Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ **Информатика и системы управления**

КАФЕДРА **Компьютерные системы и сети**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.01 Информатика и вычислительная техника**

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ БАКАЛАВРА НА ТЕМУ:

ПОДСИСТЕМА СЖАТИЯ ДАННЫХ С ИСПОЛЬЗОВАНИЕМ РАЗНОСТНЫХ АЛГОРИТМОВ

Студент	ИУ6-82Б		Р.Д. Векшин
	(Группа)	(Подпись, дата)	(И.О. Фамилия)
Руководитель			Г.П. Можаров
		(Подпись, дата)	(И.О. Фамилия)
Нормоконтролер			О.Ю. Ерёмин
		(Подпись, дата)	(И.О. Фамилия)

2020 г.

АННОТАЦИЯ

В данной выпускной квалификационной работе представлен процесс проектирования и разработки подсистемы сжатия данных с использованием разностных алгоритмов. Работа состоит из введения, основной части, представленной тремя частями, заключения, списка использованных источников и приложений.

Во введении представлена актуальность анализируемой темы выпускной квалификационной работы, показаны цели и краткое описание конечного продукта.

В части «Исследование принципов работы аналогичных подсистем» представлена информация о существующих аналогах подсистемы сжатия данных, описан набор выполняемых функций, основные принципы и алгоритмы их работы.

В части «Разработка подсистемы сжатия» описано проектирование программной подсистемы, разработка структурных схем, схемы алгоритма сжатия данных, схемы компоновки программы.

В части «Разработка технологии тестирования подсистемы» представлены разработанные технологии тестирования подсистемы сжатия данных, приведены примеры тестов, проведено оценочное тестирование программной подсистемы.

В заключении приведены основные выводы, полученные в результате проведённого исследования, проектирования и реализации подсистемы сжатия данных.

Общий объем работы: 59 страниц.

Ключевые слова: алгоритм, избыточность, наибольшая общая последовательность, преобразование, разностные алгоритмы, расстояние Левенштейна, сжатие, сжатие данных, эффективность.

РЕФЕРАТ

РПЗ 59 с., 10 рис., 7 таб., 9 ист., 5 прил.

Ключевые слова: алгоритм, избыточность, наибольшая общая последовательность, преобразование, разностные алгоритмы, расстояние Левенштейна, сжатие, сжатие данных, эффективность.

Целью данной работы является проектирование и разработка подсистемы сжатия данных с использованием разностных алгоритмов. Данный программный продукт предназначен для использования в системах приема и передачи данных с целью сжатия данных.

Программная подсистема представляет собой подключаемый модуль (библиотеку) для компилируемого многопоточного языка программирования Golang. Подсистема обеспечивает ускорение передачи данных при лимитированном сетевом соединении при передачи данных между электронно-вычислительными машинами и комплексами и экономию места на дисковых носителях при хранении информации за счёт уменьшения размера данных и может быть внедрена в любое программное обеспечение, написанное на языке программирования Golang. Вместе с программной подсистемой разработан набор утилит для контроля и тестирования подсистемы в автономном режиме с использованием виртуального терминала.

Особенностью данной подсистемы сжатия является использование алгоритмов, демонстрирующих хорошую степени сжатия на данных, содержащих регулярную информацию, при небольшом размере используемой оперативной памяти. Подсистема сжатия данных на основе разностных алгоритмов также актуальна для систем, хранящих версионированные версии файлов, данные в которых меняются небольшими относительными объемами.

СОДЕРЖАНИЕ

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

Дедупликация — Исключение дублирующих фрагментов или объектов

Декодер — Программа или алгоритм декодирования (разжатия) данных

Кодек — Термин для совместного обозначения кодера и декодера

Кодер — Программа или алгоритм кодирования (сжатия) данных

МОП — Максимальная общая последовательность

НКРЯ — Национальный корпус русского языка

СУБД — Система управления базами данных

СХД — Система хранения данных

API (Application programming interface) — Интерфейс прикладного программирования

CRC (Cyclic redundancy check) — Циклический избыточный код

EOF (End of file) — Сигнал «конец файла»

GOP (Group of pictures) — Группы изображений

RLE (Run-length encoding) — Кодирование длин серий

S3 (Simple Storage Service) — Облачное объектное хранилище

ВВЕДЕНИЕ

В современном мире можно наблюдать ежегодный рост объемов передаваемой по сети и хранимой на носителях информации. В настоящее время огромной популярностью пользуется концепция Интернета вещей. Множество датчиков и систем обмениваются между собой информацией, которая, лишь незначительно отличается от информации, которую посылали эти же устройства ранее. Невооруженным взглядом видно, что для долгосрочного хранения этой информации в исходном виде необходимы большие объемы систем хранения данных (далее СХД). В случае нехватки пространств для хранения информации приходится искать способы их экономии. Один из способов — сжатие данных.

Существуют различные методы и алгоритмы, обеспечивающие сжатие информации. Эти алгоритмы позволяют значительно разгрузить каналы связи и СХД за счет исключения ненужных или дублированных блоков данных (дедупликация), что приводит к увеличению пропускных способностей систем сбора, передачи и обработки данных или увеличению емкости устройств хранения, а следовательно и к уменьшению стоимости их поддержки и обслуживания.

Если известно, что большая часть информации повторяется от одного фрагмента к другому, а это имеет место быть в случае наличия четкой структуры данных, то можно реализовать сжатие за счет сохранения только лишь разницы между этими фрагментами данных. Применяя разностный алгоритм сжатия, можно достичь высокой степени компрессии такой информации и существенно сэкономить пропускную способность канала связи или место в СХД.

1 Исследование принципов работы аналогичных подсистем

1.1 Анализ аналогов

1.1.1 Статистические методы

Кодирование с использованием статистики (энтропии) — кодирование последовательности значений с возможностью однозначного восстановления с целью уменьшения объёма данных (длины последовательности) с помощью усреднения вероятностей появления элементов в закодированной последовательности.

Рисунок 1 - Частотность букв русского языка по НКРЯ

Предполагается, что до кодирования отдельные элементы последовательности имеют различную вероятность появления особенно это выражено в текстах (см. пример для букв русского языка на рис. 1). После кодирования в результирующей последовательности вероятности появления отдельных символов практически одинаковы (энтропия на символ максимальна), т.е. для символов с максимальной частотностью необходимо брать коды минимальной длины, а для минимальной частотности — максимальной [1].

1.1.1.1 Алгоритм Хаффмана

Идея, положенная в основу алгоритма Хаффмана, основана на частоте появления символа в последовательности. Символ, который встречается в исходном тексте больше всего, получает небольшой код, а символ, который встречается меньше всего, получает длинный код. Это необходимо, т. к. следует стремиться к уменьшению кодированной последовательности и самые часто встречаемые символы занимали меньше места, чем они занимали в исходной последовательности, а редко встречаемые — больше, но с учётом их редкости не сильно увеличивали суммарную длину закодированной последовательности.

Для нахождения этих кодов необходимо построить очередь с приоритетом символами из исходного текста. Приоритетом символа будет являться его частота появления в исходной последовательности. Далее, начиная с символа с наименьшим приоритетом, строится бинарное дерево.

На рисунке 2 представлено дерево Хаффмана для символов, с приоритетами, указанными на рисунке 1.

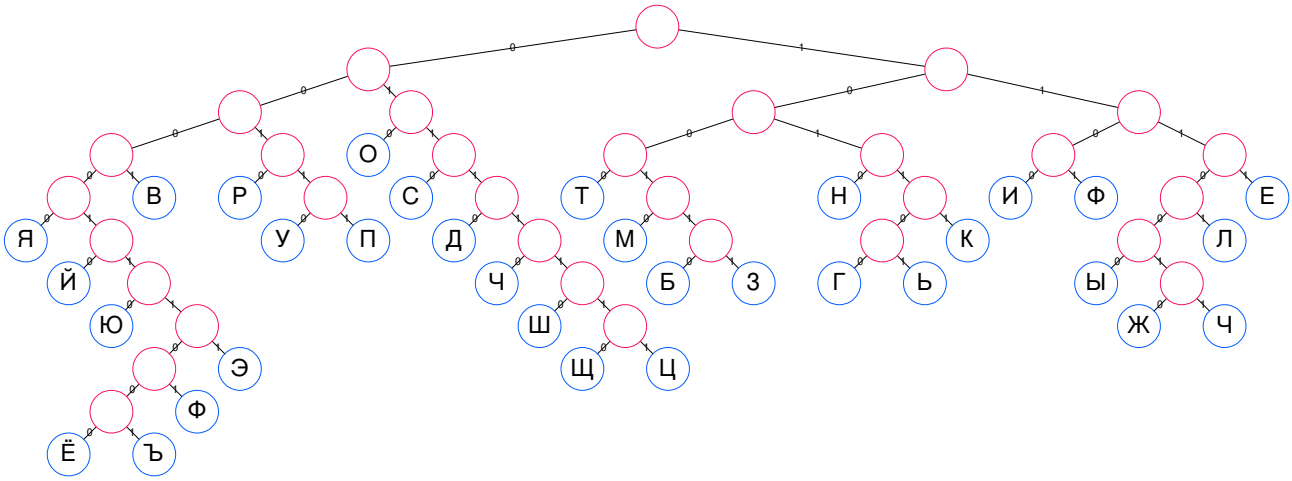

Рисунок 2 - Пример дерева Хаффмана

Заметим, что этот алгоритм хорошо работает для сжатия небольших последовательностей, но если в исходном тексте встречается повторение сочетаний символов, то алгоритм Хаффмана не способен эффективно сжать повторы сочетаний.

1.1.1.2 Адаптивный алгоритм Хаффмана

Для сжатия непрерывного потока данных наиболее оптимальным будет рассматривать не простой алгоритм Хаффмана, а адаптивный [2]. 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.

Основная идея адаптивного кодирования заключается в том, что кодер и декодер начинают работать с незаполненного дерева Хаффмана, а затем модифицируют его по мере поступления и обработки символов. Соответственно, кодер и декодер должны изменять дерево Хаффмана одинаково, чтобы использовать одинаковые кодовые последовательности для каких-либо символов в каждый момент процесса сжатия или распаковки.

Алгоритм кодирования для окна размера N заключается в выполнении следующих действий:

1. Перед кодированием очередного символа обновляются счетчики частот появления в окне всех символов исходного алфавита $A=\{a_1, a_2, \dots, a_n\}$. Обозначим эти частоты как $q(a1), q(a2), \dots, q(an)$. Тогда вероятности символов исходного алфавита оцениваются на основе значений частот символов в окне $P(a_1)=\frac{q(a_1)}{N}; P(a_2)=\frac{q(a_2)}{N}; \dots; P(a_n)=\frac{q(a_n)}{N}$.
2. По полученному распределению вероятностей строится код Хаффмана для алфавита A (размер ограничен N символами).
3. Очередная буква кодируется при помощи построенного кода.
4. Окно сдвигается на один символ вправо, и далее повторяется алгоритм, начиная с пункта 1.

1.1.2 Словарные методы

Метод сжатия с использованием словаря — разбиение данных на слова и замена их на индексы в полученном словаре. В настоящее время является

самым распространенным подходом для сжатия данных и является естественным обобщением RLE (RLE — алгоритм сжатия данных, заменяющий повторяющиеся символы (серии) на один символ и число его повторов).

В наиболее распространенном варианте реализации словарь постепенно пополняется словами из исходного блока данных в процессе сжатия.

Основной параметр любого словарного метода — это размер словаря. Чем больше словарь, тем выше эффективность. Однако для нерегулярных (случайных) данных большой размер словаря может быть неэффективен, т. к. при существенном изменении данных словарь будет заполнен неактуальными словами. Для эффективной работы этих методов при сжатии требуется дополнительная память — приблизительно на порядок больше, чем нужно для исходных данных словаря. Существенное преимущество словарных методов — простая и быстрая процедура распаковки. Дополнительная память при этом не требуется. Такая особенность крайне важна, если необходим оперативный доступ к данным.

