических данных
JRadio Button	2	Кодирование	Выбор процесса сжатия томографических данных комплексным методом
		Декодирование	Выбор процесса восстановления сжатых комплексным методом томографических данных
JText Field	3	Файлы	Отображение наименований выбранных для обработки файлов
		Имя файла (без расширения)	Ввод наименования файла, полученного посредством сжатия томографических данных комплексным методом
JLabel	5	Режим	Вывод информативного текста на форме
		Настройки кодирования	

Первоначальный вид окна программы «Codec» для сжатия томографических данных комплексным методом и восстановления сжатых комплексным методом томографических данных представлен на рисунках 3.1 – 3.2 соответственно.

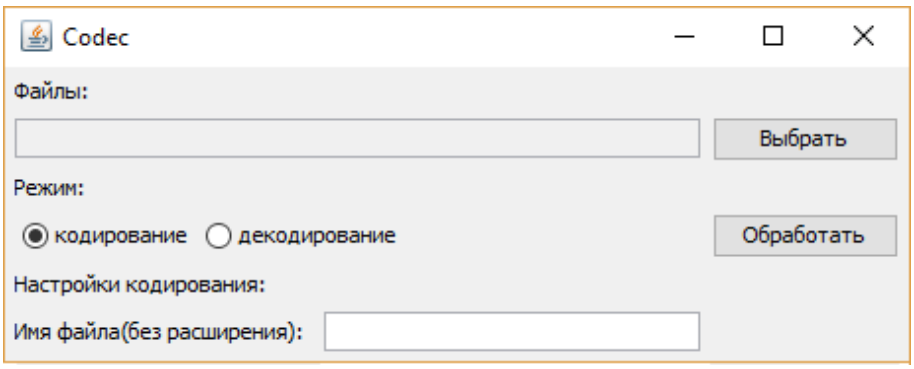


Рисунок 3.1 – Окно программы «Codec» в режиме кодирования

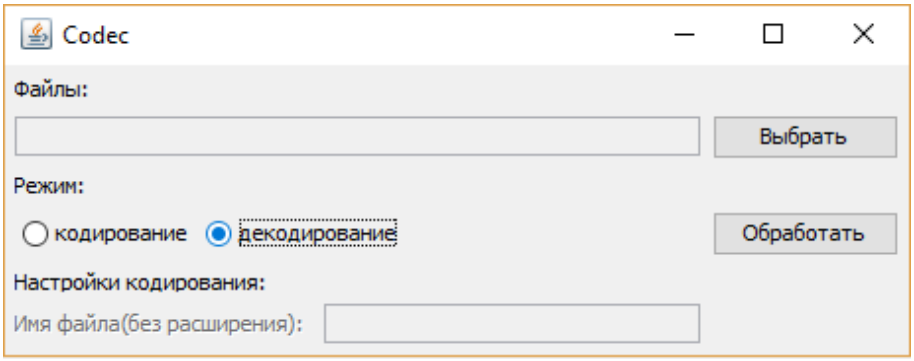


Рисунок 3.2 – Окно программы «Codec» в режиме декодирования

Согласно алгоритму комплексного сжатия томографических данных разработан программный код, обеспечивающий сжатие и восстановление томографических данных посредством программного средства «Codec», представленный в листингах А.1 – А.11 в приложении А.

3.3 Порядок работы с программным средством

Для работы с программным средством для сжатия томографических данных комплексным методом необходимо обеспечить наличие среды выполнения Java на компьютере.

Данные, подлежащие сжатию, должны быть предварительно конвертированы в формат *.BMP и расположены в одной папке на любом диске компьютера. Так, для демонстрации работы с программным средством

подготовлены 22 файла томографических данных, занимающие 15,49 МБ памяти на диске и представленные на рисунках 3.3 – 3.4.

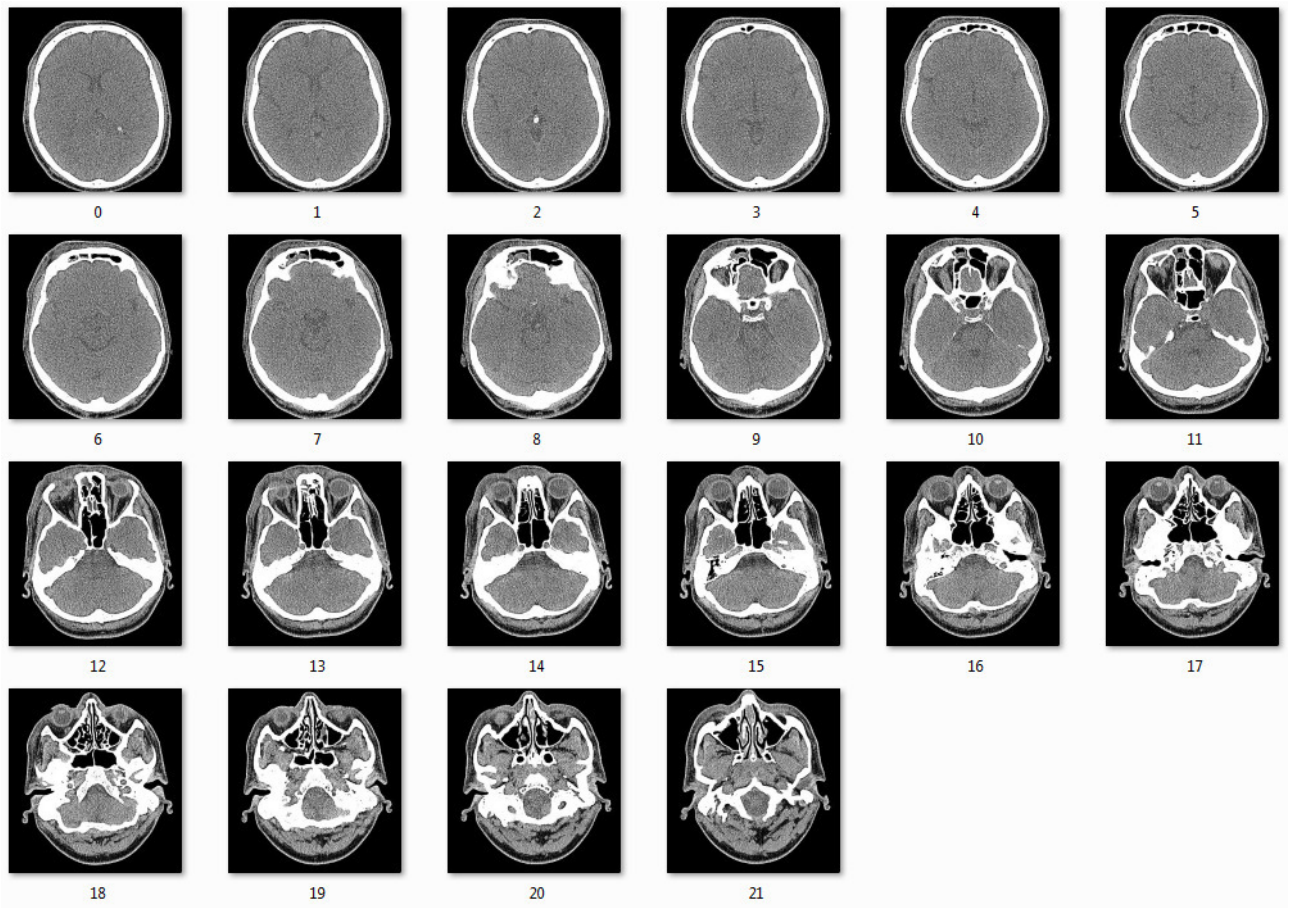


Рисунок 3.3 – Тестовые томографические данные

Запуск программного средства для сжатия томографических данных комплексным методом осуществляется посредством файла «start.bat» в папке «build». В окне программы «Codec» необходимо установить следующие настройки:

- в поле «Файлы» посредством кнопки «Выбрать» перейти в окно «Проводника» и выбрать подготовленные для сжатия томографические данные;
- в поле «Режим» выбрать значение «кодирование»;
- поле «Пороговое значение» оставить без изменений;
- в поле «Имя файла(без расширения)» ввести название сжатого файла.

Результаты выполненной настройки для сжатия томографических данных комплексным методом в окне программы «Codec» представлены на рисунке 3.5.

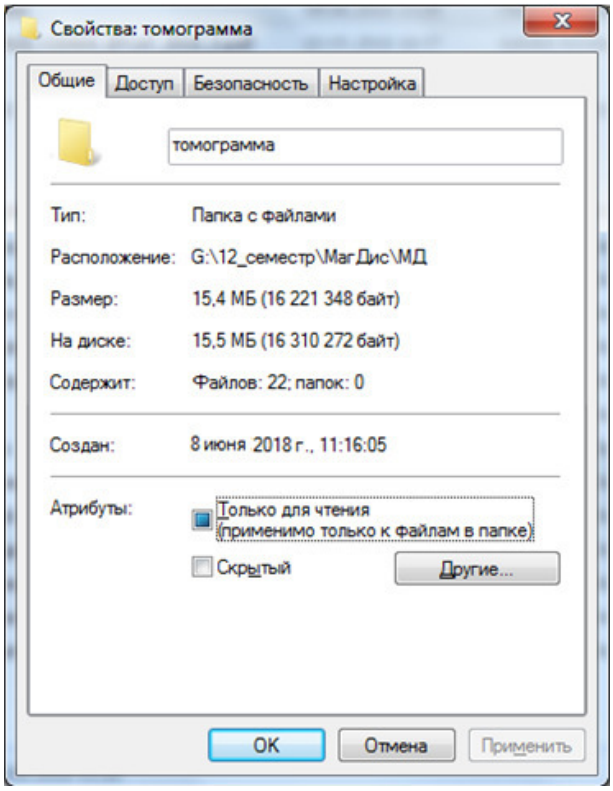


Рисунок 3.4 – Объем памяти, занимаемый томографическими данными

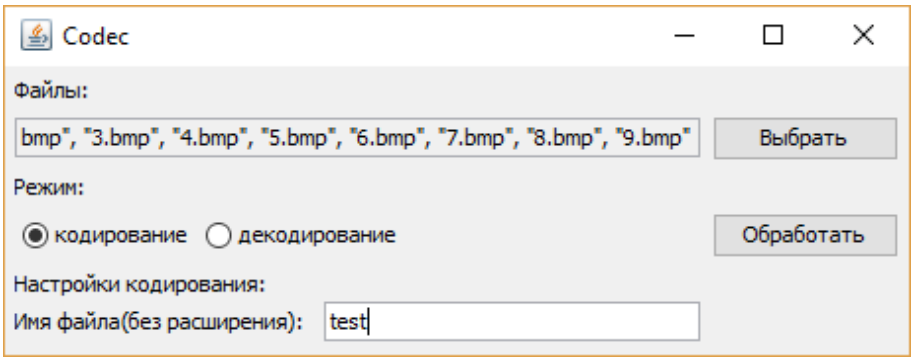


Рисунок 3.5 – Настройка для сжатия томографических данных комплексным методом в окне программы «Codec»

