

Министерство образования Республики Беларусь

Учреждение образования  
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

К ЗАЩИТЕ ДОПУСТИТЬ  
Зав. каф. ЭВМ  
\_\_\_\_\_ Б.В. Никульшин

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА  
к дипломному проекту  
на тему  
ПРОГРАММНЫЙ МОДУЛЬ СЖАТИЯ ДАННЫХ НА ОСНОВЕ  
РЕКОМЕНДАЦИЙ СТАНДАРТА CCSDS

БГУИР ДП 1–40 02 01 01 013 ПЗ

Студент	А.А. Волчков
Руководитель	Л.П. Поденок
Консультанты:	
от кафедры ЭВМ	Л.П. Поденок
по экономической части	Ф.М. Файзрахманов
Нормоконтролер	А.С. Сидорович
Рецензент	

МИНСК 2019

## РЕФЕРАТ

Дипломный проект предоставлен следующим образом. Электронные носители: 1 компакт-диск. Чертежный материал: 6 листов формата A1. Пояснительная записка: 110 страниц, 42 рисунка, 6 таблиц, 13 литературных источников, 3 приложения.

Ключевые слова: изображение, сжатие, компрессия, сжатие без потерь, алгоритм, гиперспектральное, мультиспектральное, энтропийное кодирование, адаптивный предсказатель, кодирование Голомба-Райса.

Предметной областью данной разработки являются гиперспектральные и мультиспектральные изображения, получаемые в процессе съемки специальной аппаратурой объектов материального мира. Объектами разработки являются алгоритмы сжатия без потерь и восстановления гиперспектральных изображений.

Целью дипломного проекта является создание библиотеки функций, с помощью которых можно осуществлять сжатие без потерь и восстановление гиперспектральных и мультиспектральных изображений.

Разработка библиотеки функций велась на языке C стандарта C99. Для написания модульных тестов использовался фреймворк Catch2. В качестве среды разработки использовался Qt Creator.

В результате проведенной работы была создана библиотека функций, с помощью которых можно производить сжатие без потерь и восстановление гиперспектральных изображений. Была написана демонстрационная программа, способная выполнять сжатие и восстановление набора PGM-файлов, каждый из которых играет роль отдельного спектрального слоя гиперспектрального изображения. Сравнение результатов сжатия программой тестового набора данных с результатами работы универсальных программ-архиваторов показало высокую эффективность разработанной библиотеки функций.

Разработанная библиотека может найти применение в организациях аэрокосмической отрасли, непосредственно связанных с приемом, передачей и хранением гиперспектральных изображений.

Экономическая эффективность разработанного программного продукта доказана результатами расчетов.

Проект является завершенным и готов к внедрению в коммерческие проекты. Работа над проектом может быть продолжена в направлении оптимизации вычислений, в том числе и с использованием графических карт. Также можно дополнить разработанную библиотеку иными вариантами энтропийного кодера, а также реализовать упрощенную версию адаптивного предсказателя, имеющую меньшую вычислительную сложность, что будет способствовать увеличению скорости работы программы, но может ухудшить эффективность сжатия.

## СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....	7
1 ОБЗОР ЛИТЕРАТУРЫ.....	9
1.1 Мультиспектральные и гиперспектральные изображения.....	9
1.2 Архитектура модуля сжатия-восстановления гиперспектральных данных.....	10
1.3 Методы и алгоритмы кодирования цифровых сигналов.....	11
1.3.1 Прогнозирующее кодирование.....	11
1.3.2 Кодирование с преобразованием.....	13
1.3.3 Преобразование Карунена-Лоэва.....	14
1.3.4 Дискретное косинусное преобразование.....	15
1.3.5 Дискретное вейвлет-преобразование.....	15
1.3.6 Энтропийное кодирование.....	16
2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ.....	18
2.1 Общая информация о структурных блоках.....	18
2.2 Блок чтения исходного гиперспектрального изображения.....	18
2.3 Блок адаптивного предсказателя.....	19
2.4 Блок энтропийного кодера.....	19
2.5 Блок записи сжатого файла.....	19
2.6 Блок чтения сжатого гиперспектрального изображения.....	20
2.7 Блок энтропийного декодера.....	20
2.8 Блок восстановления исходного изображения.....	20
2.9 Блок записи восстановленного изображения.....	20
2.10 Блок чтения параметров командной строки.....	21
2.11 Блок вывода информационных сообщений.....	21
3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ.....	23
3.1 Модуль адаптивного предсказателя.....	25
3.2 Модуль энтропийного кодера.....	30
3.3 Модуль сохранения сжатого изображения.....	32
3.4 Модуль загрузки сжатого изображения.....	36
3.5 Модуль загрузки исходного изображения.....	37
3.6 Модуль сохранения исходного изображений.....	37
4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ.....	38
4.1 Алгоритм функции вычисления локальной суммы.....	38
4.2 Алгоритм функции вычисления вектора локальных разниц.....	39
4.3 Алгоритм функции инициализации вектора весов.....	41
4.4 Алгоритм функции вычисления отображенного остатка предсказания.....	42
4.5 Алгоритм функции восстановления исходного значения элемента.....	43
4.6 Алгоритм работы адаптивного предсказателя.....	44
4.7 Алгоритм работы элементарно-адаптивного энтропийного кодера.....	48
4.8 Алгоритм работы элементарно-адаптивного энтропийного декодера.....	51
5 ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ.....	54
5.1 Описание исходных данных.....	54
5.2 Тестирование работы программы в режиме сжатия.....	55