1.1.2.1 Алгоритм LZ77

LZ77 — алгоритм сжатия без потерь, опубликованный в статье Абрахама Лемпеля и Якоба Зива в 1977 году. Этот алгоритм один из наиболее известных в семействе LZ*, которое включает в себя также LZW, LZSS, LZMA и другие алгоритмы. LZ77 использует, так называемое, «скользящее окно», что эквивалентно неявному использованию словарного подхода.

Основная идея алгоритма это замена повторного вхождения строки ссылкой на одну из предыдущих позиций вхождения. Для этого используют метод скользящего окна. Скользящее окно можно представить в виде динамической структуры данных, которая организована так, чтобы запоминать «сказанную» ранее информацию и предоставлять к ней доступ. Таким образом, сам процесс сжимающего кодирования согласно LZ77 напоминает написание программы, команды которой позволяют обращаться к элементам скользящего окна, и вместо значений сжимаемой последовательности вставлять ссылки на

эти значения в скользящем окне. В стандартном алгоритме LZ77 совпадения строки кодируются парой:

1. Длина совпадения (match length);
2. Смещение (offset) или дистанция (distance).

Кодируемая пара трактуется именно как команда копирования символов из скользящего окна с определенной позиции, или дословно как: «Вернуться в словаре на значение смещения символов и скопировать значение длины символов, начиная с текущей позиции». Особенность данного алгоритма сжатия заключается в том, что использование кодируемой пары длина-смещение является не только приемлемым, но и эффективным в тех случаях, когда значение длины превышает значение смещения. Пример с командой копирования не совсем очевиден: «Вернуться на 1 символ назад в буфере и скопировать 7 символов, начиная с текущей позиции». Каким образом можно скопировать 7 символов из буфера, когда в настоящий момент в буфере находится только 1 символ? Однако следующая интерпретация кодирующей пары может прояснить ситуацию: каждые 7 последующих символов совпадают (эквивалентны) с 1 символом перед ними. Это означает, что каждый символ можно однозначно определить переместившись назад в буфере, даже если данный символ еще отсутствует в буфере на момент декодирования текущей пары длина-смещение.

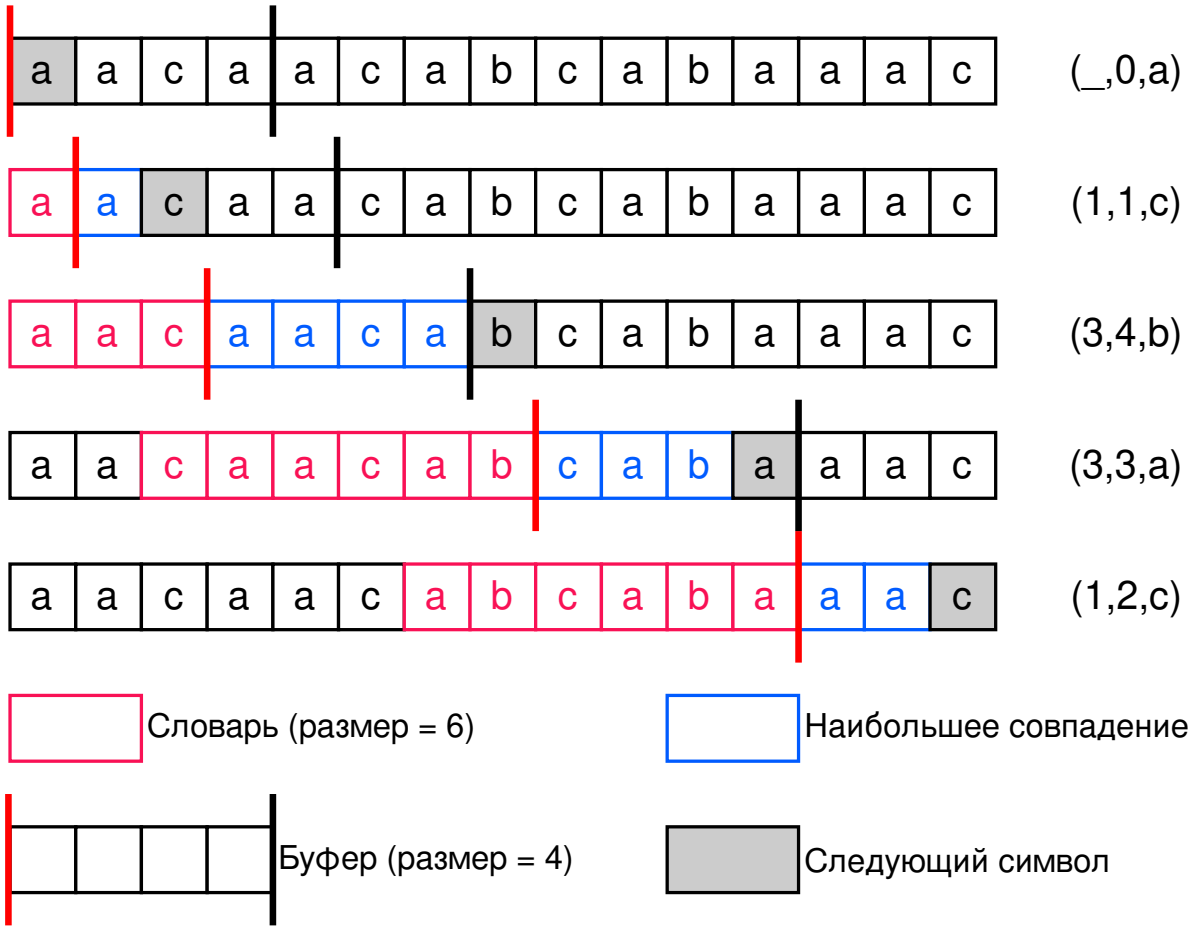

Рисунок 3 - Пример работы алгоритма LZ77

На рисунке 3 представлен пример работы алгоритма LZ77 и использования скользящего окна по входящему сообщению. Допустим, на текущей итерации окно зафиксировано. С правой стороны окна наращиваем подстроку, пока она есть в строке <скользящее окно + наращиваемая строка> и начинается в скользящем окне. Назовем аккумулирующую строку буфером. После наращивания алгоритм выдает код состоящий из трех элементов:

1. Смещение в окне;
2. Длина буфера;
3. Последний символ буфера.

В конце итерации алгоритм сдвигает окно на длину равную длине буфера+1.

1.1.2.2 Алгоритм Deflate

Deflate — это алгоритм сжатия без потерь, использующий комбинацию алгоритмов LZ77 и Хаффмана. Изначально был описан Филом Кацем для второй версии его архиватора PKZIP, который впоследствии был определён в RFC 1951 (1996 год).

Сжатый набор данных состоит из ряда блоков, соответствующих последовательным блокам входных данных. Размеры блоков произвольны, за исключением того, что несжимаемые блоки ограничены 65 535 байтами.

Каждый блок сжимается с помощью комбинации алгоритма LZ77 и кодирования Хаффмана. Деревья Хаффмана для каждого блока независимы от деревьев предыдущих или последующих блоков; алгоритм LZ77 может использовать ссылку на дублированную строку, встречающуюся в предыдущем блоке, до 32K входных байт до этого.

Каждый блок состоит из двух частей: пары кодовых деревьев Хаффмана, описывающих представление сжатой части данных, и сжатой части данных. (Сами деревья Хаффмана сжимаются с помощью кодировки Хаффмана.) Сжатые данные состоят из ряда элементов двух типов: литеральных байтов (строк, которые не были обнаружены как дублированные в предыдущих 32K входных байтах) и указателей на дублированные строки, где указатель представлен в виде пары <длина, обратное расстояние>. Представление, используемое в формате "deflate", ограничивает расстояния до 32K байт и длины до 258 байт, но не ограничивает размер блока, за исключением несжимаемых блоков, которые ограничены, как было отмечено выше.

Каждый тип значений (литералы, расстояния и длины) в сжатых данных представлен с помощью кода Хаффмана, используя одно дерево кода для литералов и длин и отдельное дерево кода для расстояний. Деревья кода для каждого блока отображаются в компактном виде непосредственно перед сжатыми данными для этого блока.

Префиксное кодирование представляет символы из априорно известного алфавита битовыми последовательностями (кодами), по одному коду для каждого символа, таким образом, что различные символы могут быть представлены битовыми последовательностями разной длины, но синтаксический анализатор всегда может однозначно разобрать кодированную строку символ за символом.

Парсер может декодировать следующий символ из закодированного входного потока, идя вниз по дереву от корня, на каждом шаге выбирая ребро, соответствующее следующему входному биту.

Учитывая алфавит с известными частотами символов, алгоритм Хаффмана позволяет построить оптимальный префиксный код (тот, который представляет строки с этими частотами символов, используя наименьшее количество битов любых возможных префиксных кодов для этого алфавита).

В отличие от простого алгоритма Хаффмана, в алгоритме deflate добавлены два дополнительных правила:

1. Все коды заданной битовой длины имеют лексикографически последовательные значения в том же порядке, что и символы, которые они представляют;
2. Более короткие коды лексикографически предшествуют более длинным кодам.

1.1.2.2.а Утилита Gzip

Gzip (сокращение от GNU Zip) — утилита сжатия и восстановления (декомпрессии) файлов, использующая алгоритм Deflate. Применяется в основном в UNIX-системах, в ряде которых является стандартом де-факто для сжатия данных. В соответствии с традициями UNIX-программирования, Gzip выполняет только две функции: сжатие и распаковку одного файла.

Gzip обеспечивает сжатие без потерь, иными словами, исходные данные можно полностью восстановить при распаковке. Он основан на алгоритме

Deflate, который использует комбинацию алгоритма LZ77 и алгоритма Хаффмана.

Gzip не является лучшим метод сжатия, он лишь обеспечивает хороший компромисс между скоростью и степенью сжатия. Сжатие и распаковка у Gzip происходят быстро и степень сжатия на высоком (но не максимальном) уровне.

Из-за огромного количества устройств в мире, которые уже умеют распознавать и обрабатывать данные в сжатом формате Gzip, внедрение каких-либо новых алгоритмов сжатия затруднено.

1.1.2.3 Алгоритм Brotli

Brotli — алгоритм сжатия данных с открытым исходным кодом, разработанный Юрки Алакуйяла и Золтаном Сабадка (разработчики компании Google).

Метод сжатия brotli основан на современном варианте алгоритма LZ77, энтропийном кодировании Хаффмана и моделировании контекста 2-го порядка. Brotli имеет открытый исходный код и позволяет сжимать данные на 20-26% эффективнее, чем его предшественник от Google, алгоритм Zopfli (тоже хлебопекарная продукция из Швейцарии, больше всего похожая внешне на нашу булку-плетенку). Оба алгоритма имеют банальную и простую цель — помочь быстрее загружать веб-страницы.

Разработка Google позволяет сжимать данные без потерь, используя комбинацию алгоритмов LZ77 и кодирование Хаффмана, что ставит Brotli в один ряд с лучшими на данный момент методами общего сжатия данных. В тоже время Brotli работает лучше, чем LZMA и bzip2, а по заверению Google по скорости работы новый алгоритм можно сравнить с Deflate ZLIB.

1.2 Основные принципы сжатия видео

Рассмотрим принципы сжатия видео-данных на примере формата MPEG [3], [4]. В процессе MPEG кодирования устраняются избыточные видеоданные в серии рядом расположенных кадров.