При нажатии кнопки «Обработать» в папке, где находятся томографические данные, подлежащие сжатию, появится файл формата *.KMSTD – результат сжатия томографических данных комплексным методом.

Так, программное средство позволило уменьшить объем занимаемой памяти до 2,46 МБ, что представлено на рисунке 3.6.

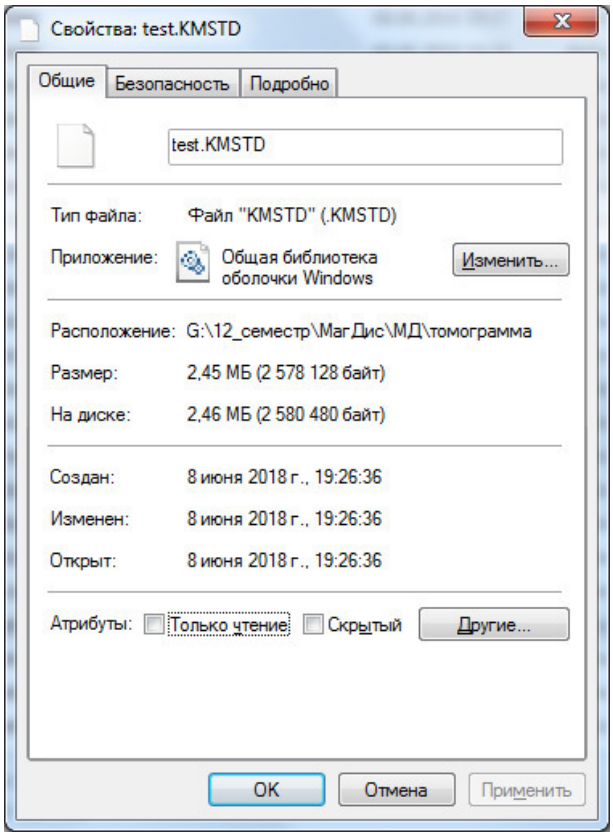


Рисунок 3.6 – Объем памяти, занимаемый сжатыми томографическими данными

В окне программы «Codec» необходимо установить следующие настройки:

- в поле «Файлы» посредством кнопки «Выбрать» перейти в окно «Проводника» и выбрать сжатые томографические данные в формате *.KMSTD;
- в поле «Режим» выбрать значение «декодирование».

Результаты выполненной настройки для восстановления томографических данных комплексным методом в окне программы «Codec» представлены на рисунке 3.7.

При нажатии кнопки «Обработать» в папке, где находятся томографические данные, подлежащие восстановлению, появится папка с

файлами формата *.BMP – результат восстановления томографических данных комплексным методом.

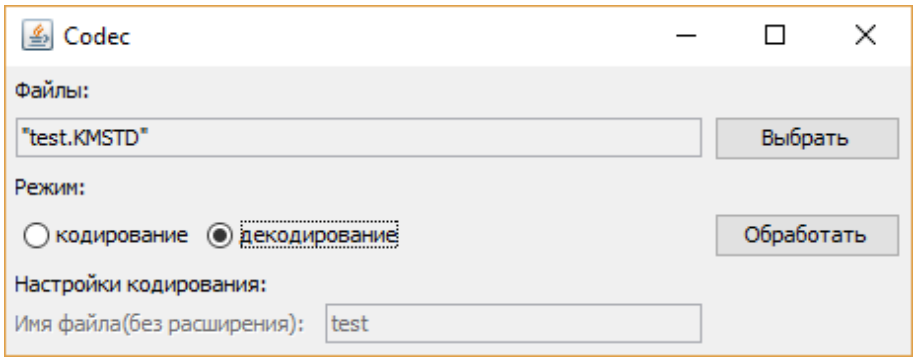


Рисунок 3.7 – Настройка для восстановления томографических данных, сжатых комплексным методом, в окне программы «Codec»

Так, программное средство позволило восстановить томографические данные, сжатые комплексным методом, без потери информации, что представлено на рисунке 3.8.

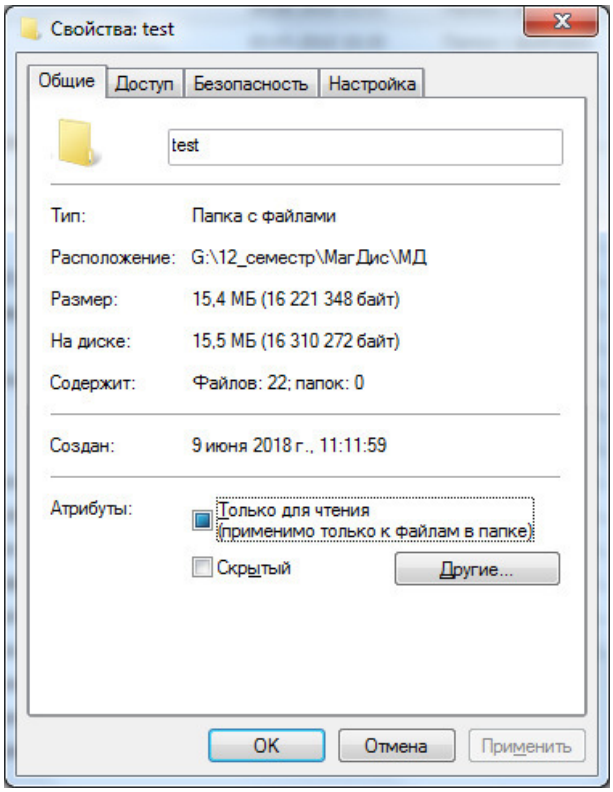


Рисунок 3.8 – Объем памяти, занимаемый восстановленными томографическими данными

Так как объем данных до сжатия равен объему данных, полученных после восстановления файла test.KMSTD, можно утверждать, что разработанный метод и программное средство на его основе соответствуют требованиям и пригодны для дальнейшей эксплуатации.

3.4 Сравнительный анализ результатов сжатия томографических данных

Посредством доступных и наиболее популярных архиваторов осуществлено сжатие томографических данных. Результат сжатия в виде файла в формате *.ZIP представлен на рисунке 3.9.

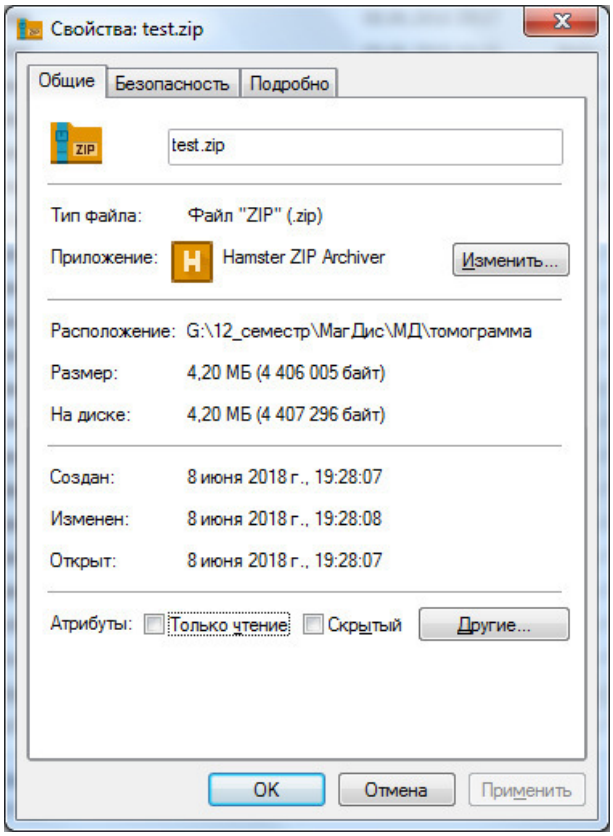


Рисунок 3.9 – Объем памяти, занимаемый данными в формате *.ZIP

Результат сжатия в виде файла в формате *.RAR представлен на рисунке 3.10.