5.3 Тестирование работы программы в режиме восстановления.....	58
5.4 Тестирование работы программы с различными параметрами сжатия. ....	61
5.5 Тестирование программы на ввод некорректных данных.....	64
5.6 Сравнение с результатами универсальных алгоритмов сжатия.....	70
6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ.....	74
6.1 Руководство разработчика.....	74
6.1.1 Требования к программному обеспечению.....	74
6.1.2 Поддерживаемый интерфейс.....	74
6.2 Руководство конечного пользователя.....	76
7 ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ ПРОГРАММНОГО МОДУЛЯ СЖАТИЯ ТРЕХМЕРНЫХ ДАННЫХ.....	80
7.1 Характеристика программного модуля сжатия трехмерных данных.....	80
7.2 Расчет затрат на разработку программного средства организации питания сотрудников.....	81
7.3 Расчет экономической эффективности реализации на рынке программного модуля сжатия трехмерных данных.....	82
7.4 Расчет показателей эффективности инвестиций в разработку программного модуля сжатия трехмерных данных.....	83
ЗАКЛЮЧЕНИЕ.....	85
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ.....	86
ПРИЛОЖЕНИЕ А.....	87
ПРИЛОЖЕНИЕ Б.....	109
ПРИЛОЖЕНИЕ В.....	110

## ВВЕДЕНИЕ

Мультиспектральные изображения обычно состоят из серий изображений одного и того же объекта в разных спектральных диапазонах. Как правило, мультиспектральные изображения получаются с использованием цветных полосовых светофильтров. Гиперспектральные изображения возникли в результате дальнейшего развития спектрографии. Если мультиспектральные изображения имели спектральное разрешение 50–100 нм, то гиперспектральные сенсоры дают точность в 5–10 нм, тем самым сильно повышая размерность пикселей изображений.

Гиперспектральные изображения представляют собой трехмерные массивы данных, заключающие в себе пространственную информацию об объекте, дополненную спектральной информацией по каждой пространственной координате. Таким образом, каждой точке изображения соответствует спектр, полученный в этой точке снимаемого объекта.

Области использования гиперспектральных изображений обширны. Среди самых актуальных применений можно отметить задачи ДЗЗ (дистанционного зондирования Земли). Результаты обработки гиперспектральных данных ДЗЗ используются в агропромышленном комплексе, геологической разведке, экологическом мониторинге. Анализ гиперспектральных изображений ДЗЗ может применяться для экологического мониторинга местности, например, при расчете плотности различных видов зеленых насаждений в городе или степени заполняемости парковочных площадок. На сегодня существует несколько исследовательских проектов, в рамках которых создаются гиперспектральные снимки земной поверхности: AVIRIS, HYDICE, HYPERION, HyMap.

Объем гиперспектральных изображений весьма велик. Так, например, фрагменты космических снимков спектрометра AVIRIS, имеют пространственный размер  $1086 \times 614$  пикселей и 224 спектральные компоненты. На описание яркости каждого пикселя в каждой компоненте отводится два байта (16 бит). Таким образом, «спектральный портрет» каждого пикселя задают 224 16-битных целых числа, а «трехмерный куб» гиперспектрального изображения составляет 298 728 192 байта (примерно 300 МБ). Подобный объем данных порождает массу сложностей по его передаче и хранению. Логичным решением данной проблемы является применение эффективных алгоритмов компрессии.

Целью данного дипломного проекта является разработка программного модуля, реализующего алгоритмы компрессии и декомпрессии гиперспектральных и мультиспектральных изображений без потерь, представленный в рекомендациях стандарта CCSDS (Consultative Committee for Space Data Systems) – международного комитета, занимающегося разработкой стандартов и рекомендаций для космических информационных систем. В качестве языка разработки предполагается использовать язык C стандарта C99.

Исходя из поставленной цели, были определены следующие задачи:

- изучить рекомендации стандарта CCSDS для сжатия гиперспектральных и мультиспектральных изображений без потерь;
- на основе стандарта CCSDS разработать и реализовать алгоритм подготовки данных к энтропийному кодированию;
- изучить существующие алгоритмы энтропийного кодирования данных, выбрать подходящий алгоритм и реализовать его;
- реализовать алгоритм декодирования данных на основе известных параметров энтропийного кодирования;
- разработать и реализовать алгоритм восстановления исходного изображения на основе декодированных данных;
- разработать демонстрационную программу, выполняющую компрессию и декомпрессию трехмерных изображений с выводом информации о затраченном времени и степени компрессии данных.

# 1 ОБЗОР ЛИТЕРАТУРЫ

## 1.1 Мультиспектральные и гиперспектральные изображения

Гиперспектральные изображения можно представить в виде трехмерной структуры данных, состоящей из слоев двумерных изображений, полученных в результате съемки некоторого объекта (например, части Земной поверхности) в очень узком спектральном диапазоне (рисунок 1.1).

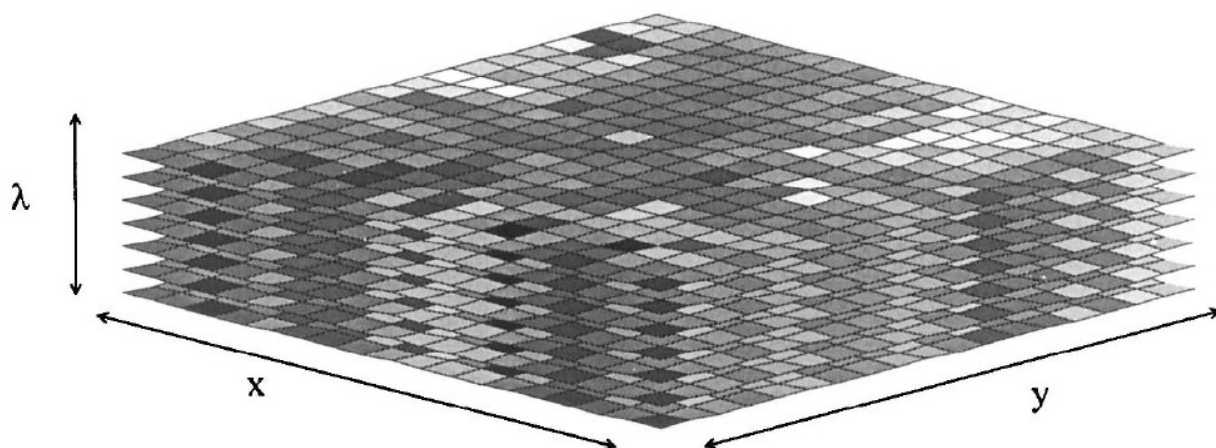


Рисунок 1.1 – Трехмерная структура гиперспектральных данных [1]