Сжатие видео основано на двух важных принципах. Первый - это пространственная избыточность, присущая каждому кадру видео-ряда. А второй принцип основан на том факте, что большую часть времени каждый кадр поход на своего предшественника. Это называется временная избыточность. Таким образом, типичный метод сжатия видео начинает с кодирования первого кадра с помощью некоторого алгоритма компрессии изображения. Затем происходит кодирование каждого последующего кадра, находя разницу между текущим кадром и его предшественником и кодируя эту разность, производится сжатие видео, при котором используются не все данные каждого видео-кадра, а динамика изменений кадров, т. к. в большинстве последовательных кадров одного видео-сюжета детали на фоне почти не изменяются, а заметные изменения чаще всего происходят на переднем плане.

Например, происходит плавное перемещение небольшого объекта на фоне неизменного заднего плана. В этом случае полная информация о изображении сохраняется только для опорных изображений (базовых кадров). Для остальных кадров оцифровывается только разностная информация: о положении объекта, направлении и величине его смещения, о новых элементах фона, открывающихся за объектом по мере его движения. Причем эта разностная информация вычисляется не только по сравнению с предыдущими изображениями, но и с последующими (поскольку именно в них по мере движения объекта открывается ранее скрытая часть фона) [5].

Процесс сокращения данных производится следующим образом. Прежде всего создается опорный кадр (I, Intra frame). Опорные I-кадры используются для восстановлении остальных кадров и размещаются последовательно через каждые 10-15 кадров. Только некоторые фрагменты кадров, которые находятся между I-кадрами, успевают измениться, и именно эти изменения фиксируются в процессе сжатия [6].

Кроме I-кадров, в MPEG различают еще два типа кадров:

- предсказуемые кадры (P, Predicted), содержащие разность текущего изображения с предыдущим I кадром или с учетом смещений отдельных фрагментов;
- двунаправленные предсказуемые кадры (B, Bidirectionally-predictive), содержащие только отсылки к предыдущим или последующим кадрам типа I или P с учетом смещений отдельных фрагментов.

Опорные кадры составляют основу MPEG потока и через них осуществляется случайный доступ к какому-либо отрывку видео. Сами I-кадры для обеспечения визуально высокого качества сжимаются незначительно.

Предсказуемые кадры кодируются относительно предыдущих кадров (I или P) и используются как сравнительный образец для дальнейшей последовательности P-кадров. В этом случае достигается высокий уровень сжатия.

Двунаправленные предсказуемые кадры кодируются с высокой степенью сжатия. Для привязки B-кадров к видео-последовательности необходимо использовать не только предыдущее, но и последующее изображение. B-кадры никогда не используются для сравнения.

I, P, B кадры объединяются в группы изображений (GOP- Group Of Pictures), представляющие собой минимальный повторяемый набор последовательных кадров, например:

$$(I_0 B_1 B_2 P_3 B_4 B_5 P_6 B_7 B_8 P_9 B_{10} B_{11})(I_{12} B_{13} B_{14} P_{15} B_{16} B_{17} P_{18} \ldots)$$

В стандарте MPEG кадры состоят из макроблоков, представляющих собой небольшие фрагменты изображения размером 16×16 пикселей. Процессор MPEG-энкодера (кодер) анализирует кадры и ищет идентичные или очень близкие макроблоки, сравнивая базовый и последующие кадры. В результате сохраняются только данные о различиях между кадрами, называемые вектором смещения. Макроблоки, которые не содержат изменений, игнорируются, и количество данных для передачи, таким образом, значительно

снижается (сокращается пространственная избыточность путем исключения мелких деталей там где при просмотре видео это визуально не заметно). Для снижения влияния ошибок при передаче данных последовательные макроблоки объединяют в независимые друг от друга разделы. Макроблоки, содержащие незначительное количество изменений отбрасываются — так реализуется сжатие с потерями в стандарте MPEG. Такие блоки могут быть восстановлены из соседних опорных кадров путём интерполяции векторов изменений.ы

Так же каждый макроблок состоит из шести блоков (микроблоки), четыре из которых несут информацию о яркости (Y), а остальные 2 блока несут информацию цветоразностных сигналов (U/V) (сокращается незначащая часть данных цветности). Блоки являются базовыми единицами, над которыми осуществляются основные математические операции кодирования, например, дискретно-косинусное преобразование . Используются схемы блоков Y:U:V 4:2:0 или для студийного (вещательного) качества 4:2:2. Зачастую в видео-потоке незначительная потеря цветов и их градиентов не заметна человеческому глазу, что позволяет слегка снизить качество изображения, но существенно сэкономить на кодировании информации. Интересно, что для корректного восприятия видео-картинки достаточно только Y блоков — сигнал яркости может создавать только монохромное изображение.

1.3 Нахождение наибольшей общей подпоследовательности

Для входных данных x и y (например строк) наибольшей общей последовательностью будет являться последовательность максимальной длины, которая одновременно является подпоследовательностью в слове x и подпоследовательностью в слове y.

МОП (Максимальная общая последовательность) часто используется в программных системах для сравнения файлов для поиска минимального количества элементарных (вставка, удаление, замена) изменений для перевода слова x в слово y.

Наиболее оптимальной и распространённой реализацией является заполнение матрицы $W[(M+1)*(N+1)]$, где M и N размеры слов X и Y для сравнения соответственно. Рассматриваемый алгоритм известен как алгоритм Нидлмана—Вунша и его сложность равна $O(N*M)$.

Заполнение матрицы производится по следующим правилам:

1. Элементы, находящиеся в столбце или строке с индексом 0 принять равными 0.
2. Если элемент в позиции $x[i]$ равен элементу в позиции $y[i]$, то значение ячейки $W[i,j]=W[i-1,j-1]+1$
3. Если элемент в позиции $x[i]$ не равен элементу в позиции $y[i]$, то значение ячейки $W[i,j]=max(W[i,j-1],W[i-1,j])$

Для примера заполним матрицу (см. табл. 1) по для слов «РАЗНОСТНЫЙ» и «АЛГОРИТМ» по данным правилам.

Этот алгоритм хорошо работает на небольших размерах последовательностей, но если нам необходимо сравнить 2 файла, объёмом 10000 строк каждый, то потребуется 10^8 ячеек памяти и такое же количество итераций для вычисления их значений.

Заметим, что для заполнения матрицы W на каждой итерации по элементам слова x нам нужна только строка, полученная на предыдущем шаге, а если итерироваться по слову y, то необходимо запоминать только значение предыдущего столбца. А если в задаче необходимо найти только длину МОП (без вычисления самой МОП), то можно снизить использование памяти до O(min(N, M)), сохраняя только две строки или два столбца матрицы W.

Таблица 1- Пример поиска наибольшей общей последовательности для слов «РАЗНОСТНЫЙ», «АЛГОРИТМ». МОП= «АОТ»

		Р	**А**	**З**	**Н**	**О**	**С**	**Т**	**Н**	**Ы**	**Й**
	0	0	0	0	0	0	0	0	0	0	0
А	0	0	1	1	1	1	1	1	1	1	1
Л	0	0	1	1	1	1	1	1	1	1	1
Г	0	0	1	1	1	1	1	1	1	1	1
О	0	0	1	1	1	2	2	2	2	2	2
Р	0	1	1	1	1	2	2	2	2	2	2
И	0	1	1	1	1	2	2	2	2	2	2
Т	0	1	1	1	1	2	2	3	3	3	3
М	0	1	1	1	1	2	2	3	3	3	3

Применяя указанные выше оптимизации и оптимизации, специфичные для применения в конкретной предметной области можно добиться исполнения алгоритма за линейное время и линейное использование памяти. Подробнее об указанном алгоритме поиска наибольшей общей последовательности и его производных можно прочитать в литературе по динамическому программированию [7].

2 Разработка подсистемы сжатия

2.1 Проектирование

2.1.1 Анализ технического задания, выбор подхода и средств разработки

Согласно техническому заданию в рамках данной работы необходимо реализовать подсистему сжатия данных (библиотеку, подключаемый модуль). Зависимости от операционной системы компьютера (дистрибутива на основе ядра Linux или же операционной система семейства Windows) нет, т. к. данная подсистема распространяется в виде исходных кодов. В ходе кодирования и декодирования информации необходимо обеспечить контроль целостности данных.

В техническом задании указано, что для реализации подсистемы сжатия данных в качестве средства разработки необходимо использовать компилируемый многопоточный язык программирования Golang [8]. Данный язык программирования накладывает некоторые ограничения на разработку программного обеспечения, одно из которых — отсутствие ООП и, соответственно, инкапсуляции, полиморфизма и наследования. Следовательно, нашим подходом к разработке программного обеспечения будет являться структурный подход. В его основе лежит декомпозиция системы декомпозиция предметной области по данным — структурам и реализация основного функционала программного продукта через методы, производящие обработку и преобразования этих структур.

В качестве интерфейсной части необходимо реализовать программный интерфейс для интеграции подсистемы в другие системы или модули и обеспечить максимальную совместимость с интерфейсами библиотеки «gzip» для обеспечения лёгкости интеграции в уже существующие системы.

За модель жизненного цикла программного обеспечения возьмём модель с промежуточным контролем. Данная модель позволит возвращаться на этапе реализации к этапам анализа и проектирования ПО для проработки незамеченных на данных этапах ранее проблем и ускорить разработку

подсистемы сжатия данных за счет отказа от деления разработки на итерации и создания прототипов.

Используем так же предметно-ориентированное программирования. Данный набор принципов разработки направлен на создание оптимальных систем (наборов) объектов и сводится к созданию программных абстракций, в которые входит бизнес-логика, устанавливающая связь между реальными условиями применения продукта и кодом. Этот подход полезен при разработке программных систем (и подсистем) в случае, если разработчик не является специалистом в предметной области, но может спроектировать разрабатываемый продукт, основываясь на ключевых моментах и знания предметной области. Это обеспечивает простоту модификации программного продукта в случае незначительных противоречий готового ПО и предметной области или изменения структур данных.

В виду небольшого объема программного обеспечения (малое количество модулей, входящих в подсистему) при проектировании и реализации программного продукта воспользуемся методом восходящей разработки, что позволит в первую очередь сконцентрироваться на структурах данных [9] низкоуровневых операциях над ними, а лишь после их полной реализации перейти к проектированию интерфейсной части программы.

2.1.2 Разработка структурной схемы

Структуру разрабатываемой подсистемы можно разделить на несколько модулей на базе основных функций: модуль сжатия данных и модуль распаковки данных для сжатия и разархивирования данных соответственно (см. рис. 4).

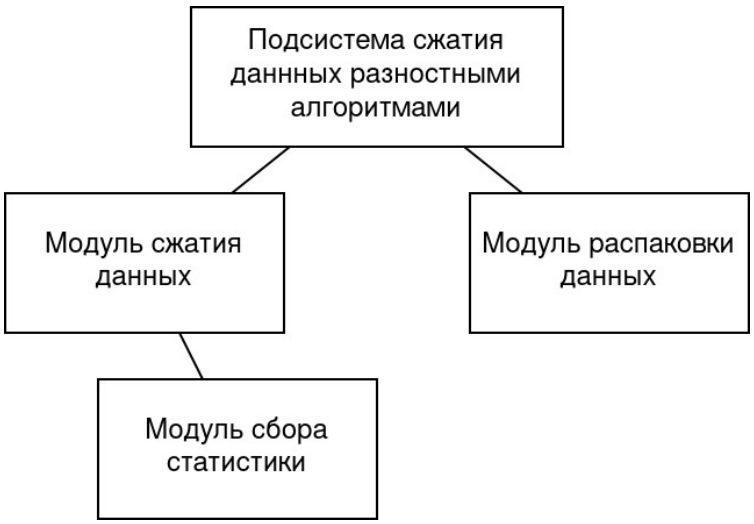

Рисунок 4 - Структурная схема

В модуль сжатия данных добавлен модуль сбора статистики. Данный модуль отвечает за определение похожести двух кадров (модифицированным алгоритмом поиска наибольшей общей последовательности) и на его основе модуль сжатия данных будет принимать решение — текущий пакет данных будет передан как базовый или составной кадр. На основе статистики по кадрам может быть принято решение по увеличению или уменьшению размера кадров. Модули сжатия и распаковки данных включают в себя подсчет контрольных сумм для сжатых и распакованных данных соответственно. Будет использоваться сцепление модулей (мера взаимодействия модулей) по образцу (модули обмениваются данными, находящимися в структурах [10]), что обеспечит среднюю устойчивость к ошибкам других модулей и хорошую наглядность.