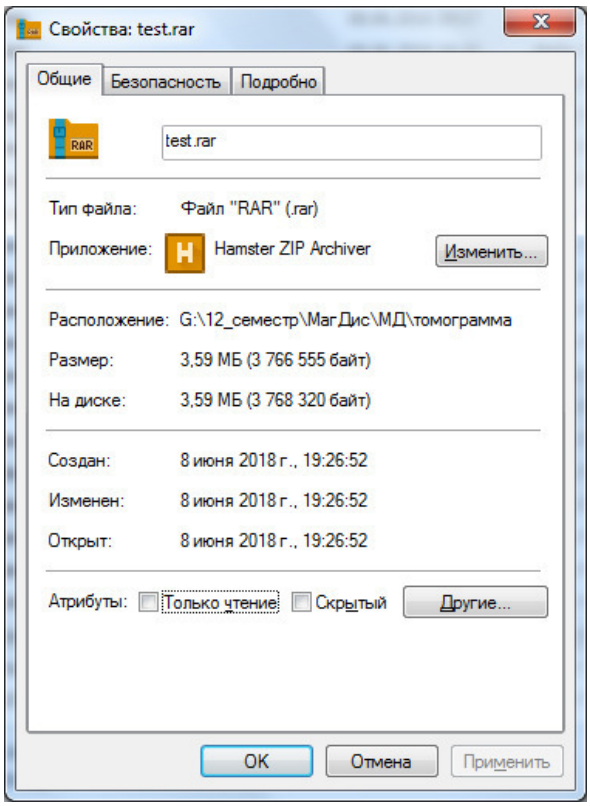


Рисунок 3.10 – Объем памяти, занимаемый данными в формате *.RAR

Результат сжатия в виде файла в формате *.7Z представлен на рисунке 3.11.

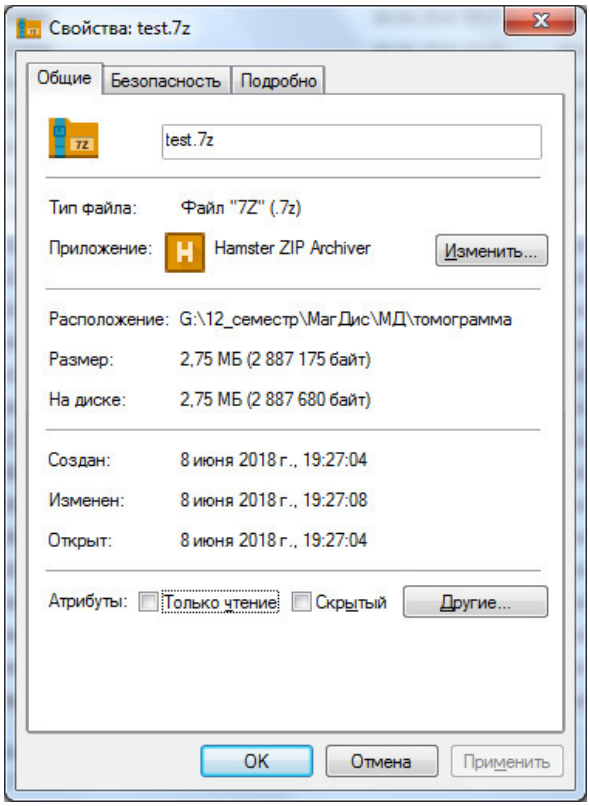


Рисунок 3.11 – Объем памяти, занимаемый данными в формате *.7Z

Основными показателями эффективности той или иной программы-архиватора являются степень сжатия и фактор сжатия файлов [40].

Степень сжатия данных – это отношение размеров сжатого файла и исходного, выраженное в процентах. Степень сжатия рассчитывается по формуле (3.1)

$$K = \frac{C}{И} \cdot 100\%, \tag{3.1}$$

где К – степень сжатия данных, %;

С – объем данных после сжатия, МБ;

И – объем данных до сжатия, МБ.

Степень сжатия отражает, сколько объема данных «осталось» от начального объема после архивации.

Вторым показателем эффективности архивации является фактор сжатия, который показывает, во сколько раз объем данных после сжатия меньше исходного объема данных.

Иными словами, фактор сжатия – это отношение размера исходного файла к размеру сжатого файла, выраженное в единицах, что соответствует формуле (3.2):

$$Ф = \frac{И}{C}, \tag{3.2}$$

где Ф – фактор сжатия данных, ед.;

С – объем данных после сжатия, МБ;

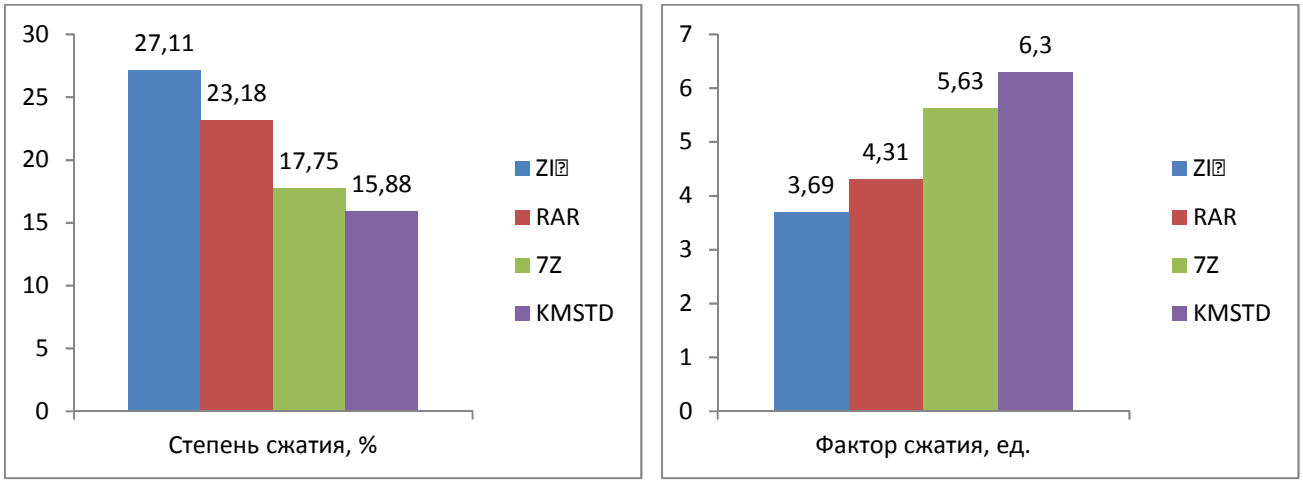
И – объем данных до сжатия, МБ.

Результаты расчета степени сжатия и фактора сжатия томографических данных комплексным методом и архиваторами ZIP, RAR, 7-Zip представлены в таблице 3.2.

Таблица 3.2 – Сравнительный анализ результатов сжатия

Метод сжатия	Исходный размер файла, МБ	Размер сжатого файла, МБ	Восстановленный размер, МБ	Степень сжатия, %	Фактор сжатия, ед.
ZIP	15,49	4,20	15,49	27,11	3,69
RAR	15,49	3,59	15,49	23,18	4,31
7-Zip	15,49	2,75	15,49	17,75	5,63
Комплексный метод	15,49	2,46	15,49	15,88	6,30

Результаты сравнительного анализа, соответствующие таблице 3.2, также представлены на рисунках 3.12,а и 3.12,б.



а б

Рисунок 3.12 – Показатели сжатия томографических данных в сравнении: а) степень сжатия; б) фактор сжатия

Выводы по третьему разделу: преобразование томографических данных в формат *.ZIP обеспечивает уменьшение объема занимаемой памяти в 3,69 раз; посредством архиватора RAR – в 4,31 раз; в формате *.7Z размер данных уменьшился в 5,63 раз; уменьшение объема данных комплексным методом происходит более эффективно – в 6,30 раз.

ЗАКЛЮЧЕНИЕ

В процессе работы над магистерской диссертацией изучена проблема хранения томографических данных, проведен анализ существующих методов сжатия данных без потерь и предложен собственный комплексный метод сжатия томографических данных.

В магистерской диссертации использованы следующие методы исследования:

- системный анализ проблемы сжатия томографических данных;
- моделирование процессов комплексного метода сжатия;
- сравнение результатов сжатия томографических данных комплексным методом и другими существующими подходами.

В итоге разработан комплексный метод сжатия без потерь томографических данных на основании модифицированного RLE-алгоритма, дельта-кодирования и адаптивного метода Хаффмана, благодаря чему достигнута цель исследования по уменьшению объема томографических данных.

Для достижения поставленной цели выполнены следующие задачи:

- изучены проблемы сжатия томографических данных;
- исследованы существующие методы сжатия информации без потерь;
- разработан метод сжатия томографических данных с учетом особенностей обрабатываемой информации;
- разработанный метод описан с помощью диаграмм бизнес-процессов;
- реализовано программное средство на основе разработанного метода сжатия томографических данных;
- описан порядок работы с программным средством в текстовой и иллюстративной форме;
- произведен анализ результатов разработки.

Применение комплексного метода обеспечивает сокращение времени и стоимости передачи информации по каналам связи в компьютерных сетях за

счет более компактного размещения информации на диске ввиду устранения межэлементной и межкадровой избыточности томографических данных, адаптации под полутоновое изображение и специфический размер томограмм.

Сжатие томографических данных комплексным методом обеспечивает уменьшение объема занимаемой памяти в 6,30 раз, что значительно превосходит результаты сжатия аналогичных данных существующими архиваторами. Так, файлы архивации в форматах *.ZIP, *.RAR, *.7Z больше файла в формате *.KMSTD (результат программной реализации комплексного метода) на 41,43%, 31,48%, 10,55% соответственно, а значит комплексный метод сжатия томографических данных более эффективен.

Использование программной реализации комплексного метода сжатия томографических данных возможно как в медицинских учреждениях (для хранения томограмм пациентов в оптимизированном виде), так и на персональных компьютерах пациентов (для передачи томографических данных по каналам связи в рамках онлайн-консультаций).

В качестве перспектив развития программного средства для сжатия томографических данных комплексным методом можно указать такие нереализованные возможности, как обработка непосредственно томограмм в формате DICOM, выполнение программы по действиям из контекстного меню при нажатии правой кнопки мыши.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Богданова, Н.А. Проектирование этапа обработки томографических данных на предмет сжатия комплексным методом без потерь [Текст] / Н.А. Богданова // Электронное периодическое издание «Аллея науки». – 2017. – Выпуск №7. – С. 772-774.