В то время как человеческий глаз видит цвет видимого света в основном в трех спектральных каналах (длинные волны воспринимаются как красные, средние длины волн – как зеленые, а короткие волны – как голубые), гиперспектральные сенсоры (дистанционные спектрометры) получают изображение во многих (очень узких) каналах, расположенных в видимом, ближнем, среднем и тепловом инфракрасном диапазоне спектра. Такие системы обычно имеют 200–400 каналов данных, которые позволяют получить эффективный непрерывный спектр яркости для каждого пикселя сцены [2].

Мультиспектральные изображения охватывают небольшое количество спектральных каналов (обычно не более 20) в разных областях электромагнитного спектра (инфракрасной, видимого света, ультрафиолетовой, рентгеновской). Различать гипер- и мультиспектральные изображения можно по числу спектральных полос или по типу измерения, которым получено изображение. При мультиспектральном способе получение изображений происходит дискретно и с созданием нескольких изображений.

Мультиспектральное получение изображений происходит по типу цветной фотографии. Многоспектральный датчик может иметь много полос, охватывающих спектр от видимого до длинноволновой инфракрасной области спектра. Мультиспектральные изображения не воспроизводят

«спектр» объекта. При гиперспектральной обработке происходит визуализация узких спектральных линий.

Так датчик с 20 полосами может быть гиперспектрального характера в том случае, если эти полосы, шириной 10 нм каждая, перекрывают диапазон от 500 до 700 нм. В то же время датчик с 20 дискретными линиями покрытия VIS, NIR, SWIR, MWIR и LWIR будет рассматриваться как мультиспектральный.

## 1.2 Архитектура модуля сжатия-восстановления гиперспектральных данных

Базовая архитектура системы сжатия гиперспектральных изображений приведена на рисунке 1.2. В данной архитектуре выделяются блоки предобработки, сжатия данных, оценки искажений, восстановления, постобработки и классификации.

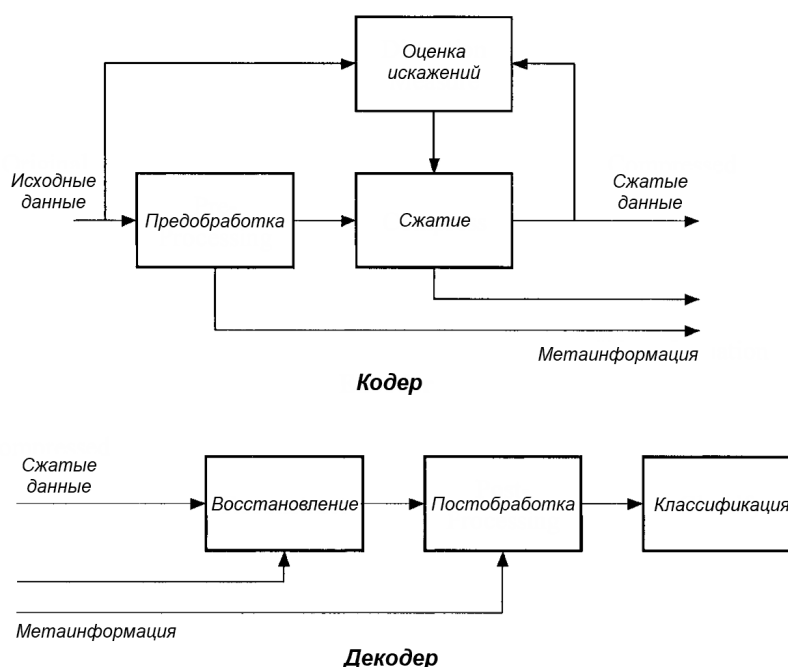


Рисунок 1.2 – Архитектуры кодера и декодера для целостной системы сжатия гиперспектральных данных [1]

Этап предварительной обработки обычно включает в себя применение некоторых простых обратимых операций, направленных на повышение эффективности работы алгоритмов сжатия. В качестве примеров предварительной обработки в литературе предложены переупорядочение и нормализация каналов, применение метода главных компонент и предварительная кластеризация.

Этап сжатия заключается в использовании одного из стандартных алгоритмов сжатия изображений, таких как векторное квантование или



кодирование с преобразованием, которые были модифицированы в соответствии с трехмерной природой гиперспектрального изображения. На данном этапе также происходит оценка искажений, вносимых алгоритмом сжатия, то есть степени соответствия восстановленного изображения оригинальному. Выходом рассматриваемого процесса являются сжатые гиперспектральные данные, а также информация, требуемая для их восстановления (метаинформация).

На этапах восстановления и постобработки происходят процессы, обратные соответственно сжатию и предобработке, приводящие к полному (или с некоторыми потерями) восстановлению оригинального изображения.

Наконец, на этапе классификации обычно используется один из стандартных методов классификации дистанционного зондирования, например классификация по методу максимального правдоподобия, извлечение граничных признаков.

### **1.3 Методы и алгоритмы кодирования цифровых сигналов**

#### **1.3.1 Прогнозирующее кодирование**

Ранние исследования в области сжатия гиперспектральных изображений базировались на методе дифференциальной импульсно-кодовой модуляции (ДИКМ). Под импульсно-кодовой модуляцией (ИКМ) понимают процесс преобразования аналоговой формы сигнала в цифровую посредством применения трех операций: дискретизации по времени, квантования по амплитуде и кодирования [3]. При дифференциальной ИКМ форма выходного сигнала определяется разницей между текущим и предыдущим измеренными значениями. Подобный подход лежит в основе прогнозирующего (предиктивного) кодирования данных.