2.1.3 Разработка интерфейсов подсистемы

Для наиболее простой интеграции подсистемы в программные продукты возьмём за основу интерфейс пакета «gzip» для языка программирования Golang. Данный пакет предоставляет 11 стандартных функций: 5 функций для интерфейса Читатель (io.Reader) и 6 функций для интерфейса Писатель (io.Writer) (подробно прочитать о структуре интерфейсов в Golang можно прочитать в статье [6]).

В общем случае файл Gzip может быть объединением файлов нескольких Gzip, каждый из которых имеет свой собственный заголовок. Чтение из интерфейса Читатель возвращает конкатенацию несжатых данных каждого из них.

Сжатые файлы хранят длину и контрольную сумму несжатых данных. Читатель вернет контрольную сумму и ошибку, если будет достигнут конц данных, но не совпадет полученная и ожидаемая длина данных или контрольная сумма (CRC-32).

Читатель - это интерфейс io.Reader, который может быть прочитан для извлечения несжатых данных из сжатого файла.

func NewReader(r io.Reader) (*Reader, error)

NewReader (конструктор) создает новый Читатель, на основе другого интерфейса Читатель. Если r также не реализует io.ByteReader (т. е. является модифицирующим интерфейсом), декомпрессор может считывать из r больше данных, чем это необходимо. Разработчику необходимо позаботиться о закрытии интерфейса, после того как данные вычитаны.

func (z *Reader) Close() error

Закрывает Читатель, но не закрывает Читатель, от которого был инициализирован. Если реализуется механизм контрольных сумм, то должны быть полностью вычитаны все данные до приходя сигнала EOF.

func (z *Reader) Multistream(ok bool)

Флаг многопоточности определяет, поддерживает ли Читатель многопоточные файлы.

Если этот параметр включен, то Читатель ожидает, что входные данные будут представлять собой последовательность отдельных потоков данных, каждый из которых имеет свой собственный заголовок и поток данных, заканчивающийся на EOF.

Если этот параметр отключен, то когда считыватель достигает конца потока данных, метод Read возвращает сигнал EOF. Чтобы запустить следующий поток, необходимо вызвать Reset(r), а затем z.Multistream(false). Если следующего потока нет, Reset (r) вернет сигнал EOF.

func (z *Reader) Read(p []byte) (n int, err error)

Метод для чтения из интерфейса в массив байт. Возвращает количество прочитанных байт и ошибку, если она произошла.

func (z *Reader) Reset(r io.Reader) error

Сбрасывает состояние Читателя z и делает его эквивалентным результату его исходного состояния из NewReader, но вместо этого считывает из r. Это позволяет повторно использовать читатель, а не выделять новый.

Писатель- это интерфейс io.WriteCloser, который пишет сжатые данные.

func NewWriter(w io.Writer) *Writer

NewWriter возвращает новый экземпляр типа Писатель. Запись сжатых данных может быть буферизована и данные не будут сбрасываются до закрытия интерфейса.

func NewWriterLevel(w io.Writer, level int) (*Writer, error)

NewWriterLevel похож на NewWriter, но позволяет указать степень сжатия.

Степень сжатия может быть сжатием по умолчанию, без сжатия или любым целочисленным значением между BestSpeed (0) и BestCompression (9) включительно.

func (z *Writer) Close() error

Закрывает интерфейс Писатель и удаляет любые несохраненные данные базового ввода-вывода, но не закрывает базовый io.Writer.

func (z *Writer) Flush() error

Flush сбрасывает все отложенные сжатые данные в базовый блок записи.

Он полезен главным образом в сжатых сетевых протоколах, чтобы гарантировать, что удаленный считыватель имеет достаточно данных для восстановления пакета.

func (z *Writer) Reset(w io.Writer)

Reset отбрасывает состояние Писателя z и делает его эквивалентным результату его исходного состояния из NewWriter или NewWriterLevel, но вместо этого пишет в w. Это позволяет повторно использовать интерфейс, а не выделять новый.

func (z *Writer) Write(p []byte) (int, error)

Write записывает сжатую форму p в базовый интерфейс Писателя z. Запись сжатых данных может быть буферизована и данные не будут сбрасываются до закрытия интерфейса.

2.1.4 Разработка вариантов использования подсистемы

Предполагается, что подсистема может быть использована в двух режимах работы:

1. Режим поточного сжатия данных;
2. Режим сжатия данных со случайным доступом.

Под режимом поточного сжатия данных предполагаем, что в конечную систему разработчика поступает непрерывный поток данных, имеющих некоторую регулярность (например, показания датчиков температуры или положения в пространстве). Пусть в данном случае имеется необходимость быстрого доступа к случайному объекту на ленте. Типичные словарные методы для получения данных в некоторый момент времени требуют полного вычитывания истории (всех предыдущих изменений данных), что может занять значительное время. Наш алгоритм потребует исполнения обратного просмотра (просмотра потока\ленты в сторону старых событий) для поиска данных, на которых строятся текущие данные. Хотя и найдётся такой частный случай, что придется произвести полное вычитывание исторических данных, но его можно избежать путём принудительно проставления базовых (неизменённых) сообщений не реже чем 1 в N информационных сообщений.

Под режимом сжатия данных со случайным доступом предполагаем, что доступ к любому файлу во времени необходимо произвести максимально быстро и файлы хранятся в некотором распределённом хранилище. Данный метод так же подразумевает, что файлы являются историческими и не могут быть изменены в прошлом (хотя система архитектура не запрещает изменять составные файлы, но для защиты от случайного изменения базового файла следует запретить любые исторические изменения). Такой подход полезен, если есть необходимость по расписанию делать копию файла (файлов) [1]. В данном случае базовый файл находятся в начале каждого календарного сезона, месяца или недели, а все остальные файлы считаются составными от него.

Рассмотрим подробнее пример построения системы на основе распределённого хранилища [11].

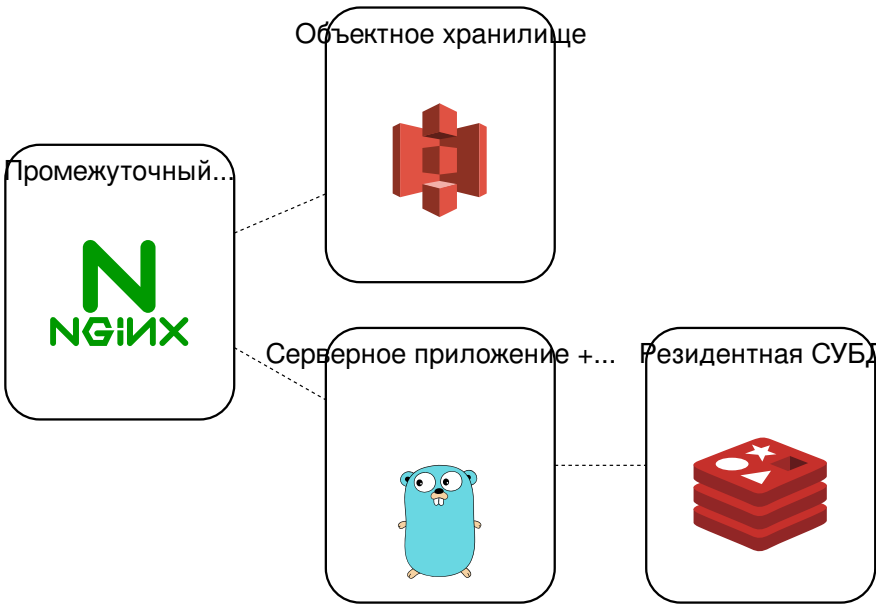

Рисунок 5 - Пример построения системы с использованием подсистемы сжатия данных

Наиболее популярным на сегодняшний день является реализация с использованием промежуточного (прокси) сервера, например NGINX. Данный сервер позволяет не только направлять запросы на нужные подсистемы, но также и обрабатывать ошибки.

В качестве СХД возьмем любое S3-совместимое облачное хранилище, например Amazon S3.

Как резидентную СУБД (системы хранения вида «ключ-значение», находящейся в оперативной памяти) [12] используем Redis 5. Это популярная база данных, используемая для хранения горячих данных (например cookie пользователя).

Серверное приложение рекомендуется объединить с подсистемой сжатия данных, что снизит накладные расходы на передачу данных между сервисами и благоприятно скажется на общей скорости работы системы.

2.2 Реализация

Задача сводится к созданию простого разностностного алгоритма для сжатия и восстановления данных. Алгоритмически реализуем подсистему сжатия данных на примере стендарта MPEG [13].

В стандарте MPEG фигурируют опорные, предсказуемые и двунаправленные кадры. В подсистеме сжатия данных введём понятия базовых (опорных) и составных (предсказуемых) кадров с сегментами данных. Решение с двунаправленными предсказуемыми кадрами применять не будем, т. к. это повлечёт за собой кратное увеличение нагрузки на кодер и увеличит сложность разрабатываемого программного обеспечения.

Базовый кадр будет содержать информацию кадра в неизменном виде, составной кадр будет формироваться на основе данных, полученных с использованием модифицированного алгоритма поиска максимальной общей последовательности. В данном алгоритмы мы будем не выделять МОП, а выделять интервалы с различающимися данными, стараясь привести суммарную длину этих интервалов к минимуму.

Заметим, что для обеспечения контроля правильности кодирования и декодирования составных кадров необходимо передавать вместе с каждым составным кадром контрольную сумму. Наиболее популярным алгоритмом для быстрого подсчета контрольных сумм, поддерживающим работу в поточном режиме, является алгоритм CRC32.

Подсистема сжатия данных состоит из двух частей — интерфейсная и внутренняя. Интерфейсная часть обеспечивает связь других программных модулей с данной подсистемой, внутренняя — выполнение основных функций подсистемы.

2.2.1 Интерфейс программной подсистемы

Обязательными к реализации интерфейсные функции:

Для распаковки данных (Reader):

1. NewReader() - конструктор читателя;
2. Close() - деструктор писателя;
3. Read() - считывает набор данных в массив байт;
4. Reset() - отмена распаковки данных. Данные из читателя передаются в сыром виде (виде «как есть»).

Ниже (см. рис. 6) представлена диаграмма состояний [14] читателя и сигналы для переходов из одного состояния в другое.