2. Телемедицина.RU – первое профильное СМИ. Телемедицина в мире. Что такое телемедицина [Электронный ресурс]. – Режим доступа: http://telemedicina.ru/news/world/chto-takoe-telemeditsina. – (Дата обращения: 04.05.2018).

3. Шамраева, Е. LOSSLESS-метод сжатия томографических данных [Текст] / Е. Шамраева, А. Шамраев, С. Удовенко // International Journal "Information Technologies & Knowledge". – 2014. – Volume 8, Number 1. – С. 29-32.

4. Википедия – свободная энциклопедия [Электронный ресурс] = Wikipedia / Wikimedia Foundation, Inc. – Магнитно-резонансная томография – Режим доступа: https://ru.wikipedia.org/wiki/Магнитно-резонансная_томография, свободный. – Яз. Мультияз. – (Дата обращения: 17.05.2018).

5. Хофер, М. Компьютерная томография. Базовое руководство. 2-е издание, переработанное и дополненное [Текст] / М. Хофер; пер. с анг. А.П. Кутько, Ф.И. Плешков, В.В. Ипатов, науч. ред. Г.Е. Труфанова. – М.: Мед.лит., 2008. – 224 с.:ил.

6. Богданова, Н.А. Специфика компрессии томографических данных [Электронный ресурс] / Н.А. Богданова // Материалы VIII Международной студенческой электронной научной конференции «Студенческий научный форум». – Режим доступа: http://www.scienceforum.ru/2017/2335/28308. – (Дата обращения: 16.05.2018).

7. Википедия – свободная энциклопедия [Электронный ресурс] = Wikipedia / Wikimedia Foundation, Inc. – Сжатие без потерь – Режим доступа:

https://ru.wikipedia.org/wiki/Сжатие_без_потерь, свободный. – Яз. Мультияз. – (Дата обращения: 07.05.2018).

8. Ватолин, Д. Методы сжатия данных. Устройство архиваторов, сжатие изображений и видео [Текст] / Д. Ватолин, А. Ратушняк, М. Смирнов, В. Юркин. – М.: ДИАЛОГ-МИФИ, 2003. – 384 с.

9. Bourezg, S. RLE Run Length Encoding [Электронный ресурс] / Said Bourezg // File Exchange on the MATLAB Central. – Режим доступа: https://www.mathworks.com/matlabcentral/fileexchange/31123-rle-run-length-encoding?focused=3863374&tab=function. – (Дата обращения: 16.04.2018).

10. НЛО МИР интернет-журнал об НЛО. Технологии. 67 израильских изобретений, которые изменили мир (37 фото) [Электронный ресурс]. – Режим доступа: http://nlo-mir.ru/tech/44673-izobretenij-izmenili-mir.html. – (Дата обращения: 31.03.2018).

11. НОУ ИНТУИТ. Сетевые технологии. Управление ключами шифрования и безопасность сети. Дополнительный материал 13 [Электронный ресурс]. – Режим доступа: https://www.intuit.ru/studies/curriculums/3410/courses/409/lecture/17870?page=1. – (Дата обращения: 23.05.2018).

12. Сэломон, Д. Сжатие данных, изображений и звука [Текст] / Д. Сэломон; пер. с анг. В.В. Чепыжова. – М.: Техносфера, 2004. – 368 с.

13. Texas A&M University Home. Department of Computer Science and Engineering. CSCE 314: Programming Languages [Электронный ресурс]. – Режим доступа: http://robotics.cs.tamu.edu/dshell/cs314/sa6/sa6.html. – (Дата обращения: 30.04.2018).

14. Википедия – свободная энциклопедия [Электронный ресурс] = Wikipedia / Wikimedia Foundation, Inc. – Адаптивный алгоритм Хаффмана – Режим доступа: https://ru.wikipedia.org/wiki/Адаптивный_алгоритм_Хаффмана, свободный. – Яз. Мультияз. – (Дата обращения: 10.04.2018).

15. Qwika. English. En.wikipedia.org [Электронный ресурс]. – Режим доступа: http://wikipedia.qwika.com/en2ru/Adaptive_Huffman_coding. – (Дата обращения: 31.05.2018).

16. Краснящих, А.В. Обработка оптических изображений [Текст] / А.В. Краснящих. – СПб: НИУ ИТМО, 2012. – 129 с.

17. Mohamed, N. Ahmed. Software Research Image Compression [Электронный ресурс] / Mohamed N. Ahmed // SlidePlayer – Upload and Share your PowerPoint presentations. – Режим доступа: http://slideplayer.com/slide/6027450/. – (Дата обращения: 30.05.2018).

18. Википедия – свободная энциклопедия [Электронный ресурс] = Wikipedia / Wikimedia Foundation, Inc. – ZIP – URL: https://ru.wikipedia.org/wiki/ZIP, свободный. – Яз. Мультияз. – (Дата обращения: 11.05.2018).

19. Википедия – свободная энциклопедия [Электронный ресурс] = Wikipedia / Wikimedia Foundation, Inc. – 7Z – Режим доступа: https://ru.wikipedia.org/wiki/7z, свободный. – Яз. Мультияз. – (Дата обращения: 11.05.2018).

20. Википедия – свободная энциклопедия [Электронный ресурс] = Wikipedia / Wikimedia Foundation, Inc. – RAR – Режим доступа: https://ru.wikipedia.org/wiki/RAR, свободный. – Яз. Мультияз. – (Дата обращения: 11.05.2018).

21. Богданова, Н.А. Моделирование программного обеспечения для оптимизации томографических данных комплексным методом без потерь [Текст] / Н.А. Богданова, И.В. Гурьянова // Сборник международной научно-практической конференции «Молодёжный форум: прикладная математика. Математическое моделирование систем и механизмов», посвященной 100-летию со дня рождения С.Г. Крейна. – 2017. – С. 67-70.

22. Помощь школьникам по дисциплинам: математика, физика, биология, химия, черчение, астрономия и информатика. Архиваторы и сжатие информации [Электронный ресурс]. – Режим доступа: http://shkolo.ru/arhivatoryi/. – (Дата обращения: 13.04.2018).

23. Черногорова, Ю.В. Методы сжатия изображений [Электронный ресурс] / Ю.В. Черногорова // Молодой ученый. – Режим доступа: https://moluch.ru/archive/115/30856/. – (Дата обращения: 06.06.2018).

24. Институт проблем передачи информации им. А.А.Харкевича [Электронный ресурс]. – Режим доступа: http://iitp.ru/ru/researchlabs/947.htm. – (Дата обращения: 24.03.2018).

25. Богданова, Н.А. Моделирование сжатия томографических данных комплексным методом без потерь [Текст] / Н.А. Богданова, Е.О. Шамраева // Сборник материалов IX-й международной научно-практической конференции «Компьютерные технологии в моделировании, управлении и экономике». – 2017. – С. 53-58.

26. Богданова, Н.А. Выбор мультипликативным методом анализа иерархий эффективного алгоритма для сжатия томографических данных [Текст] / Н.А. Богданова, Е.О. Шамраева // Сборник материалов I Молодёжной научно-практической конференции с международным участием «Естественнонаучные, инженерные и экономические исследования в технике, промышленности, медицине и сельском хозяйстве». – 2017. – С. 21-23.

27. Оценка экономических величин и управление предприятием: теория и практика для студентов и исследователей. Метод анализа иерархий: процедура применения [Электронный ресурс]. – Режим доступа: http://vamocenka.ru/metod-analiza-ierarxij-procedura-primeneniya/. – (Дата обращения: 04.03.2018).

28. Богданова, Н.А. Выбор эффективного алгоритма для сжатия томографических данных с помощью DSSPROJ [Текст] / Н.А. Богданова, Е.А. Зайцева, С.В. Игрунова // Сборник V Всероссийской (с международным участием) научно-практической конференции студентов, магистрантов, аспирантов и молодых ученых «Культурные тренды современной России: от национальных истоков к культурным инновациям». – 2017. – С. 81-83.

29. Теория и методы принятия решений, а также Хроника событий в Волшебных Странах [Текст]: учебник / О.И. Ларичев. – Москва: Логос, 2000. – 296 с : ил.

30. Константинов, И.И. Иерархия оценки и сравнения заявок участников закупки в строительстве [Электронный ресурс] / И.И. Константинов // Электронный научный журнал «Экономика, управление и инвестиции». – 2014. – №1(3). – Режим доступа: http://euii.esrae.ru/pdf/2014/1(3)/8.pdf. – (Дата обращения: 01.04.2017).

31. Шамраева, Е.О. Комплексный метод компрессии томограмм [Текст] / Е.О. Шамраева // Збірник наукових праць Харківського національного університету Повітряних Сил. – 2014. – Выпуск №2(39). – С. 160-162.

32. Богданова, Н.А. Выбор методом анализа иерархий эффективного алгоритма для сжатия томографических данных [Текст] / Н.А. Богданова, С.В. Игрунова // Сборник материалов Международной научно-практической конференции «Актуальные проблемы развития науки и современного образования». – 2017. – С. 81-83.

33. Богданова, Н.А. Определение спецификаций процессов оптимизации томографических данных комплексным методом [Текст] / Н.А. Богданова, Е.О. Шамраева // Сборник материалов V Молодежной научно-практической конференции «Студенчество России: век XXI». – 2018. – С. 78-84.

34. Маклаков, Сергей. BPwin и Erwin. CASE-средства для разработки информационных систем [Электронный ресурс] / С. Маклаков. – Электрон. текстовые дан. – Москва: Диалог-МИФИ, 2000. – Режим доступа: http://dit.isuct.ru/ivt/books/case/case5/prewords.html. – (Дата обращения: 18.05.2018).