Предсказание обычно основывается на окрестности прогнозирования – значениях данных, которые пространственно, во времени и спектрально примыкают друг к другу. Эти значения используются в расчете прогноза на том основании, что, в естественных сигналах, таких как аудио, фотографические изображения, обычно существует высокая степень корреляции между последовательными значениями выборки, и, следовательно, предыдущая выборка во времени, пространстве или частоте сделает хороший прогноз текущей выборки. Затем прогноз вычитается из текущего значения и передается только разность или остаток прогноза. Если существует большая корреляция между соседними выборками, то итоговый сигнал потребует меньшую ширину канала передачи, чем исходный. В декодере же генерируется то же самое предсказание, что и при кодировании, а исходное значение сигнала восстанавливается путем сложения переданного остатка с предсказанным значением. Рисунок 1.3 иллюстрирует описанные выше процессы кодирования и декодирования.

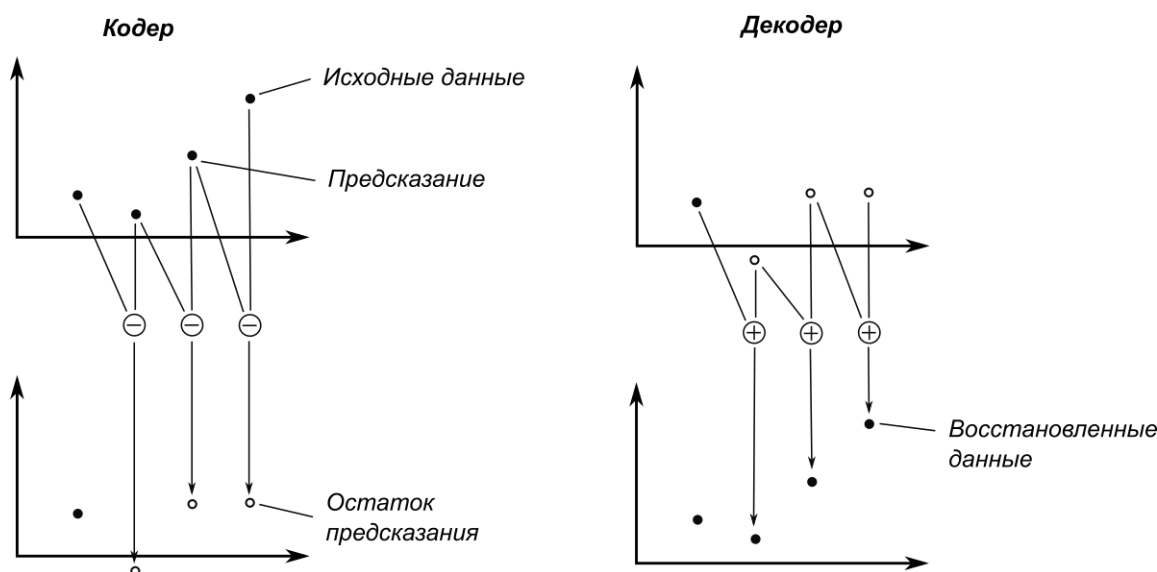


Рисунок 1.3 Процессы кодирования и декодирования сигнала [1]

При сжатии гиперспектральных изображений, в качестве прогноза обычно используется линейная комбинация значений пикселей, пространственно и спектрально смежных с текущим пикселем (рисунок 1.4). Важно отметить, что для прогнозирования текущего значения пикселя можно использовать только те пиксельные элементы, которые ранее были закодированы и переданы декодеру.

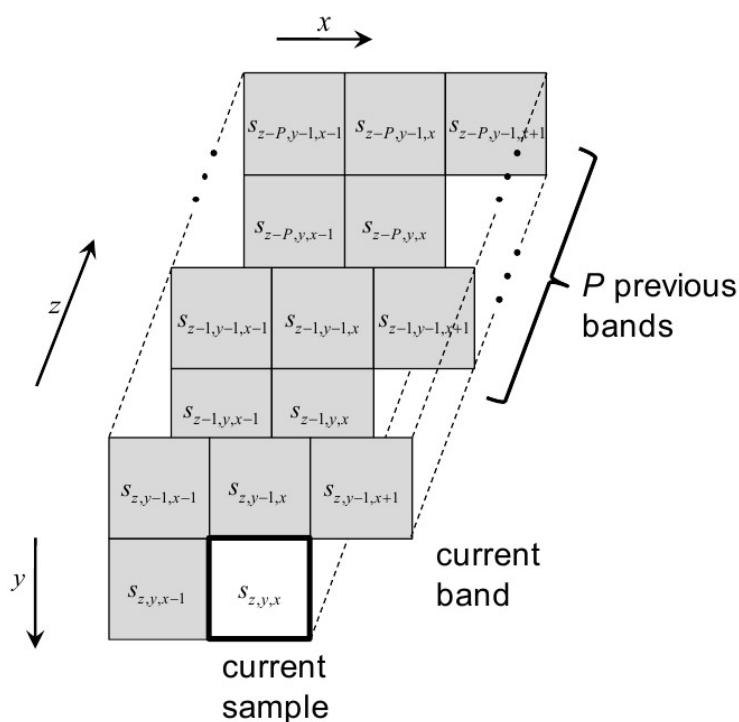


Рисунок 1.4 – Типичная окрестность прогнозирования в гиперспектральных изображениях [4]

### 1.3.2 Кодирование с преобразованием

При кодировании с преобразованием последовательность исходных значений заменяется на набор амплитудных коэффициентов базисных функций, по которым раскладывается данный сигнал. Для более простых преобразований базисные функции обычно представляют собой синусоиды с дискретно возрастающей частотой. На рисунке 1.5 изображен процесс кодирования с преобразованием.