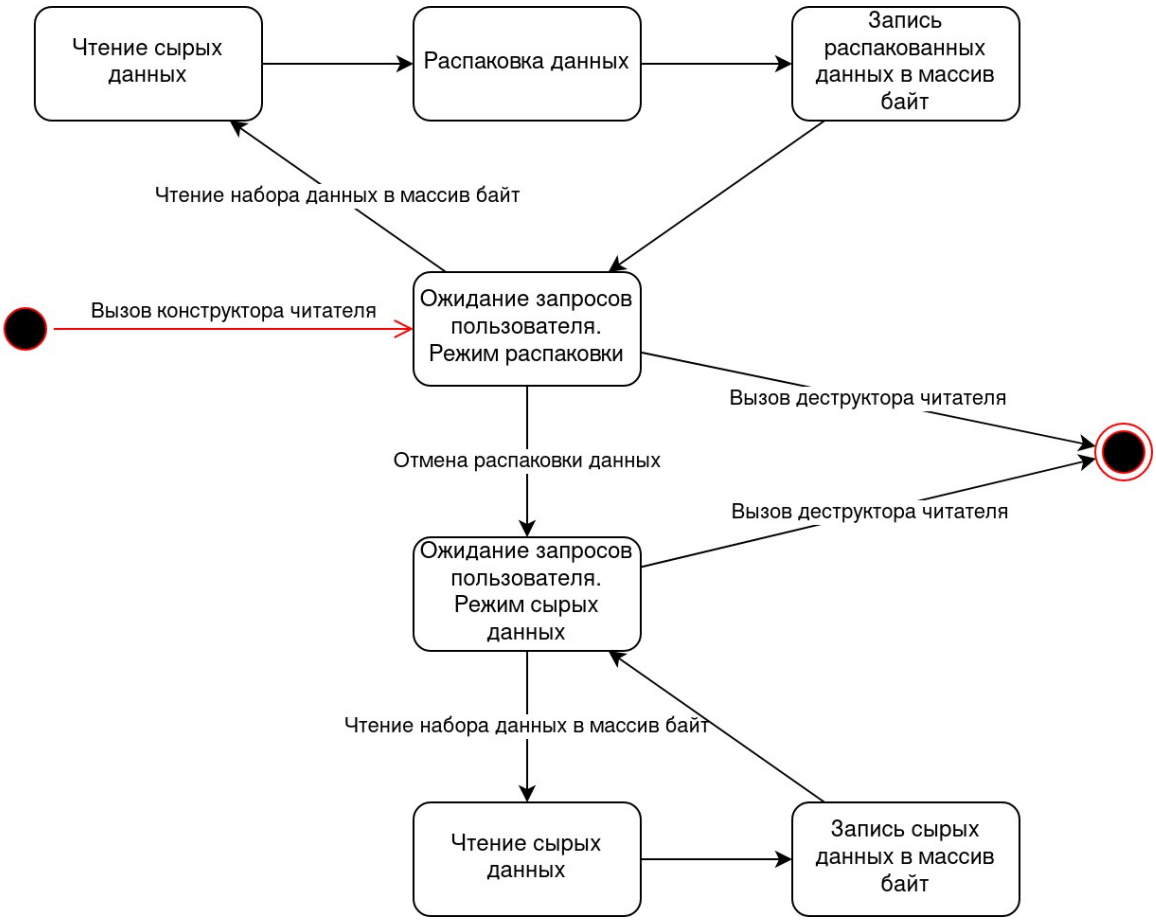

Рисунок 6 - Диаграмма состояний читателя

Для сжатия данных (Writer):

1. NewWriter() - конструктор писателя;
2. Close() - деструктор писателя;
3. Flush() - сброс статистики и начало записи данных с нового базового кадра;
4. Reset() - отмена сжатия данных. Данные из писателя передаются в сыром виде (виде «как есть»);
5. Write() - записывает блок данных в писателя.

Ниже (см. рис. 7) представлена диаграмма состояний писателя и сигналы для переходов из одного состояния в другое.

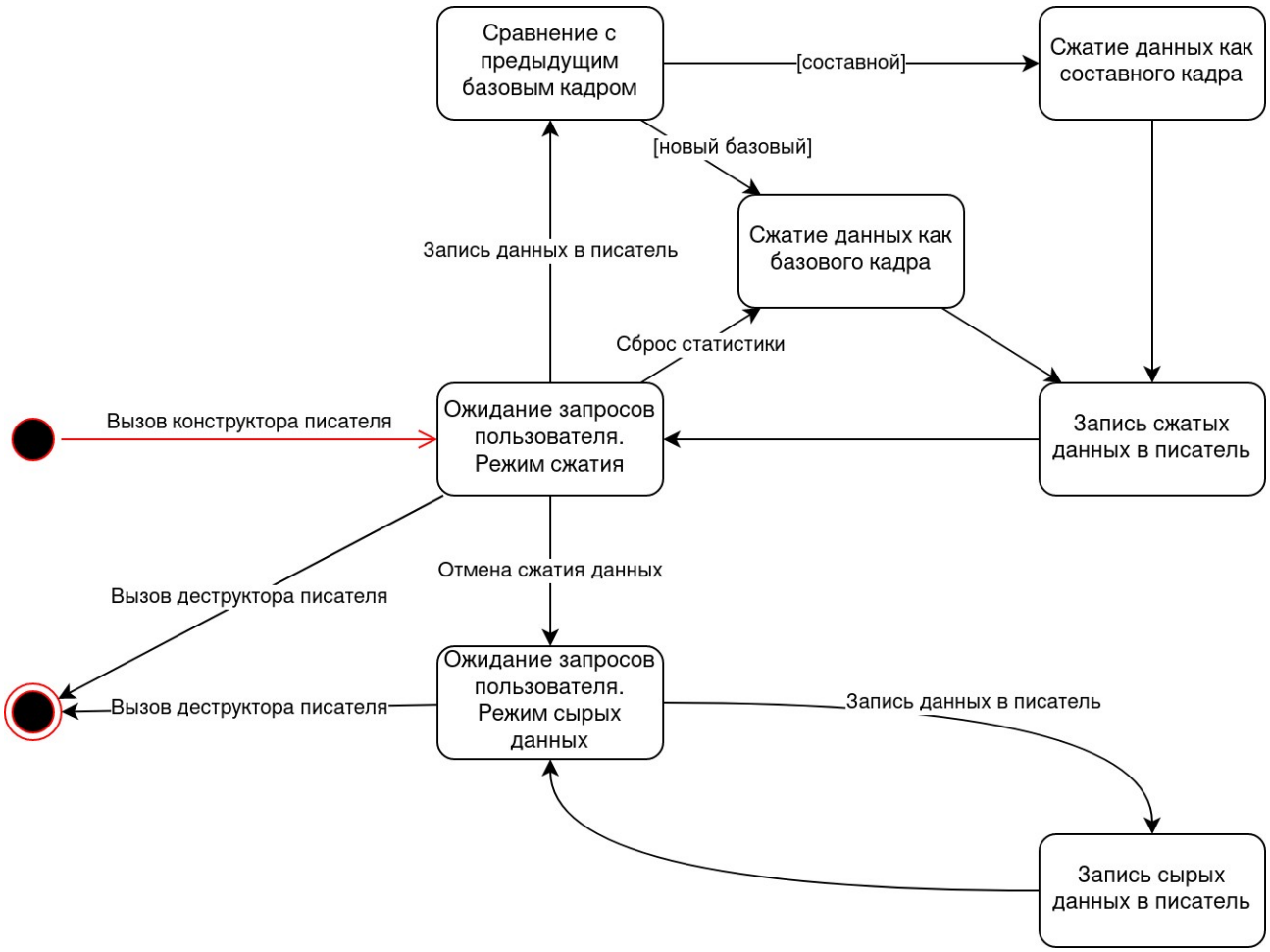

Рисунок 7 - Диаграмма состояний писателя

2.2.2 Реализация внутренних функций

Обязательными к реализации внутренние функции:

1. Метод вычисления разницы между двумя блоками байт и выделения фрагментов данных с их смещениями. В случае, когда базовый и составной кадры разной длины необходимо предусмотреть 3 типа сегментов (для добавление, изменения или удаления).
2. Метод построения кадра на основе базового кадра и фрагментов данных и смещений составного кадра.
3. Метод вычисления оптимального размера блока для текущего потока байт (метод эвристической подстройки размера кадра).
4. Сжатые данные — набор кадров (базовых и составных).

Проанализировав выполняемые функции и требования к системе опишем формат кадра. Кадр содержит следующую информацию:

1. Свой тип (базовый или составной);
2. Размере кадра;
3. Смещения сегментов данных;
4. Сегменты с данными;
5. Типах сегментов (добавленный, удалённый или изменённый);
6. Контрольную сумму (опционально).

В формате proto кадру будет иметь следующий вид:

```
message Frame {
  bool isBaseFrame = 1;
  uint64 frameSize = 2;
  repeated uint64 fragmentShift = 3;
  repeated bytes fragmentData = 4;
  repeated uint8 fragmentType = 5;
  string checksum = 6;
}
```

2.2.2.1 Сбор статистики

Методы по сбору статистики позволяют принять решение об изменении типа используемого кадра или размеров кадра. Тип используемого кадра может быть изменён на основе отношения размера получаемых сегментов составного кадра и размера базового кадра, а так же в зависимости от переменных, влияющих на сбор статистики.

В модуле сбора статистики определены следующие переменные, которые можно переопределить в момент интеграции подсистемы в другое программное обеспечение:

1. **EndMarker** — Маркер кадра — символ или определённая их последовательность, позволяющая определить во входном потоке различные кадры для сжатия (например «<\html>» для веб-страниц). По умолчанию используется пустая строка, т. е. Весь входной поток будет считаться единым кадром.
2. **FixSize** — Фиксированный размер кадров — количество маркеров, позволяющих определить момент, с которого начнется следующий кадр. Данная переменная полезна при использовании непустых маркеров кадра (например «\n») и позволяет реализовать блочное сжатия по N строк.
3. **ForceBaseEvery** — Принудительное использование базового кадра — флаг, устанавливающий принудительное использование базового кадра реже чем раз в N кадров. Данный флаг полезен при использовании случайного доступа к потоку данных с возможностью обратного просмотра и позволяет за приемлемое время восстановить составной кадр, прочитав не более чем N-1 предыдущих кадров для поиска базового.
4. **Ratio** — отношение разницы суммарного размера сегментов и размера данных базового кадра к размеру данных базового кадра. Может изменяться в интервале от 0.1 до 1. По умолчанию 0.5.

2.2.2.2 Сжатие данных

Реализация сжатия данных требует реализованные методы по сбору статистики по кадрам данных.