35. НОУ ИНТУИТ. Проектирование информационных систем. Лекция. Моделирование бизнес-процессов средствами BPwin [Электронный ресурс]. – Режим доступа: http://www.intuit.ru/studies/courses/2195/55/lecture/1630. – (Дата обращения: 23.05.2018).

36. Система построения диаграмм и бизнес-схем. Программа векторной графики MS Visio [Электронный ресурс]. – Режим доступа: http://bourabai.ru/einf/visio/gl3.html. – (Дата обращения: 03.04.2018).

37. НеоСофт – IT-решения для учета и управления по всей РФ. Пресс-центр. Программное обеспечение. Управление требованиями. Управление требованиями к программному обеспечению [Электронный ресурс]. – Режим доступа: https://www.neosoft24.ru/news/682/. – (Дата обращения: 05.05.2018).

38. Студенческие реферативные статьи и материалы. Информатика. Технология разработки программного обеспечения. Эксплуатационные требования [Электронный ресурс]. – Режим доступа: https://studref.com/311796/informatika/ekspluatatsionnye_trebovaniya. – (Дата обращения: 05.04.2018).

39. Википедия – свободная энциклопедия [Электронный ресурс] = Wikipedia / Wikimedia Foundation, Inc. – Java – Режим доступа: https://ru.wikipedia.org/wiki/Java, свободный. – Яз. Мультияз. – (Дата обращения: 11.05.2018).

40. Бобродобро – сервис для поиска рефератов, дипломов, курсовых работ. Программы архиваторы информации. Показатель степени сжатия файлов [Электронный ресурс]. – Режим доступа: http://prog.bobrodobro.ru/54405. – (Дата обращения: 21.04.2018).

ПРИЛОЖЕНИЕ А

Программный код

```
package edu.codec;
import edu.codec.gui.MainForm;
import javax.swing.*;
public class Main {
   public static void main(String[] args) throws ClassNotFoundException, UnsupportedLookAndFeelException,
InstantiationException, IllegalAccessException {
      UIManager.setLookAndFeel(
         UIManager.getSystemLookAndFeelClassName());
      MainForm mainForm = new MainForm();
   }
}
```

Листинг А.1 – Файл «Main.java»

```
package edu.codec.processing;
import edu.codec.processing.coding.AbstractCodec;
import edu.codec.processing.coding.DeltaCodec;
import edu.codec.processing.coding.ICodecConstants;
import edu.codec.processing.coding.RLECodec;
import edu.codec.processing.context.Context;
import edu.codec.processing.structure.Node;
import edu.codec.processing.structure.PictureData;
import edu.codec.processing.structure.Plan;
import javax.imageio.ImageIO;
import java.awt.image.BufferedImage;
import java.io.*;
import java.util.*;
import static edu.codec.processing.structure.Plan.buildPlan;
public class Codec implements ICodecConstants {
   private final Context con;
   public Codec() {
      con = Context.getCurrentContext();
   }
   public File[] doEncode(File[] files){
      int count = files.length;
      List<Node> nodeList = new ArrayList<>(count);
      int idx = 0;
      for (File file : files){
         try {
            BufferedImage img = ImageIO.read(file);
            byte[][] data = byteArrayByBufferedImage(img);
            nodeList.add(idx, new Node(idx, file.getName(), new PictureData(data), img.getType()));
            idx++;
         } catch (IOException e) {
            throw new RuntimeException(e);
         }
      }
      Plan plan = buildPlan(nodeList);
      for (Node node : plan){
         AbstractCodec codec;
         if (node.getRelativeDeltaSize() > con.ratio){
            codec = new RLECodec();
         } else {
            codec = new DeltaCodec();
         }
         codec.encode(node);
      }
```

```
        File outFile;
        if (con.encodedFileName != null && !con.encodedFileName.isEmpty()){
            // если явно указали имя файла, то используем его
            // ограничим длину файла до 100 символов
            String fileName = con.encodedFileName.length() > 100 ? con.encodedFileName.substring(0, 100) :
con.encodedFileName;
            outFile = new File(files[0].getParentFile(), fileName + ENCODED_EXTENSION);
            // если с таким именем уже создан, то удалим старый файл
            if (outFile.exists())
                outFile.delete();
        } else {
            // иначе сгенерируем случайное имя файла
            outFile = new File(files[0].getParentFile(), "compressed_" + (int)(Math.random() * Integer.MAX_VALUE) +
ENCODED_EXTENSION);
            while (outFile.exists())    // если такой файл уже есть то создадим с другим именем
                outFile = new File(files[0].getParentFile(), "compressed_" + (int)(Math.random() * Integer.MAX_VALUE) +
ENCODED_EXTENSION);
        }
        try {
            try(ObjectOutputStream out = new ObjectOutputStream(new BufferedOutputStream(new
FileOutputStream(outFile), BUFFER_SIZE))){
                out.writeObject(plan);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
        return new File[]{outFile};
    }
    private static byte[][] byteArrayByBufferedImage(BufferedImage img){
        // пиксели по периметру не берем
        byte[][] data = new byte[img.getWidth() - 2][img.getHeight() - 2];
        for (int i = 1; i < img.getWidth() - 1; i++){
            for (int j = 1; j < img.getWidth() - 1; j++){
                byte pixel = (byte)(img.getRGB(i, j) & 0xFF);
                data[i-1][j-1] = pixel;
            }
        }
        return data;
    }
    public File[] doDecode(File[] files){
        List<File> fileList = new ArrayList<>();
        for (File file : files){
            try(ObjectInputStream in = new ObjectInputStream(new BufferedInputStream(new FileInputStream(file)))) {
                Plan plan = (Plan)in.readObject();
                for (Node node : plan) {
                    AbstractCodec codec;
                    if (node.encodingType == TYPE_RLE)
                        codec = new RLECodec();
                    else
                        codec = new DeltaCodec();
                    codec.decode(node);
                }
                File directory = new File(file.getParentFile(), cutOfExtension(file.getName()));
                if (directory.exists())
                    directory.delete();
                directory.mkdir();
                for (Node node : plan.getNodeList()){
                    BufferedImage img = bufferedImageByByteArray(node);
                    File resultFile = new File(directory, node.name);
                    ImageIO.write(img, "bmp", resultFile);
//                    fileList.add(file);
                }
                fileList.add(directory);
```

```
            } catch (IOException | ClassNotFoundException e) {
                e.printStackTrace();
            }
        }
        return fileList.toArray(new File[fileList.size()]);
    }
    private static BufferedImage bufferedImageByByteArray(Node node){
        byte[][] data = node.pictureData.data;
        BufferedImage img = new BufferedImage(data.length + 2, data[0].length + 2, node.imageType);
        for (int i = 0; i < data.length; i++){
            for (int j = 0; j < data[0].length; j++){
                int x = i + 1;
                int y = j + 1;
                int blue = ((int) data[i][j]) & 0xFF;
                int green = blue << 8;
                int red = blue << 16;
                img.setRGB(x, y, red | green | blue);
            }
        }
        return img;
    }
    private static String cutOfExtension(String fileName){
        int idx = fileName.lastIndexOf(".");
        if (idx > -1)
            return fileName.substring(0, idx);
        else
            return fileName;
    }
}
```

Листинг А.2 – Файл «Codec.java»

```
package edu.codec.processing.coding;
import edu.codec.processing.structure.Node;
public abstract class AbstractCodec implements ICodecConstants{
    public void encode(Node node){
        node.encodingType = getEncodingType();
    }
    public void decode(Node node){
        assert node.getEncodedData()[0] == getEncodingType() &&
                node.encodingType == getEncodingType();
    }
    public abstract byte getEncodingType();
    protected byte[] joinEncodedData(byte[] q1, byte[] q2, byte[] q3, byte[] q4){
        int length = q1.length + q2.length + q3.length + q4.length +
                3 * QUADRANT_SEPARATOR.length + IMAGE_SEPARATOR.length +
ENCODE_TYPE_INFO_LENGTH;
        byte[] result = new byte[length];
        int startIdx = 0;
        result[startIdx] = getEncodingType();
        ++startIdx;
        System.arraycopy(q1, 0, result, startIdx, q1.length);
        startIdx += q1.length;
        System.arraycopy(QUADRANT_SEPARATOR, 0, result, startIdx, QUADRANT_SEPARATOR.length);
        startIdx += QUADRANT_SEPARATOR.length;
        System.arraycopy(q2, 0, result, startIdx, q2.length);
        startIdx += q2.length;
        System.arraycopy(QUADRANT_SEPARATOR, 0, result, startIdx, QUADRANT_SEPARATOR.length);
        startIdx += QUADRANT_SEPARATOR.length;
        System.arraycopy(q3, 0, result, startIdx, q3.length);
        startIdx += q3.length;
        System.arraycopy(QUADRANT_SEPARATOR, 0, result, startIdx, QUADRANT_SEPARATOR.length);
        startIdx += QUADRANT_SEPARATOR.length;
        System.arraycopy(q4, 0, result, startIdx, q4.length);
```

```
        startIdx += q4.length;
        System.arraycopy(IMAGE_SEPARATOR, 0, result, startIdx, IMAGE_SEPARATOR.length);
        startIdx += IMAGE_SEPARATOR.length;
        assert startIdx == length;
        return result;
    }
}
```