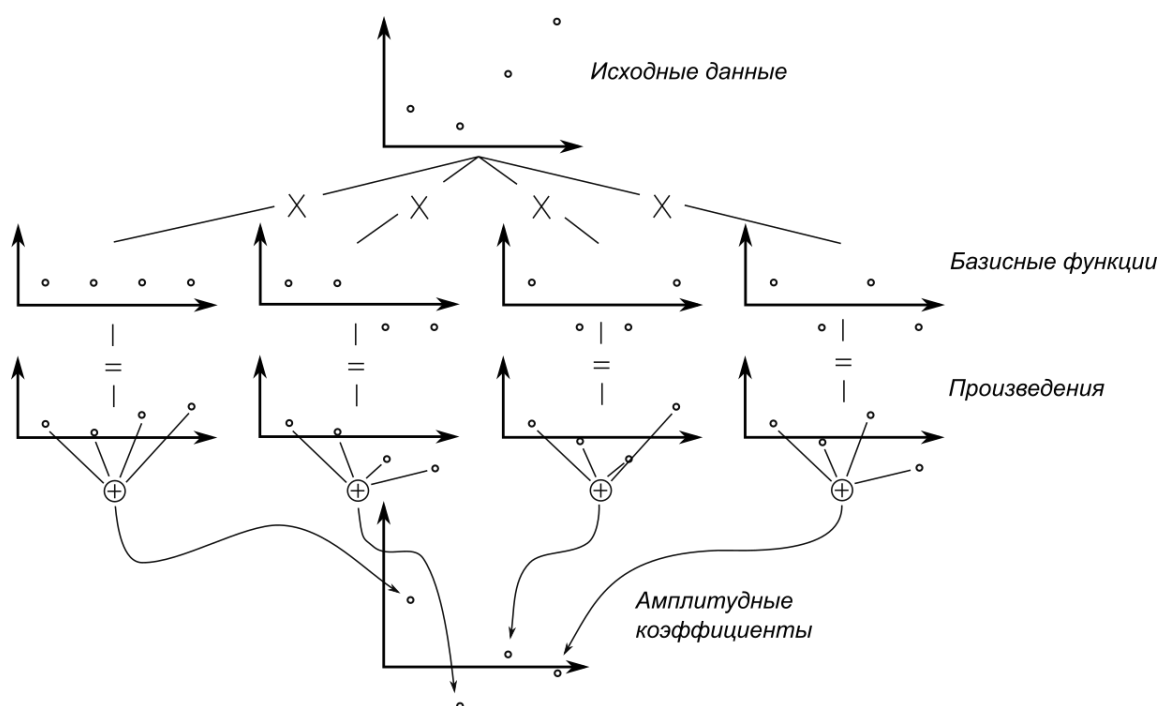


Рисунок 1.5 – Процесс кодирования с преобразованием [1]

Если исходные значения последовательности близки по форме данной базисной функции, то значение ее амплитудного коэффициента будет большим положительным числом, если противоположны по форме – то большим отрицательным числом, а если не имеют одинаковой формы, то ее коэффициент будет равен нулю. При этом, зачастую, форма исходного сигнала с достаточной точностью может быть выражена путем сложения всего нескольких базисных функций с учетом их амплитудных коэффициентов, а влиянием других функций можно пренебречь. В таком случае, для восстановления исходного сигнала достаточно передать или сохранить лишь номера значимых функций и их коэффициенты, за счет чего и достигается эффект сжатия. Рисунок 1.6 иллюстрирует процесс обратного преобразования, приводящий к восстановлению исходных данных.

Главное требование, предъявляемое к набору базисных функций – их ортогональность. Множество непрерывных функций действительного переменного  $\{U_n(t)\} = \{U_0(t), U_1(t), \dots\}$  называется ортогональным на

интервале  $[t_0, t_0+T]$ , если

$$\int_{t_0}^{t_0+T} U_m(t) U_n(t) dt = \begin{cases} c, & \forall m=n, \\ 0, & \forall m \neq n. \end{cases} \quad (1.1)$$

При  $c = 1$  множество  $\{U_n(t)\}$  называется ортонормированным [5].

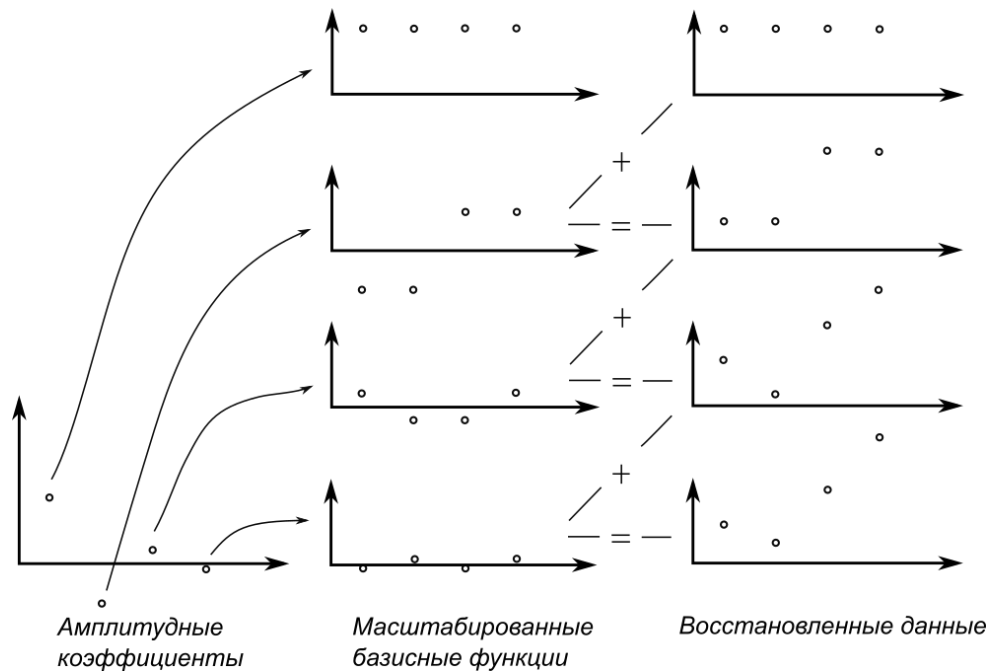


Рисунок 1.6 – Процесс обратного преобразования [1]