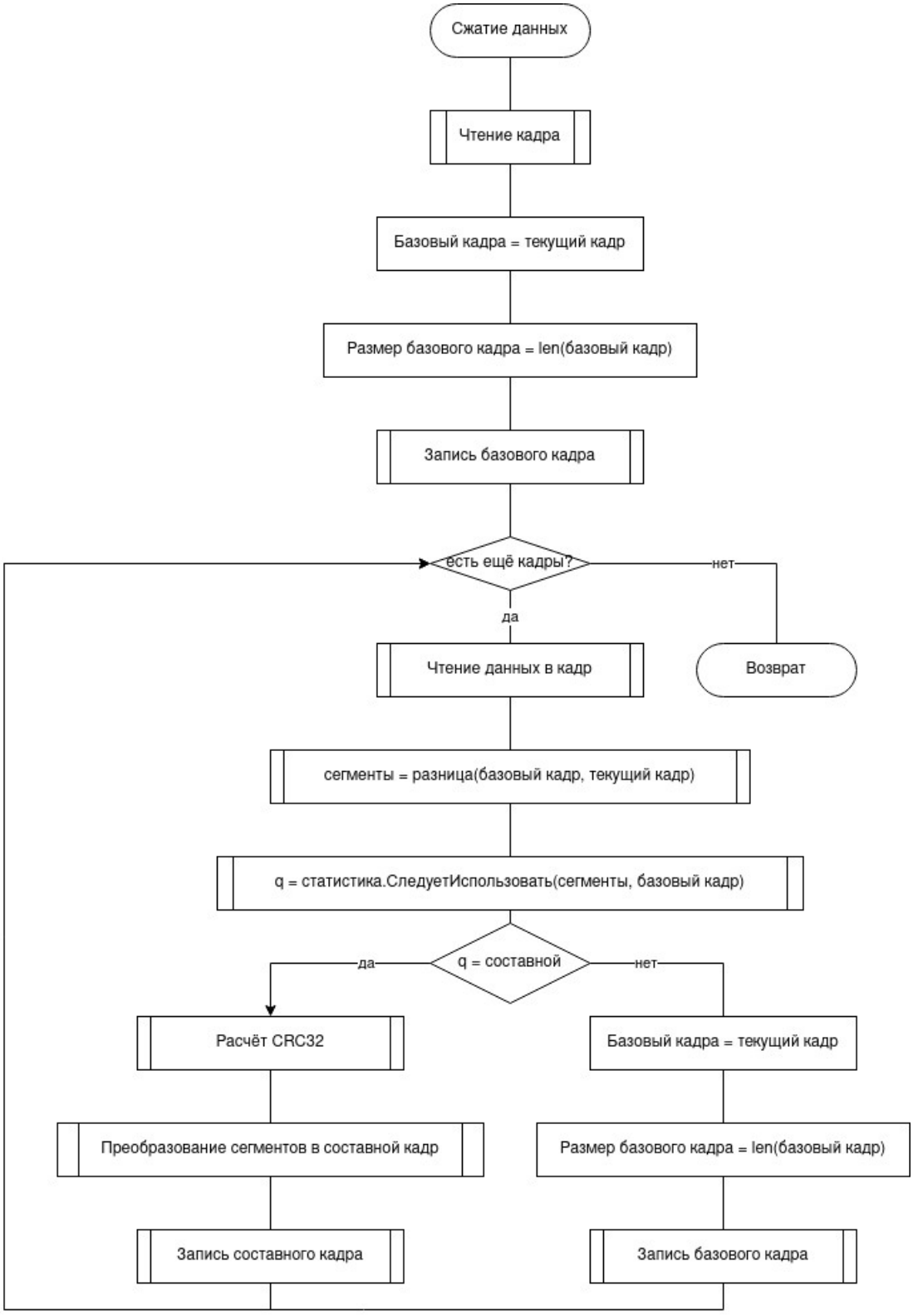

Рисунок 8 - Схема алгоритма сжатия данных

Схема сжатия данных (см. рис. 8) не описывает алгоритмы работы с интерфейсом Писатель (Writer), а лишь показывает алгоритм, заложенный в реализацию подсистему сжатия данных с использованием разностных алгоритмов. Подробнее о составлении схем алгоритмов можно прочитать в дополнительной литературе [15].

Т.к. сжатие данных решено проектировать и реализовывать, вводя понятия базовых и составных кадров, то для начала необходимо произвести формирование базового кадра, на основе переданных данных. Небольшой оптимизацией будет расчет размера этого базового кадра для дальнейшего сравнения с полученными размерами составных кадров. В случае если данные успешно вычитаны в кадр и при этом не возникло ошибок на уровне интерфейсов, то можно переходить к этапу формирования последующих кадров.

В случае если есть ещё кадры (интерфейс читателя не закрыт и не был сброшен в прошлом) будем вычитывать данные в новый кадр. На нового кадра и базового кадра, который хранится в оперативной памяти, получим сегменты с дынными для вставки, удаления или модификации данных на основе базового кадра. Рассчитаем суммарный объем полученных сегментов с данными относительно ранее просчитанного размера базового кадра.

Если это рекомендуется использовать базовый кадр, т. к. сделано предположение, что последующие кадры будут похожи на наш текущий и суммарный объем будущих сегментов на базе этого кадра будет меньше, если мы оставим старый базовый кадр, то перезапишем в ОЗУ базовый кадр текущим и передадим в интерфейс сериализованный в бинарный вид базовый кадр. Иначе, следуют использовать составной кадр. Рассчитаем контрольную сумму для данных текущего кадра, сформируем составной кадр на основе сегментов изменений и передадим его в интерфейс в сериализованном в бинарное представление виде.

2.2.2.3 Распаковка данных

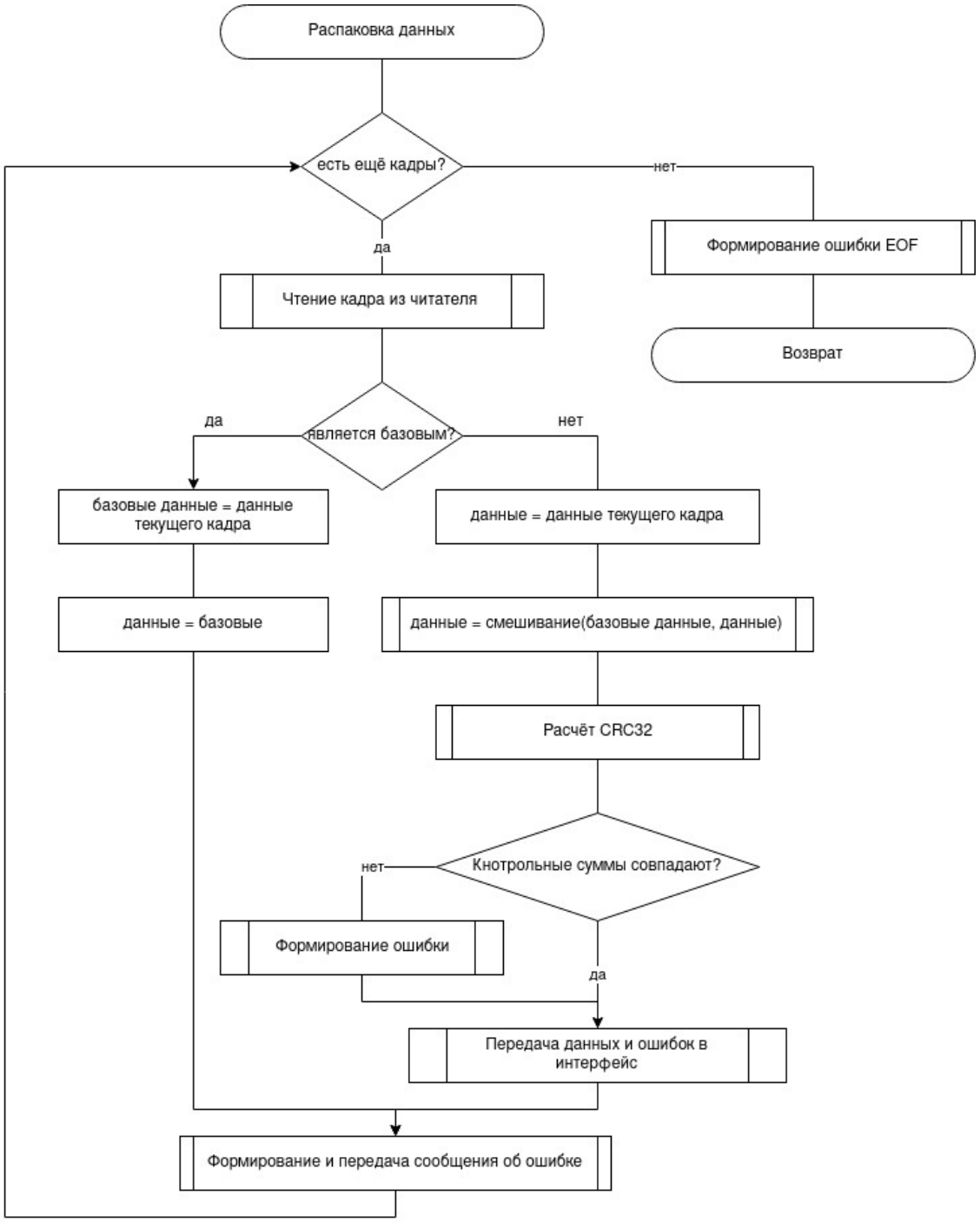

Рисунок 9 - Схема алгоритма распаковки данных

Распаковка данных, сжатых с использованием разностных алгоритмов, требует затрат оперативной памяти в 2 раза больше, чем максимальный размер получаемого кадра.

Алгоритм распаковки данных требует вычитывание кадров из интерфейса, их десериализзацию из бинарного вида в структуру и дальнейшая

обработка. За методом «Чтение кадра из читателя» находится взаимодействие с интерфейсом и преобразование бинарных данных в структуру кадра.

Если кадр является базовым, то необходимо перезаписать переменную «базовый кадр» в ОЗУ и передать данные в переменную на чтения, не изменяя их. Если кадр является составным, то необходимо смешать текущие данные с данными базового кадра, рассчитать контрольную сумму CRC32 для них и сверить её с тем, что передано в поле checksum. Если контрольные суммы совпали, то передача и преобразование данных произведены без ошибок и эти данные можно передать в переменную чтение, иначе выдать ошибку (file corrupted) для данного кадра.

2.2.3 Компоновка подсистемы

В результате предыдущих 3 пунктов были реализованы 3 модуля, которые необходимо объединить в одну подсистему (библиотеку). Необходимо так же обеспечить функционирование подсистемы в качестве автономного программного продукта [11] — реализовать возможность получения основного функционала подсистемы сжатия данных удалённо. Для этого, используя стандартный функционал выбранного языка программирования, добавил обработчики на HTTP и GRPC интерфейсы. Данные обработчики в качестве входных данных принимают http и grpc запросы соответственно и преобразуют данные так, что становится возможно обработать их с использованием реализованной библиотеки. Ответ (сжатые или распакованные данные) так же сериализуются для отправки клиенту через один из обработчиков.

При группировке данных модулей следует полагаться на следующие принципы:

1. Корневой пакет для доменных типов (в нашем случае Reader и Writer);
2. Группировка пакетов по зависимостям.

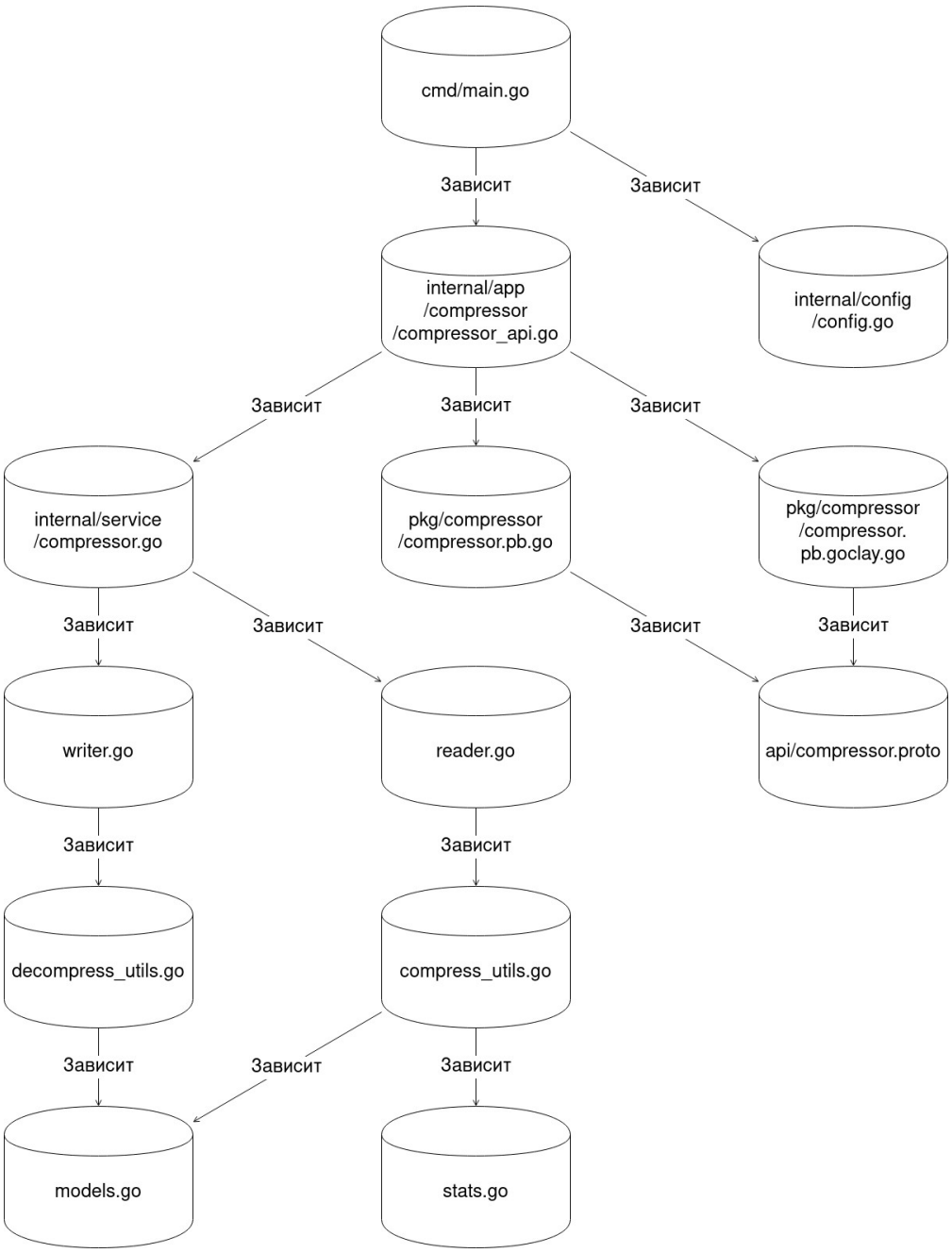

Рисунок 10 - Диаграмма компоновки подсистемы сжатия данных с использованием разностных алгоритмов

Сгруппировав компоненты подсистемы по описанным выше принципам получим готовый к распространению в виде библиотеки и в виде автономной системы сжатия данных программный продукт. Интерфейс прикладного программирования (API) описан в спецификации «compressor.proto».