Листинг А.3 – Файл «AbstractCodec.java»

```
package edu.codec.processing.structure;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
public class Node implements Iterable<Node> {
    public int idx;
    public String name;
    public PictureData pictureData;
    public final List<Node> childNodes = new ArrayList<>();
    public Node parentNode;
    public int deltaValue = Integer.MAX_VALUE;
    public byte[] encodedData;
    private int encodedDataLength = -1;
    public byte encodingType;
    public int imageType;
    public Node(){}
    public Node(int idx, String name, PictureData pictureData, int imageType){
        this.idx = idx;
        this.name = name;
        this.pictureData = pictureData;
        this.imageType = imageType;
    }
    @Override
    public boolean equals(Object obj) {
        return obj instanceof Node && ((Node) obj).idx == idx;
    }
    @Override
    public String toString() {
        return "#" + idx + " : " +
                name + (parentNode != null ? ("; abs_delta=" + deltaValue + "; rel_delta=" + getRelativeDeltaSize()) : "");
    }
    @Override
    public Iterator<Node> iterator() {
        return childNodes.iterator();
    }
    public int dataSize(){
        return pictureData.size;
    }
    public int deltaValue(){
        return deltaValue;
    }
    public byte[] getEncodedData() {
        return encodedData;
    }
    public int getEncodedDataLength() {
        return encodedDataLength;
    }
    public void setEncodedData(byte[] encodedData) {
        this.encodedData = encodedData;
        this.encodedDataLength = encodedData.length;
    }
    public void setEncodedDataLength(int encodedDataLength) {
        if (this.encodedDataLength < 0)
            this.encodedDataLength = encodedDataLength;
```

```
    }
    public Node getParentNode() {
        return parentNode;
    }
    public void setParentNode(Node parentNode) {
        // если у уже есть родитель, то меняя его, удалим текущий
        // нод из списка дочерних у старого родителя
        if (this.parentNode != null){
            this.parentNode.childNodes.remove(this);
        }
        this.parentNode = parentNode;
        if (parentNode != null)
            parentNode.childNodes.add(this);
    }
    public double getRelativeDeltaSize(){
        return ((double)deltaValue() / (double)dataSize());
    }
}
```

Листинг А.4 – Файл «Node.java»

```
package edu.codec.processing.coding;
import edu.codec.processing.structure.Node;
import edu.codec.processing.structure.PictureData;
import edu.codec.processing.structure.Quadrant;
import java.util.Arrays;
public class DeltaCodec extends AbstractCodec {
    @Override
    public void encode(Node node) {
        super.encode(node);
        byte[] q1 = encodeQuadrant(node, 1);
        byte[] q2 = encodeQuadrant(node, 2);
        byte[] q3 = encodeQuadrant(node, 3);
        byte[] q4 = encodeQuadrant(node, 4);
        node.setEncodedData(joinEncodedData(q1, q2, q3, q4));
    }
    private byte[] encodeQuadrant(Node node, int quadrantNumber){
        Quadrant parentQuadrant = node.getParentNode().pictureData.getQuadrant(quadrantNumber,
DELTA_QUADRANT_IS_ROTATED);
        Quadrant childQuadrant = node.pictureData.getQuadrant(quadrantNumber,
DELTA_QUADRANT_IS_ROTATED);
        byte[][] parentData = parentQuadrant.data;
        byte[][] childData = childQuadrant.data;
        // возьмем с запасом
        int bufferLength = (node.deltaValue() * 4) > (childData.length * childData.length * 4) ?
                (childData.length * childData.length * 4) : (node.deltaValue() * 4);
        byte[] buffer = new byte[bufferLength];
        int bufferIdx = -1;
        for (int i = 0; i < childData.length; i++){
            for (int j = 0; j < childData.length; j++){
                if (childData[i][j] != parentData[i][j]){
                    // Прибавим единицы, чтобы не было коллизий с последовательностью нулей,
                    // означающей конец квадранта(т.к. размер квадранта 255x255, то можем это сделать).
                    // Т.о. если встречаем один 0 - это означает что значение дельты отрицательное;
                    //      если встречаем  два 0 - это означает конец квадранта;
                    //      если встречаем  три 0 - это означает конец изображения.
                    buffer[++bufferIdx] = (byte) (i + 1);
                    buffer[++bufferIdx] = (byte) (j + 1);
                    int childValue = ((int) childData[i][j]) & 0xFF;
                    int parentValue = ((int) parentData[i][j]) & 0xFF;
                    if (childValue > parentValue){
                        buffer[++bufferIdx] = (byte)(childValue - parentValue);
                    } else {
                        buffer[++bufferIdx] = DELTA_MINUS_SIGN;
```

```
                    buffer[++bufferIdx] = (byte)(parentValue - childValue);
                }
            }
        }
    }
    return Arrays.copyOf(buffer, bufferIdx + 1);
}
@Override
public void decode(Node node) {
    super.decode(node);
    int idx = 0;
    node.pictureData = new PictureData();
    idx = decodeQuadrant(node, 1, idx);
    idx = decodeQuadrant(node, 2, idx);
    idx = decodeQuadrant(node, 3, idx);
    idx = decodeQuadrant(node, 4, idx);
    assert (idx + 1) == node.getEncodedData().length;
}
@Override
public byte getEncodingType() {
    return TYPE_DELTA;
}
private int decodeQuadrant(Node node, int quadrantNumber, int lastReadByteIdx){
    Quadrant parentQuadrant = node.getParentNode().pictureData.getQuadrant(quadrantNumber,
DELTA_QUADRANT_IS_ROTATED);          // скопируем квадрант родительского изображения, чтобы
относительно него строился по дельте квадрант дочернего изображения
    Quadrant childQuadrant = parentQuadrant.clone();
    byte[][] result = childQuadrant.data;
    byte[] data = node.getEncodedData();
    // определим какой индекс был прочитан последним
    int idx = lastReadByteIdx;
    while (idx < data.length){
        int i = (((int)data[++idx]) & 0xFF);
        int j = (((int)data[++idx]) & 0xFF);
        if (i == ZERO && j == ZERO){
            // попалось два нуля подряд, проверим является ли следующий элемент нулем
            if (data[idx + 1] == ZERO){
                // если да, то эта последовательность должна указывать на конец изображения {@link
IMAGE_SEPARATOR}
                if (quadrantNumber != 4)
                    throw new RuntimeException("Unexpected three zeros in a row.");
                else if (data.length != (idx + 2))  // если эти три нуля подряд не являются последними элементами в
массиве - что-то пошло не так
                    throw new RuntimeException("Three zeros in a row should be in the end of encoded data array.");
                // "пометим" элемент прочитанным
                ++idx;
            }       // если все ок, то запишем данные квадранта в общий массив изображения
            node.pictureData.setQuadrant(childQuadrant);
            break;
        }
        // отнимем единицы
        i = i - 1;
        j = j - 1;
        int val = ((int)data[++idx]) & 0xFF;
        if (val == DELTA_MINUS_SIGN)
            val = -(((int)data[++idx]) & 0xFF);
        int parentVal = ((int) result[i][j]) & 0xFF;
        result[i][j] = (byte) (val + parentVal);
    }
    return idx;
}
}
```

Листинг А.5 – Файл «DeltaCodec.java»

```
package edu.codec.processing.structure;
import edu.codec.processing.coding.ICodecConstants;
public class PictureData {
   public final byte[][] data;
   public final int size;
   public PictureData(byte[][] data){
      this.data = data;
      size = data.length * data.length;
   }
   public Quadrant getQuadrant(int quadrantNumber, boolean isRotated){
      int width = data.length;
      int height = data[0].length;
      byte[][] result = new byte[width/2][height/2];
      int xFrom = ((quadrantNumber & 1) == 0) ? width/2 : 0;
      int yFrom = (quadrantNumber >= 3) ? height/2 : 0;
      for (int i = 0; i < width/2; i++){
         System.arraycopy(data[i + xFrom], yFrom, result[i], 0, height / 2);
      }
      return new Quadrant(result, quadrantNumber, isRotated);
   }
   public PictureData(){
      this.data = new byte[ICodecConstants.QUADRANT_WIDTH * 2][ICodecConstants.QUADRANT_HEIGHT * 2];
      size = data.length * data.length;
   }
   public PictureData(Quadrant[] quadrants){
      this();
      for (Quadrant q : quadrants) {
         setQuadrant(q);
      }
   }
   public void setQuadrant(Quadrant q){
      int width = data.length;
      int height = data[0].length;
      byte[][] quadrantData;
      if (q.isRotated){
         switch (q.quadrantNumber){
            case 1 : quadrantData = q.data; break;
            case 2 : quadrantData = Quadrant.mirrorVertically(q.data); break;
            case 3 : quadrantData = Quadrant.mirrorHorizontally(q.data); break;
            case 4 : quadrantData = Quadrant.mirrorCollateralDiagonal(q.data); break;
            default: throw new RuntimeException("Wrong quadrant number.");
         }
      } else {
         quadrantData = q.data;
      }
      int xFrom = ((q.quadrantNumber & 1) == 0) ? width/2 : 0;
      int yFrom = (q.quadrantNumber >= 3) ? height/2 : 0;
      for (int i = 0; i < width/2; i++){
         System.arraycopy(quadrantData[i], 0, data[i + xFrom], yFrom, height / 2);
      }
   }
}
```