### 1.3.3 Преобразование Карунена-Лозва

Преобразование Карунена-Лозва (ПКЛ) считается наиболее эффективным преобразованием, приводящим к максимизации количества энергии, содержащейся в наименьшем числе коэффициентов преобразования. Базисные векторы для ПКЛ генерируются для каждого конкретного набора данных с использованием их статистических свойств. Процесс ПКЛ по существу идентичен методу главных компонент, где преобразование используется для уменьшения размерности данных. Если данные из многих измерений в исходном пространстве данных тесно связаны, то метод главных компонент позволяет представить данные для этих измерений с помощью одного измерения в преобразованной системе координат. Недостатком ПКЛ является то, что он зависит от данных, и оптимальные базисные векторы должны быть рассчитаны и переданы в качестве побочной информации. Эту проблему можно решить с помощью фиксированного набора базисных векторов, известных как кодеру, так и декодеру. Общим преобразованием,

использующим такой набор фиксированных базисных векторов, является дискретное косинусное преобразование.

### 1.3.4 Дискретное косинусное преобразование

Дискретное косинусное преобразование (ДКП) является одним из самых распространенных преобразований, используемых при сжатии с потерями изображений (JPEG) и видео (MPEG). Элементы базисных векторов для данного преобразования могут быть найдены по формуле:

$$t_{i,j} = k_j \sqrt{\frac{2}{N}} \cos\left(\frac{(2i+1)j\pi}{2N}\right), \text{ при } i, j = 0, 1, \dots, N-1, \quad (1.2)$$

где  $t_{i,j}$  –  $i$ -ый элемент  $j$ -го базисного вектора;

$N$  – размер базисного вектора;

$k_j$  – коэффициент, определяемый как:

$$k_j = \begin{cases} \frac{1}{\sqrt{2}}, & j=0 \\ 1, & j \neq 0 \end{cases}. \quad (1.3)$$

К преимуществам ДКП можно отнести существование алгоритмов быстрого преобразования, подобных алгоритмам быстрого преобразования Фурье (БПФ), существенно уменьшающие вычислительную сложность. Для преобразований с фиксированной размерностью вектора существуют также алгоритмы, позволяющие свести количество операций умножения к минимуму.

### 1.3.5 Дискретное вейвлет-преобразование

Дискретное вейвлет-преобразование, представляет собой процесс фильтрации поддиапазонов, исходных данных. Входной сигнал, путем применения соответствующих фильтров, разделяется на низкочастотную и высокочастотную составляющие (рисунок 1.7). Полученные составляющие подвергаются операции прореживания (децимации), так как после фильтрации избыточны, и после нее формируют представление сигнала в области вейвлет-преобразования. Результирующие выходные наборы обычно называются вейвлет-коэффициентами. Базисные векторы вейвлет-преобразования являются версиями импульсного отклика фильтра, используемого в процессе фильтрации поддиапазонов.

Чтобы получить восстановленный сигнал, выборки поддиапазонов сначала подвергаются повышению дискретизации с коэффициентом два, а

затем фильтруются для получения восстановленных версий исходного сигнала для верхних и нижних частот. Выходные данные фильтров восстановления затем суммируются для получения окончательного восстановленного сигнала (рисунок 1.7).

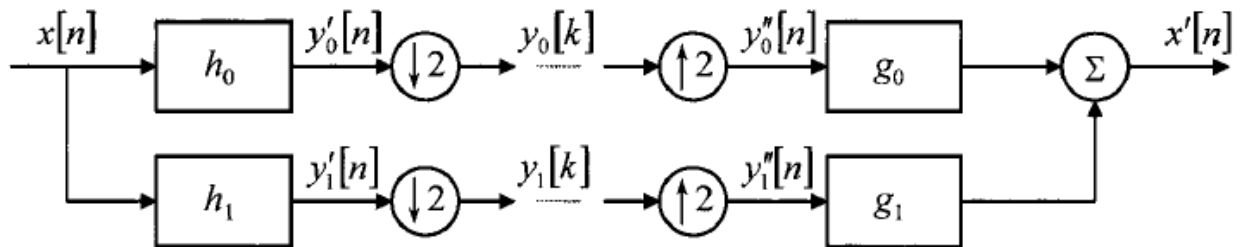


Рисунок 1.7 – Схема одномерного одноуровневого вейвлет-преобразования и последующего восстановления [1]

Процесс двухканальной декомпозиции может быть рекурсивно повторен на выборках нижних частот предыдущей стадии фильтрации, чтобы обеспечить декомпозицию исходного сигнала с большим разрешением. Для двумерных данных, таких как изображения, обычно применяются одномерные вейвлет-преобразования в горизонтальном и вертикальном направлениях.

### 1.3.6 Энтропийное кодирование

Энтропийное кодирование обычно является завершающим этапом любого алгоритма сжатия изображений. Оно включает в себя преобразование данных, полученных на предыдущих этапах кодирования в поток двоичных кодовых слов. Двумя основными методами, используемыми в энтропийном кодировании, являются кодирование Хаффмана и арифметическое кодирование.