3 Разработка технологии тестирования подсистемы

На каждом этапе жизненного цикла программного обеспечения необходимо выполнять валидацию (проверка на соответствие требованиям к программному продукту) и верификацию (проверка на соответствие условиям, сформированным в начале разработки) реализуемого продукта. Тестирование представляет собой важный этап процесса разработки программного обеспечения. Оно преследует несколько целей: повысить вероятность правильной работы при любых обстоятельствах, повысить вероятность соответствия всем описанным требованиям, предоставить информацию о состоянии (готовности) программного продукта на конкретном этапе разработки.

В данной работе необходимо разработать технологию и провести тестирование подсистемы сжатия данных. Для тестирования разрабатываемого продукта реализуем следующие виды тестирования: модульное тестирование (отдельных модулей подсистемы), функциональное тестирование (основных функций подсистемы) и оценочное тестирование.

3.1 Модульное тестирование

Модульное тестирование — тестирование программного продукта на уровне отдельных взятых модулей, методов. Целью данного вида тестирования является выявление локализованных ошибок в реализации алгоритмов и определение готовности к интеграции отдельных модулей в подсистему. Модульное тестирование считается низкоуровневым (близким к исходному коду модуля) и проходит по методу «белого ящика», т. е. для написания тестов используется знания о внутренней реализации (алгоритмах) модулей.

3.1.1 Тестирование модуля сбора статистики

Для тестирования данного модуля рассмотрим в программе тестирования несколько наборов данных:

1) Данные для сравнения являются полной копией друг друга;
2) Данные для сравнения являются одинаковыми по размеру и схожими по содержанию;
3) Данные для сравнения являются одинаковыми по размеру и различными по содержанию;
4) Данные для сравнения являются различными по размеру и схожими по содержанию;
5) Данные для сравнения являются различными по размеру и различными по содержанию.

Проведём тестирование на описанных выше наборах данных и занесём ожидаемую и полученную реакцию подсистему в таблицу (см. табл. 2).

Таблица 2 - Результаты тестирования модуля сбора статистики

Тест	**Размеры**	**Содержания**	**Ожидалось**	**Получено**
1	Одинаковые	Одинаковые	Использовать составной кадр	Использовать составной кадр
2	Одинаковые	Различные	Новый базовый кадр	Новый базовый кадр
3	Одинаковые	Схожие	Использовать составной кадр	Использовать составной кадр
4	Различные	Схожие	Использовать составной кадр	Использовать составной кадр
5	Различные	Различные	Новый базовый кадр	Новый базовый кадр

По результатам тестирования модуля сбора статистики (см. табл. 2) можно говорить о корректной работе методов по указанию использования базовых и составных кадров для модуля сжатия данных.

3.1.2 Тестирование модуля сжатия данных

Для тестирования данного модуля в программе тестирования необходимо рассмотреть кадры, которые предположительно должны хорошо сжиматься разностными алгоритмами, совместно с выставленными флагами для сжатия («использовать базовый кадр», «использовать составной кадр»):

1. Кадры, схожие по размеру и содержанию с флагом «использовать составной кадр»
2. Кадры, различные по размеру и содержанию с флагом «использовать базовый кадр»
3. В колонках «Ожидалось» и «Получено» таблицы тестирования модуля сжатия (см. табл. 3) укажем коэффициент сжатия.

Таблица 3 - Результаты тестирования модуля сжатия

Содержимое и размер кадров	**Флаг**	**Ожидалось**	**Получено**
Схожие	использовать составной кадр	2	1.9
Различные	использовать базовый кадр	1	0.9

По результатам тестов коэффициент сжатия оказался немного меньше ожидаемого из-за наличия служебной информации о размерах и типах кадров в результирующем файле. На данных с большими размерами кадров эффективность использования сжатого пространства возрастает из-за уменьшения относительного размера служебной информации [12]. Заметим, что тестирование сжатия производилось на заведомо хороших для сжатия разностными алгоритмами образцах данных, сжатие бинарных или случайных (близких к белому шуму) данных может показывать результаты хуже представленных выше.

3.1.3 Тестирование модуля распаковки данных

Для тестирования данного модуля в программе необходимо синтезировать образцы сжатых кадров следующего формата:

```
message Frame {
  bool isBaseFrame = 1;
  uint64 frameSize = 2;
  repeated uint64 fragmentShift = 3;
  repeated bytes fragmentData = 4;
  repeated uint8 fragmentType = 5;
  string checksum = 6;
}
```

В качестве данных для базового кадра закодируем сообщение «Подсистема сжатия данных словарными методами», а для составного - «Подсистема сжатия данных с использованием разностных алгоритмов» (см. табл. 4).

Таблица 4 - Результаты тестирования модуля распаковки

Поле	Базовый кадр	Составной кадр	Получено	Ожидалось
isBaseFrame	True	False	Базовый: «Подсистема сжатия данных словарными методами» Составной: «Подсистема сжатия данных с использованием разностных алгоритмов»	Базовый: «Подсистема сжатия данных словарными методами» Составной: «Подсистема сжатия данных с использованием разностных алгоритмов»
frameSize	44	63		
fragmentShift	{0}	{26}		
fragmentData	[«Подсистема сжатия данных словарными методами»]	[« использованием разностных алгоритмов»]		
fragmentType	{0}	{3}		

Синтезировав 2 сжатых кадра (базовый и составной) и передав их в метод разархивирования кадров, мы получили несжатые кадры, аналогичные расчетным. Заметим, что во время тестирования модуля распаковки данных было проигнорировано поле контрольной суммы, т. к. подсчет контрольной суммы производится стандартной библиотекой "hash/crc32" и для одинаковых (в нашем случае текстовых) данных ожидается одинаковые результаты подсчета CRC32.

3.2 Функциональное тестирование

На данном этапе программа тестирования должна использовать основные функции подсистемы сжатия данных: сжатие и распаковка данных. Для тестирования можно взять какой-либо файл, сжать его, вывести информационное сообщение о характеристиках сжатого файла, распаковать сжатый файл и сравнить исходный и распакованный файлы.

Приложение для проведения тестирования подсистемы сжатия данных должно выполнять следующие функции:

1) Ввод данных о размещении файла на диске;
2) Компрессия исходного файла, используя подсистему сжатия данных;
3) Вывод информации о размерах исходного и сжатого файлов;
4) Декомпрессия сжатого файла, используя подсистему сжатия данных;
5) Вывод информации о размерах сжатого и распакованного файлов;
6) Проверка соответствия исходного файла файлу, полученному в результате последовательной компрессии и декомпрессии.

Функция ввода данных о размещении файла на диске осуществляется через чтение переменных окружения. Для функции вывода информации о размерах исходного, сжатого и распакованного файлов необходимо вывести имя файла и его размер в байтах. Необходимо предусмотреть корректный выход из программы по системному прерыванию SIGINT.

3.2.1 Консольное приложения для тестирования подсистемы

Приложение для проведения тестирования подсистемы сжатия данных для большей наглядности и структурированности состоит из функциональных частей, реализованные в виде подпрограмм.

```
func main() {
	fmt.Println("---Архиватор---")
	fmt.Println("Тестирование сжатия")

	flag.Parse() // Получение параметров окружения

	filename := flag.Arg(0)

	if filename == "" {
		fmt.Println("Укажите файл для сжатия")
		os.Exit(1)
	}

	compressTest(filename, filename+".compressed")

	fi, err := os.Stat(filename)
	if err != nil {
		log.Fatalln(err)
	}
	size1 := fi.Size()
	fi2, err := os.Stat(filename + ".compressed")
	if err != nil {
		log.Fatalln(err)
	}
	size2 := fi2.Size()

	fmt.Println("Файл сжат с", size1, "байт до", size2, "байт, т.е. в",
float64(size1)/float64(size2), "раз")

	decompressTest(filename+".compressed", filename+".decompressed")

	fi, err := os.Stat(filename + ".compressed")
	if err != nil {
		log.Fatalln(err)
	}
	size1 := fi.Size()
	fi2, err := os.Stat(filename + ".decompressed")
	if err != nil {
		log.Fatalln(err)
	}
	size2 := fi2.Size()

	fmt.Println("Файл распакован с", size1, "байт до", size2, "байт, т.е. в",
float64(size1)/float64(size2), "раз")

	if isFilesOK(filename, filename+".decompressed") {
		fmt.Println("Файлы равнозначны")
	} else {
		fmt.Println("Файлы отличаются")
	}
}
```

В данном приложении используются три подпрограммы, выполняющих основной функционал:

1. compressTest — выполняет сжатие исходного файла и запись сжатого файла на диск;

```
func compressTest(inputFileName string, OutputFileName string) {
	rawfile, err := os.Open(inputFileName)
	if err != nil {
		fmt.Println(err)
		os.Exit(1)
	}
	defer rawfile.Close()

	info, _ := rawfile.Stat()
	var size int64 = info.Size()
	rawbytes := make([]byte, size)

	buffer := bufio.NewReader(rawfile)
	_, err = buffer.Read(rawbytes)

	if err != nil {
		fmt.Println(err)
		os.Exit(1)
	}

	var buf bytes.Buffer
	writer := compressor.NewWriter(&buf)
	writer.Write(rawbytes)
	writer.Close()

	err = ioutil.WriteFile(OutputFileName, buf.Bytes(), info.Mode())
	if err != nil {
		fmt.Println(err)
		os.Exit(1)
	}

	fmt.Printf("%s сжат в %s\n", inputFileName, OutputFileName)
}
```

2. decompressTest — выполняет декомпрессию сжатого файла и запись на диск распакованного файла;

```
func decompressTest(inputFileName string, OutputFileName string) {
	rawfile, err := os.Open(inputFileName)
	if err != nil {
		fmt.Println(err)
		os.Exit(1)
	}
	defer rawfile.Close()

	info, _ := rawfile.Stat()
	var size int64 = info.Size()
	rawbytes := make([]byte, size)

	buffer := bufio.NewWriter(rawfile)
	_, err = buffer.Write(rawbytes)
	if err != nil {
		fmt.Println(err)
		os.Exit(1)
	}

	var buf bytes.Buffer
	writer := compressor.NewReader(&buf)
```

```
	writer.Read(rawbytes)
	writer.Close()

	err = ioutil.WriteFile(OutputFileName, buf.Bytes(), info.Mode())
	if err != nil {
		fmt.Println(err)
		os.Exit(1)
	}

	fmt.Printf("%s распакован в %s\n", inputFileName, OutputFileName)
}
```

3. isFilesOK — проверяет побайтовую эквивалентность двух файлов (сжатого и распакованного).

```
const chunkSize = 64000

func isFilesOK(file1, file2 string) bool {
    // Check file size ...
    f1, err := os.Open(file1)
    if err != nil {
        log.Fatal(err)
    }

    f2, err := os.Open(file2)
    if err != nil {
        log.Fatal(err)
    }
    for {
        b1 := make([]byte, chunkSize)
        _, err1 := f1.Read(b1)

        b2 := make([]byte, chunkSize)
        _, err2 := f2.Read(b2)

        if err1 != nil || err2 != nil {
            if err1 == io.EOF && err2 == io.EOF {
                return true
            } else if err1 == io.EOF || err2 == io.EOF {
                return false
            } else {
                log.Fatal(err1, err2)
            }
        }
        if !bytes.Equal(b1, b2) {
            return false
        }
    }
}
```

3.3 Результаты модульного и функционального тестирования

По результатам тестирования трёх модулей (сжатия, распаковки и сбора статистики) подсистемы сжатия данных с использованием разностных алгоритмов можно говорить о готовности к интеграции каждого модуля в единую подсистему. Для оценки результатов интеграции необходимо прибегнуть к функциональному тестированию.