Листинг А.6 – Файл «PictureData.java»

```
package edu.codec.processing.structure;
import edu.codec.processing.coding.ICodecConstants;
import java.util.Arrays;
import java.util.Iterator;
import java.util.stream.Stream;
public class Quadrant implements Iterable<Byte>{
   public byte[][] data;
   final int quadrantNumber;
```

```
final boolean isRotated;
public Quadrant(int quadrantNumber, boolean isRotated){
   this.data = new byte[ICodecConstants.QUADRANT_WIDTH][ICodecConstants.QUADRANT_HEIGHT];
   this.quadrantNumber = quadrantNumber;
   this.isRotated = isRotated;
}
public Quadrant(byte[][] data, int quadrantNumber, boolean isRotated){
   if (!isRotated || quadrantNumber == 1){
      this.data = data;
   } else if (quadrantNumber == 2){
      this.data = mirrorVertically(data);
   } else if (quadrantNumber == 3){
      this.data = mirrorHorizontally(data);
   } else if (quadrantNumber == 4){
      this.data = mirrorCollateralDiagonal(data);
   }
   this.quadrantNumber = quadrantNumber;
   this.isRotated = isRotated;
}
public static byte[][] mirrorHorizontally(byte[][] data){
   byte[][] result = new byte[data.length][data.length];
   for (int i = 0; i < data.length; i++){
      for (int j = 0; j < data.length; j++){
         result[i][j] = data[i][data.length - 1 - j];
      }
   }
   return result;
}
public static byte[][] mirrorVertically(byte[][] data){
   byte[][] result = new byte[data.length][data.length];
   for (int i = 0; i < data.length; i++){
      System.arraycopy(data[data.length - 1 - i], 0, result[i], 0, data.length);
   }
   return result;
}
public static byte[][] mirrorCollateralDiagonal(byte[][] data){
   byte[][] result = new byte[data.length][data.length];
   for (int i = 0; i < data.length; i++){
      for (int j = 0; j < data.length; j++){
         result[i][j] = data[data.length - 1 - i][data.length - 1 - j];
      }
   }
   return result;
}
@Override
public QuadrantIterator iterator() {
   return new QuadrantIterator();
}
public class QuadrantIterator implements Iterator<Byte> {
   int lastReturnedI = 0;
   int lastReturnedJ = 0;
   int i = 0;
   int j = 0;
   int idxSum = 0;
   boolean isUpDirection = false;
   boolean isEndingPart = false;
   int lastIdxInLine = 0;
   int returnedCount = 0;
   final int arraySize = data.length * data.length;
   @Override
   public boolean hasNext() {
      return returnedCount < arraySize;
   }
```

```
        @Override
        public Byte next() {
            return nextByte();
        }
        public byte nextByte(){
            ++returnedCount;
            lastReturnedI = i;
            lastReturnedJ = j;
            byte value = data[i][j];
            if (!isUpDirection && !isEndingPart && i == idxSum ||
                    !isUpDirection &&  isEndingPart && j == lastIdxInLine ||
                    isUpDirection && !isEndingPart && j == idxSum ||
                    isUpDirection &&  isEndingPart && i == lastIdxInLine) {
                if (idxSum == data.length - 1)
                    isEndingPart = true;
                ++idxSum;
                if (isEndingPart){
                    lastIdxInLine = idxSum % (data.length - 1);
                }
                isUpDirection = !isUpDirection;
                if (isUpDirection) {
                    if (idxSum < data.length)
                        ++i;
                    else
                        ++j;
                } else {
                    if (idxSum < data.length)
                        ++j;
                    else
                        ++i;
                }
            } else if (isUpDirection){
                j = idxSum - --i;
            } else {
                i = idxSum - --j;
            }
            return value;
        }
        public void set(byte value){
            data[lastReturnedI][lastReturnedJ] = value;
        }
    }
    @Override
    public Quadrant clone(){
        Quadrant result = new Quadrant(this.quadrantNumber, this.isRotated);
        result.data = Stream.of(this.data).map(row -> Arrays.copyOf(row, row.length)).toArray(byte[][]::new);
        return result;
    }
}
```

Листинг А.7 – Файл «Quadrant.java»

```
package edu.codec.processing.coding;
public interface ICodecConstants {
    byte TYPE_RLE = 20;
    byte TYPE_DELTA = 30;
    byte ENCODE_TYPE_INFO_LENGTH = 1;
    byte ZERO = 0;
    byte[] QUADRANT_SEPARATOR = new byte[]{ZERO, ZERO};
    byte[] IMAGE_SEPARATOR = new byte[]{ZERO, ZERO, ZERO};
    boolean RLE_QUADRANT_IS_ROTATED = true;
    boolean DELTA_QUADRANT_IS_ROTATED = false;
    int QUADRANT_WIDTH = 255;
    int QUADRANT_HEIGHT = 255;
```

```
    byte DELTA_MINUS_SIGN = 0;
    byte RLE_ZERO = 0;
    String ENCODED_EXTENSION = ".KMSTD";
    String DECODED_EXTENSION = ".BMP";
    int BUFFER_SIZE = 4096;
}
```

Листинг А.8 – Файл «ICodecConstants.java»

```
package edu.codec.processing.coding;
import edu.codec.processing.structure.Node;
import edu.codec.processing.structure.PictureData;
import edu.codec.processing.structure.Quadrant;
import java.util.Arrays;
public class RLECodec extends AbstractCodec {
    @Override
    public void encode(Node node) {
        super.encode(node);
        byte[] q1 = encodeQuadrant(node.pictureData, 1);
        byte[] q2 = encodeQuadrant(node.pictureData, 2);
        byte[] q3 = encodeQuadrant(node.pictureData, 3);
        byte[] q4 = encodeQuadrant(node.pictureData, 4);
        node.setEncodedData(joinEncodedData(q1, q2, q3, q4));
    }
    private byte[] encodeQuadrant(PictureData pictureData, int quadrantNumber){
        Quadrant quadrant = pictureData.getQuadrant(quadrantNumber, RLE_QUADRANT_IS_ROTATED);
        // возьмем с запасом
        byte[] buffer = new byte[quadrant.data.length * quadrant.data.length];
        int bufferIdx = -1;
        int zeroCount = 0;
        Quadrant.QuadrantIterator it = quadrant.iterator();
        while (it.hasNext()){
            byte val = it.nextByte();
            if (val == RLE_ZERO && zeroCount < 255){
                ++zeroCount;
            } else {
                if (zeroCount > 0){
                    buffer[++bufferIdx] = RLE_ZERO;
                    buffer[++bufferIdx] = (byte) zeroCount;
                    if (val == RLE_ZERO){
                        zeroCount = 1;
                    } else {
                        zeroCount = 0;
                        buffer[++bufferIdx] = val;
                    }
                } else {
                    buffer[++bufferIdx] = val;
                }
            }
        }
        // допишем оставшийся кусочек
        if (zeroCount > 0) {
            buffer[++bufferIdx] = RLE_ZERO;
            buffer[++bufferIdx] = (byte) zeroCount;
        }
        return Arrays.copyOf(buffer, bufferIdx + 1);
    }
    @Override
    public void decode(Node node) {
        super.decode(node);
        byte[] data = node.getEncodedData();
        int idx = 0;
        int zerosInARow = 0;
        byte val;
```

```
        int quadrantNumber = 1;
        Quadrant q = new Quadrant(quadrantNumber, RLE_QUADRANT_IS_ROTATED);
        Quadrant.QuadrantIterator it = q.iterator();
        Quadrant[] quadrants = new Quadrant[4];
        quadrants[quadrantNumber - 1] = q;
        while (idx < data.length ){
            if (!it.hasNext()){//квадрант кончился
                idx = idx - zerosInARow;
                if (quadrantNumber < 4){//это последний квадрант?
                    byte[] separator = {data[++idx], data[++idx]};
                    if (!Arrays.equals(separator, QUADRANT_SEPARATOR)) {
                        throw new RuntimeException("Expected " + Arrays.toString(QUADRANT_SEPARATOR) +
                                " at index " + (idx - 1) + " of encoded sequence of " + node.name + ".");
                    }
                    q = new Quadrant(++quadrantNumber, RLE_QUADRANT_IS_ROTATED);
                    it = q.iterator();
                    quadrants[quadrantNumber - 1] = q;
                } else {//проверим, что это действительно конец последовательности
                    byte[] separator = {data[++idx], data[++idx], data[++idx]};
                    assert Arrays.equals(separator, IMAGE_SEPARATOR);
                    break;
                }
            }
            val = data[++idx];
            if (val == RLE_ZERO){
                ++zerosInARow;
                if (idx < data.length){
                    val = data[++idx];
                    if (val == RLE_ZERO)
                        ++zerosInARow;
                }
                if (isEncodedZeroSequence(zerosInARow)){
                    int count = ((int)val) & 0xFF;
                    for (int n = 0; n < count; n++){
                        assert it.hasNext();
                        it.nextByte();
                        it.set(RLE_ZERO);
                    }
                    zerosInARow = 0;
                }
            } else {
                assert it.hasNext();
                it.nextByte();
                it.set(val);
            }
        }
        node.pictureData = new PictureData(quadrants);
    }
    @Override
    public byte getEncodingType() {
        return TYPE_RLE;
    }
    private boolean isEncodedZeroSequence(int zerosInARow){
        return zerosInARow == 1;
    }
}
```

Листинг А.9 – Файл «RLECodec.java»

```
package edu.codec.processing.context;
public class Context {
    private final static ThreadLocal<Context> threadLocal = new ThreadLocal<>();
    private Context(){}
    public double ratio;
```

```
    public String encodedFileName;
    public static Context createContext(){
        Context con = new Context();
        threadLocal.set(con);
        return con;
    }
    public static Context getCurrentContext(){
        return threadLocal.get();
    }
}
```