При кодировании Хаффмана передаваемые значения данных представляются в виде заранее определенного алфавита символов или сообщений. Вероятность того, что каждое сообщение будет передано, измеряется, и каждому сообщению выделяется уникальный набор битов, называемый кодовым словом. Эти кодовые слова имеют различную длину, и самые короткие из них распределяются между сообщениями, имеющими наибольшую вероятность, в то время как более длинные кодовые слова распределяются между сообщениями, которые встречаются относительно редко. Таким образом, среднее число передаваемых битов будет меньше, чем если бы всем сообщениям было присвоено кодовое слово одинаковой длины.

Существует два основных недостатка кодирования Хаффмана. Во-первых, используемые кодовые слова фиксированы и должны быть известны и кодеру и декодеру. Это означает, что кодирование Хаффмана не может легко

адаптироваться к изменению распределения вероятностей в кодируемых сообщениях. Во-вторых, самое маленькое кодовое слово, которое можно использовать, – это один бит. В случаях, когда некоторое сообщение возникает очень часто, это может привести к неэффективному использованию доступной скорости передачи данных.

Эти две проблемы можно решить с помощью арифметического кодирования. Арифметическое кодирование является относительно новым методом энтропийного кодирования, который позволяет кодировать очень частые сообщения со средним значением менее одного бита на сообщение. Это достигается путем рассмотрения сообщений, передаваемых в группах фиксированной длины, и создания единого кодового слова, описывающего конкретную комбинацию сообщений, которые появляются в группе. Таким образом, группа из десяти сообщений с высокой вероятностью, например, может быть передана с кодовым словом длиной менее десяти бит. Арифметическое кодирование обычно пересчитывает статистику данных в процессе кодирования и, следовательно, может адаптироваться к изменяющимся вероятностям каждого сообщения. Недостатком арифметического кодирования является то, что оно имеет более высокую вычислительную сложность, чем кодирование Хаффмана.

## **2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ**

### **2.1 Общая информация о структурных блоках**

Исходя из поставленной цели и определенных задач, следуя рекомендациям стандарта CCSDS [4] о написании программных модулей сжатия гиперспектральных и мультиспектральных, разрабатываемый программный модуль можно разбить на следующие логические блоки:

- блок чтения исходного файла гиперспектрального изображения для его последующей компрессии;
- блок адаптивного предсказателя;
- блок энтропийного кодера;
- блок записи сжатого файла;
- блок чтения сжатого гиперспектрального изображения;
- блок энтропийного декодера;
- блок восстановления исходного изображения после декодирования;
- блок записи восстановленного изображения;
- блок чтения ключей командной строки;
- блок вывода информационных сообщений.

Перечисленные выше логические блоки приведены на структурной схеме ГУИР.400201.013 С1, отражающей их взаимосвязь.

Разрабатываемый программный модуль предназначен для выполнения двух основных операций:

- операции сжатия гиперспектрального или мультиспектрального изображения с указанными параметрами;
- операции восстановления оригинального изображения из сжатых данных.

Исходя из того, какую операцию необходимо выполнить в данном конкретном случае, цепочки взаимодействия структурных блоков между собой будут различны.

### **2.2 Блок чтения исходного гиперспектрального изображения**

Блок чтения исходного гиперспектрального изображения производит попытку открытия файла по пути, указанному пользователем через аргументы командной строки. При успешном открытии файла, в рассматриваемом блоке будут выполняться операции чтения метаданных гиперспектрального изображения (header), а также поиск и вычитывание самого «тела» (body) изображения, хранящего информацию о запечатленном объекте. Вычитанная информация далее передается блоку адаптивного предсказателя. При ошибке открытия файла, блок чтения исходного изображения выведет соответствующее информационное сообщение на экран и завершит выполнение программы.



## **2.3 Блок адаптивного предсказателя**

Блок адаптивного предсказателя, наравне с блоком энтропийного кодера, является одним из ключевых блоков всей программы. В рассматриваемом блоке производится первичная подготовка гиперспектральных данных для последующего их сжатия энтропийным кодером.

Суть подготовки данных для энтропийного кодера заключается в увеличении степени предсказуемости появления символов входного алфавита, а также уменьшении его размеров. Для этих целей используется адаптивный линейный алгоритм предсказания значения каждого пикселя изображения, основанного на значениях небольшой области его пикселей-соседей в трехмерном пространстве. Одним из достоинств данного алгоритма является его низкая вычислительная сложность. Далее выполняется вычисление остатка предсказания – разности между истинным значением пикселя и его предсказанным значением. Так как значение остатка предсказания может быть отрицательным, происходит трансляция этого значения в значение без знака.

Результатом работы блока является массив оттранслированных остатков предсказания, а также собранная по нему статистическая информация.

## **2.4 Блок энтропийного кодера**

В блоке энтропийного кодера выполняется, собственно, энтропийное кодирование данных, которые были подготовлены блоком адаптивного предсказателя, а именно, оттранслированных остатков предсказания. Кодирование выполняется с учетом параметров, указанных пользователем в аргументах командной строки, а также на основе собранной статистической информации о входных данных: размер алфавита, наибольшее и наименьшее значение, вероятности появления отдельных символов.

Результатом работы рассматриваемого блока является массив закодированных данных, а также метainформация, содержащая в себе параметры энтропийного кодирования, опираясь на которые будет производиться работа декодера при восстановлении оригинального изображения.

## **2.5 Блок записи сжатого файла**

В данном блоке выполняется формирование файла сжатого гиперспектрального изображения и последующая его запись на диск. Структура сжатого файла условно делится на «заголовок» (header) и «тело» (body). Заголовок файла записывается различная метainформация о

структуре исходного гиперспектрального изображения (линейные размеры изображения, количество спектральных каналов), о параметрах работы энтропийного кодера и адаптивного предсказателя. Само же сжатое изображение записывается в часть, именуемую «телом» сжатого файла.