После запуска реализованного выше консольного приложения для тестирования подсистемы сжатия данных без передачи имени файла в виде аргумента на экран выведется информация о том, что необходимо указать имя файла. Если указать имя исходного файла в виде аргумента, то будут произведены компрессия, декомпрессия и сравнение файлов (см. рис. 11).

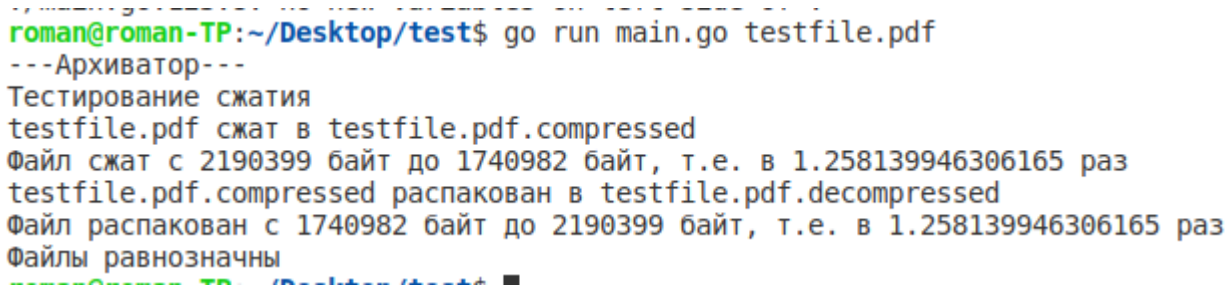

Рисунок 11 - Пример работы консольного приложения для тестирования подсистемы сжатия данных

В результате произведённых выше процедур в директории рядом с исходным файлом (testfile.pdf) появились 2 новых файла: сжатый (testfile.pdf.compressed) и распакованный (testfile.pdf.decompressed). Размеры и контрольные суммы исходного и разархивированного файлов совпадают, следовательно, подсистема сжатия данных не теряет данные на этапах сжатия и распаковки. Заметим, что исходный файл был программно синтезирован и в результате сжатия разностными алгоритмами получен коэффициент компрессии больше 1.

3.4 Оценочное тестирование

Оценочное тестирование — вид тестирования программного обеспечения, включающий в себя тестирование производительности, основной задачей которого является сбор показателей (метрик), времени отклика программного продукта в условиях, максимально приближенным или превышающим условия использования подсистемы после внедрения в конечный продукт, тестирование требований к памяти.

3.4.1 Определение основных показателей и ограничений

Основными критериями для оценки алгоритмов (и программных продуктов) сжатия данных являются:

1. Время, затраченное на сжатие данных;
2. Время, затраченное на распаковку данных;
3. Объем дополнительной выделяемой оперативной памяти, необходимой для сжатия тестового образца файла;
4. Объем дополнительной выделяемой оперативной памяти, необходимой для разархивирования тестового образца файла;
5. Степень сжатия.

Заметим, что необходимо рассмотреть 2 случая передачи и хранения данных для сжатия:

1. Данные передаются последовательно или представлены в виде одного файла. Это позволяет оценить эффективность алгоритмов в случае поточного сжатия.
2. Данные предполагают случайный доступ к нескольким файлам. В данном случае можно оценить эффективность алгоритма в случае сжатия отдельных файлов, основываясь на других файлах.

В качестве входных данных для режима поточного сжатия возьмём файл с показаниями датчиков температуры, положения в пространстве и времени

регистрации, полученные синтетическим путём (см. Ошибка: источник перекрёстной ссылки не найден).

В качестве входных данных для режима сжатия со случайным доступом загрузим из сети Интернет главную веб-страницу поисковой системы Яндекс (https://yandex.ru/) (см. Ошибка: источник перекрёстной ссылки не найден).

Рассмотрим так же случай, когда на сжатие подан файл, не содержащий регулярности в данных (например, уже сжатое изображение или аудио-файл).

3.4.2 Результаты тестирования и сравнение с аналогами

В результате проведённых тестов получены следующие показатели (см. табл. 5, 6 и 7; показатели округлены до целых; объем занятой оперативной памяти при распаковке во всех случаях оценён как «Малое», т. к. в результате профилирования выяснено, что у всех алгоритмов объем примерно равен размеру кадра) (ООЗОЗУ - Относительный объем занятой ОЗУ). По графикам (см. Ошибка: источник перекрёстной ссылки не найден), составленным по данным таблицам, можно заметить, что подсистема сжатия разработанной подсистемы с использованием разностных алгоритмов похожа на библиотеку LZ4, а модули Brotli и Gzip дают стабильно высокий коэффициент компрессии на любых данных, но требуют значительных вычислительных ресурсов.

Таблица 5 - Результаты нагрузочного тестирования для поточного режима

Показатель\ Алгоритм	**Brotli**	**Gzip**	**LZ4**	**Разрабатываемая подсистема**
Время сжатия, мс	10	9	3	6
Время распаковки, мс	2	2	1	1
ООЗОЗУ сжатие	~10	~10	5	3
ООЗОЗУ распаковка	Малое	Малое	Малое	Малое
Степень сжатия	5	5	2	5

Таблица 6 - Результаты нагрузочного тестирования для режима с случайным доступом

Показатель\ Алгоритм	**Brotli**	**Gzip**	**LZ4**	**Разрабатываемая подсистема**
Время сжатия, мс	10	9	5	6
Время распаковки, мс	2	2	1	1
ООЗОЗУ, сжатие	~10	~10	4	3
ООЗОЗУ, распаковка	Малое	Малое	Малое	Малое
Степень сжатия	6	5	2	6

Таблица 7 - Результаты нагрузочного тестирования при подаче файлов, не содержащих регулярных данных

Показатель\ Алгоритм	**Brotli**	**Gzip**	**LZ4**	**Разрабатываемая подсистема**
Время сжатия, мс	10	9	2	6
Время распаковки, мс	2	2	1	1
ООЗОЗУ сжатие	~10	~10	5	3
ООЗОЗУ, распаковка	Малое	Малое	Малое	Малое
Степень сжатия	2.9	2.8	1.2	~1

По результатам, полученным в результате тестирования, можно говорить о том, что подсистема с использованием разностных алгоритмов хорошо проявила себя в сжатии в поточном режиме (см. табл. 5) и её ближайшим конкурентом является алгоритм LZ4. Для режима сжатия файлов, содержащих регулярности и предполагающих случайный доступ (см. табл. 6), разрабатываемая подсистема продемонстрировала хорошие показатели по времени распаковки, объему затрачиваемой ОЗУ для распаковки и степени сжатия. По результатом сжатия файлов, не содержащих регулярности данных (см. табл. 7), можно говорить о непригодности разработанной подсистемы с использованием разностных алгоритмов для сжатия случайных файлов.

ЗАКЛЮЧЕНИЕ

В соответствии с техническим заданием произведено проектирование и реализация подсистемы сжатия данных с использованием разностных алгоритмов.

Подсистема сжатия с использованием разностных алгоритмов предназначена для использования в системах приема и передачи данных, в которых циркулирует информация, обладающая постоянной структурой или включающая в себя часто повторяющиеся фрагменты информации. Для облегчения задачи интеграции в уже существующие системы разработанная подсистема сжатия данных имеет интерфейс, аналогичный библиотеке «gzip». Также разработчику предоставлена возможность сжимать отдельные файлы на основе других файлов путём конкатенации этих файлов во входном потоке, что позволяет интегрировать эту библиотеку для сжатия файлов, находящихся в распределённых СХД.

Подсистема сжатия данных с использованием разностных алгоритмов отлично проявила себя для сжатия регулярных файлов (веб-страниц, дампы показаний датчиков и т.п.). Преимуществом над аналогами является нетребовательность к размерам оперативной памяти и вычислительной мощности устройства, которому надлежит распаковывать сжатые данные.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Орищенко, В.И.; Санников, В.Г.; Свириденко, В.А. Сжатие данных в системах сбора и передачи информации: .- М.: Изд-во Радио и связь. 1985.- 184с.
2. Ватолин Д., Ратушняк А., Смирнов М., Юкин В Методы сжатия данных. Устройство архиваторов, сжатие изображений и видео: .- М.: Изд-во ДИАЛОГ-МИФИ. 2002.- 384с.
3. Уэлстил С. Фракталы и вейвлеты для сжатия изображений в действии: Учебное пособие.- М.: Изд-во Триумф. 2003.- 320с.
4. Ян Ричардсон Видеокодирование. X.264 и MPEG-4 - стандарты нового поколения : .- М.: Изд-во Техносфера. 2005.- 369с.
5. Миано Дж. Форматы и алгоритмы сжатия изображений в действии: .- М.: Изд-во Триумф. 2003.- 336с.
6. Краш-курс по интерфейсам в Go [Электронный ресурс] // Habr. 2006-2020. URL: https://habr.com/en/post/276981/ (Дата обращения: 21.11.2019)
7. Окулов С.М., Пестов О.А. Динамическое программирование: .- М.: Изд-во БИНОМ. Лаборатория знаний. 2012.- 299с.
8. Калеб Докси Введение в программирование на Go: .- : Изд-во CCAL. 2015.- 90с.
9. Окасаки К. Чисто функциональные структуры данных : .- М.: Изд-во ДМК Пресс. 2016.- 252с.
10. Иванова Г.С. Технология программирования: Учебник для вузов.- М.: Изд-во Кнорус. 2013.- 333с.
11. Эрджиес К. Распределенные системы реального времени. Теория и практика : .- М.: Изд-во ДКМ Пресс. 2018.- 382с.
12. Дадян Э.Г. Данные: хранение и обработка: Учебник.- М.: Изд-во НИЦ ИНФРА-М . 2018.- 205с.
13. Сэломон Д. Сжатие данных, изображений и звука: .- М.: Изд-во Техносфера. 2004.- 368с.

14. Иванова Г.С. Программирование: Учебник.- М.: Изд-во Кнорус. 2013.- 432с.
15. Иванова Г.С. Основы программирования: Учебник для втузов.- М.: Изд-во МГТУ им. Н.Э. Баумана. 2007.- 416с.

ПРИЛОЖЕНИЯ

ПРИЛОЖЕНИЕ А

Техническое задание
Листов 7

ПРИЛОЖЕНИЕ Б

Руководство пользователя
Листов 4

ПРИЛОЖЕНИЕ В

Графический материал
Листов 6

ПРИЛОЖЕНИЕ Г

Фрагменты исходного кода
Листов 2

ПРИЛОЖЕНИЕ Д

Фрагменты тестовых данных
Листов 1