Листинг А.10 – Файл «Context.java»

```
package edu.codec.processing.structure;
import edu.codec.processing.coding.ICodecConstants;
import java.io.*;
import java.nio.ByteBuffer;
import java.nio.charset.StandardCharsets;
import java.util.*;
import java.util.stream.Collectors;
import java.util.stream.Stream;
import java.util.zip.*;
public class Plan implements Iterable<Node>, Externalizable{
    private static final long serialVersionUID = -4979222771682063908L;
    public Node rootNode;
    private List<Node> nodeList;
    public Plan(){}
    public Plan(Node rootNode, List<Node> nodeList) {
        this.rootNode = rootNode;
        this.nodeList = nodeList;
    }
    public static Plan buildPlan(List<Node> nodeList){
        int count = nodeList.size();
        int[][] deltaSizeArray = new int[count][count];
        nodeList.parallelStream().forEach(left -> {
            int x = left.idx;
            nodeList.parallelStream().forEach(right -> {
                int y = right.idx;
                deltaSizeArray[x][y] = (x >= y) ? Integer.MAX_VALUE : calcDeltaSize(left.pictureData.data,
right.pictureData.data);
            });
        });
        final Map<Integer, Node> map = new HashMap<>();
        nodeList.forEach(node -> map.put(node.idx, node));
        int idx = 0;
        while (!map.isEmpty()/*map.size() > 1*/){
            Node node = map.remove(idx);
            int x = 0, y = 0, newNodeIdx = 0;
            int value = Integer.MAX_VALUE;
            Node childNode;
            // если в мапе остался 1 нод, то проставим его в качействе чаилда,
            // а в следующий проход найдем ему подходящего парента
            if (map.size() == 1){
                childNode = map.values().iterator().next();
                if (childNode.idx > idx){
                    x = idx; y = childNode.idx;
                } else {
                    x = childNode.idx; y = idx;
                }
                value = deltaSizeArray[x][y];
                newNodeIdx = childNode.idx;
            } else {
                // ищем по строке
                for (int i = 0; i < idx; i++) {
```

```
                if (value > deltaSizeArray[i][idx]) {
                    x = i;
                    y = idx;
                    value = deltaSizeArray[i][idx];
                    newNodeIdx = i;
                }
            }
            // ищем по столбцу
            for (int i = idx + 1; i < count; i++) {
                if (value > deltaSizeArray[idx][i]) {
                    x = i;
                    y = idx;
                    value = deltaSizeArray[idx][i];
                    newNodeIdx = i;
                }
            }
            childNode = map.get(newNodeIdx);
        }
        // если не нашли чайлда, то заменим парента на вариант получше
        if (childNode != null){
            childNode.deltaValue = value;
            childNode.setParentNode(node);
            deltaSizeArray[x][y] = Integer.MAX_VALUE;
            idx = newNodeIdx;
        } else if (value < node.deltaValue) {
            // todo: сюда зайдет только 1 раз, когда мапа опустошится
            Node newParentNode = nodeList.get(newNodeIdx);
            Node oldParentNode = node.getParentNode();
            node.setParentNode(newParentNode);
            int oldValue = node.deltaValue;
            node.deltaValue = value;
            if (oldParentNode.idx > node.idx)
                deltaSizeArray[node.idx][oldParentNode.idx] = oldValue;
            else
                deltaSizeArray[oldParentNode.idx][node.idx] = oldValue;
            deltaSizeArray[x][y] = Integer.MAX_VALUE;
            // возвращаем нод в мапу, если это был не последний нод
            if (!map.isEmpty())
                map.put(node.idx, node);
        } else {
            deltaSizeArray[x][y] = Integer.MAX_VALUE;
            // возвращаем нод в мапу, если это был не последний нод
            if (!map.isEmpty())
                map.put(node.idx, node);
        }
    }
    return new Plan(nodeList.get(0), nodeList);
}
private static int calcDeltaSize(byte[][] left, byte[][] right){
    int size = 0;
    for (int i = 0; i < left.length; i++)
        for (int j = 0; j < left[0].length; j++)
            if (left[i][j] != right[i][j]) size++;
    return size;
}
@Override
public Iterator<Node> iterator() {
    return new PlanIterator();
}
public class PlanIterator implements Iterator<Node>{
    private Node next = rootNode;
    private final Deque<Iterator<Node>> stack = new ArrayDeque<>();
    private Iterator<Node> currentIterator = null;
```

```
        @Override
        public boolean hasNext() {
            return next != null || currentIterator != null && currentIterator.hasNext();
        }
        @Override
        public Node next() {
            Node aNext = next;
            next = null;
            currentIterator = aNext.iterator();
            while (next == null && currentIterator != null){
                if (currentIterator.hasNext()) {
                    next = currentIterator.next();
                    stack.addFirst(currentIterator);
                } else {
                    currentIterator = stack.pollFirst();
                }
            }
            return aNext;
        }
    }
    @Override
    public void writeExternal(ObjectOutput out) throws IOException {
        Iterator<Node> it = iterator();
        // сначала запишем метаданные о наборе файлов, укажем количество файлов
        out.write(ByteBuffer.allocate(4).putInt(nodeList.size()).array());
        while (it.hasNext()){
            Node node = it.next();
            byte[] name = node.name.getBytes(StandardCharsets.UTF_8);
            int bufferSize =
                    4 + // идентификатор нода
                    4 + // идентификатор родительского нода
                    4 + // длина имени в байтах
                    name.length +  // имя нода
                    4 + // тип цветовой схемы изображения
                    4;  // длина в байтах закодированной последовательности (без учета кодирования Хаффмана)
            ByteBuffer buffer = ByteBuffer.allocate(bufferSize);
            buffer.putInt(node.idx);
            buffer.putInt(node.getParentNode() != null ? node.getParentNode().idx : -1);
            buffer.putInt(name.length);
            buffer.put(name);
            buffer.putInt(node.imageType);
            buffer.putInt(node.getEncodedDataLength());
            out.write(buffer.array());
            out.flush();
        }
        // теперь запишем сжатые изображения
        ObjectOutputStream objectOutputStream = (ObjectOutputStream) out;
        int bufferSize = ICodecConstants.BUFFER_SIZE;
        byte[] buffer = new byte[bufferSize];
        DeflaterOutputStream deflaterOutputStream = new DeflaterOutputStream(objectOutputStream, new
Deflater(Deflater.HUFFMAN_ONLY), bufferSize);
        it = iterator();
        while (it.hasNext()){
            Node node = it.next();
            ByteArrayInputStream byteArrayInputStream = new ByteArrayInputStream(node.getEncodedData());
            int readCount;
            while (byteArrayInputStream.available() > 0){
                readCount = byteArrayInputStream.read(buffer);
                deflaterOutputStream.write(Arrays.copyOfRange(buffer, 0, readCount));
            }
        }
        deflaterOutputStream.finish();
    }
```

```
    @Override
    public void readExternal(ObjectInput in) throws IOException, ClassNotFoundException {
        int filesCount = in.readInt();
        int filesMetaDataReadCount = 0;
        // создадим пустые ноды в нужном количестве
        nodeList = Stream.generate(Node::new).limit(filesCount).collect(Collectors.toList());
        // заполним их метаданными
        while (filesMetaDataReadCount < filesCount){
            int idx = in.readInt();   // получаем идентификатор нода
            Node node = nodeList.get(idx);   // достанем нод из списка
            node.idx = idx;
            int parentIdx = in.readInt();   // установим у него родительский нод, если таковой имеется
            if (parentIdx > -1)
                node.setParentNode(nodeList.get(parentIdx));
            byte[] name = new byte[in.readInt()];   // получим имя
            in.read(name);
            node.name = new String(name, StandardCharsets.UTF_8);
            node.imageType = in.readInt();
            node.setEncodedDataLength(in.readInt());   //установим ожидаемое количество закодированных байт
данных (без кодирования Хаффмана)
            ++filesMetaDataReadCount;
        }
        rootNode = nodeList.get(0);
        Iterator<Node> it = iterator();
        int bufferSize = ICodecConstants.BUFFER_SIZE;
        byte[] buffer = new byte[bufferSize];
        ObjectInputStream objectInputStream = (ObjectInputStream)in;
        InflaterInputStream inflaterInputStream = new InflaterInputStream(objectInputStream, new Inflater(), bufferSize);
        // Получим из потока байт закодированные последовательности
        while (it.hasNext()){
            Node node = it.next();
            int dataLength = node.getEncodedDataLength();
            ByteArrayOutputStream byteArrayInputStream = new ByteArrayOutputStream(dataLength);
            int totalReadBytesCount = 0;
            // читаем, пока не получим весь массив согласно заданной длине
            while (totalReadBytesCount < dataLength){
                int bytesToRead = (dataLength - totalReadBytesCount) > bufferSize ? bufferSize : dataLength -
totalReadBytesCount;     // не всегда возвращает readBytesCount == bytesToRead, даже если поток не исчерпан,
вызвано тем, что за один read не весь запрашиваемый набор байт может быть раскодирован
                int readBytesCount = inflaterInputStream.read(buffer, 0, bytesToRead);
                if (readBytesCount > 0) {
                    byteArrayInputStream.write(buffer, 0, readBytesCount);
                    totalReadBytesCount = totalReadBytesCount + readBytesCount;
                }
            }
            byte[] encodedData = byteArrayInputStream.toByteArray();
            byte[] ending = Arrays.copyOfRange(encodedData, encodedData.length -
ICodecConstants.IMAGE_SEPARATOR.length, encodedData.length);
            if (!Arrays.equals(ICodecConstants.IMAGE_SEPARATOR, ending))
                throw new RuntimeException("Image separator " +
Arrays.toString(ICodecConstants.IMAGE_SEPARATOR) +
                        " expected. Met : " + Arrays.toString(ending) + ".");
            node.setEncodedData(encodedData);
            node.encodingType = encodedData[0];
        }
        if (inflaterInputStream.read() != -1 && inflaterInputStream.available() > 0)
            throw new RuntimeException("Broken data input. Stream has some bytes to read.");
    }
    public List<Node> getNodeList() {
        return nodeList;
    }
}
```

Листинг А.11 – Файл «Plan.java»

Магистерская диссертация выполнена мной совершенно самостоятельно. Все использованные в работе материалы и концепции из опубликованной научной литературы и других источников имеют ссылки на них.

«___» _____ _____ г.

_____ _____

(подпись) *(Ф.И.О.)*