## **2.6 Блок чтения сжатого гиперспектрального изображения**

Блок чтения сжатого гиперспектрального изображения, аналогично блоку, описанному в пункте 2.2, производит попытку открытия сжатого файла по пути, указанному пользователем по средствам аргументов командной строки. При успешном открытии файла, из начала файла будут вычитаны метаданные (header), и «тело» (body) сжатого изображения.

При ошибке открытия файла по указанному пути, блок чтения сжатого файла, рассматриваемый блок выведет соответствующее сообщение на экран и завершит работу программы.

## **2.7 Блок энтропийного декодера**

Блок энтропийного декодера производит восстановление закодированного изображения до состояния, аналогичного тому, что было подано на вход блока энтропийного кодера, а именно, до оттранслированных остатков предсказания. Информация, необходимая для работы энтропийного декодера берется из метаданных, сохраненных в заголовочной части сжатого файла.

## **2.8 Блок восстановления исходного изображения**

В рассматриваемом блоке выполняется восстановление оригинального гиперспектрального изображения из массива оттранслированных остатков предсказания, полученного от блока энтропийного декодера. Восстановление происходит на основе вычитанных из сжатого файла метаданных о работе адаптивного предсказателя, выполненной при процедуре сжатия изображения.

## **2.9 Блок записи восстановленного изображения**

Блок записи восстановленного изображения производит воссоздание структуры оригинального гиперспектрального изображения:

- производит формирование заголовка файла гиперспектрального изображения;
- заполняет сформированный заголовок метаинформацией;
- выполняет запись сформированного заголовка в файл;
- выполняет запись основной информации в «тело» файла.

## **2.10 Блок чтения параметров командной строки**

Так как разрабатываемая программа не имеет пользовательского графического интерфейса (GUI), то логичным решением является организация взаимодействия пользователя с программой посредством передачи аргументов командной строки.

Аргументы командной строки – это необязательные строковые аргументы, передаваемые операционной системой в программу при ее запуске. Программа может их использовать в качестве входных данных, либо проигнорировать. Подобно тому, как параметры одной функции предоставляют данные для параметров другой функции, так и аргументы командной строки предоставляют возможность людям или программам предоставлять входные данные для программы.

Рассматриваемый блок выполняется на самом старте программы. В нем происходит анализ аргументов командной строки, указанных пользователем при вводе команды запуска программы.

Обязательными аргументами, которые должен указать пользователь являются:

- пути до исходного файлов;
- компрессия или декомпрессия
- имя выходного файла.

При выполняемой операции компрессии, пользователь может указать также дополнительные параметры:

- число спектральных слоев, используемое адаптивным предсказателем;
- разрешение (количество бит) элементов вектора весов;
- параметры обновления вектора весов;
- начальные значения аккумулятора и счетчика для элементно-адаптивного энтропийного кодера.

При не указанном каком-либо из необязательных параметров, для него будет использовано значение по умолчанию.

При передаче неизвестных ключей, либо при неверных их значениях, рассматриваемый блок будет информировать пользователя об ошибке путем вывода справочной информации о допустимых ключах и приемах работы с программой.

## **2.11 Блок вывода информационных сообщений**

Разрабатываемая демонстрационная программа является консольной и не предполагает наличия графического пользовательского интерфейса. Но так как входные данные, в виде оригинальных или сжатых гиперспектральных изображений, имеют значительный объем, измеряемый сотнями мегабайт, работа программы может занять существенное для пользователя время. В

течение этого времени важно любым доступным способом поддерживать «обратную связь» с пользователем, информировать его о ходе выполнения программы, иначе пользователь может решить что программа попросту зависла и прервать выполнение программы.

Блок вывода информационных сообщений в разрабатываемой программе будет отвечать за организацию информирования пользователя о ходе выполнения программы. В частности, планируется отрисовывать в окне консоли постепенно нарастающую полосу символов, а также численное значение от 0% до 100%, отражающие процесс обработки информации. После обработки всех данных будет выведен отчет, содержащий следующие сведения:

- исходный объем данных;
- конечный объем данных;
- затраченное время;
- достигнутая степень компрессии.

При различных ошибках, возникших в ходе выполнения программы (невозможность найти указанные файлы для обработки, неверный формат файлов, невозможность обработки заданного объема информации, неизвестные аргументы командной строки, недопустимые значения переданных параметров) также будут выведены на экран соответствующие сообщения.

### 3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

Первоначально, необходимо определиться с понятиями, которыми будем оперировать при описании алгоритмов, реализуемых разрабатываемыми функциями.

Под *изображением* будем понимать трехмерный массив знаковых или беззнаковых целых чисел-элементов изображения. *Элементы изображения* будем обозначать как  $s_{z,y,x}$ , где  $x$  и  $y$  – пространственные координаты, а  $z$  – номер спектрального слоя. Координаты  $x$ ,  $y$  и  $z$  – это целые беззнаковые числа, значения которых удовлетворяют следующим критериям:  $0 \leq x \leq N_x - 1$ ,  $0 \leq y \leq N_y - 1$  и  $0 \leq z \leq N_z - 1$ , где  $N_x$ ,  $N_y$  и  $N_z$  – размеры изображения по соответствующим координатам.

Под *динамическим диапазоном*  $D$  будем понимать минимальное количество бит, необходимое для кодирования элементов изображения. Для реализуемого алгоритма принятое следующее ограничение:  $2 \leq D \leq 16$ .

Величины  $s_{min}$ ,  $s_{max}$  и  $s_{mid}$  обозначают, соответственно, наименьшее, наибольшее и медианное значение, которое можно представить в данном динамическом диапазоне. Когда значения элементов изображения – беззнаковые целые числа, величины  $s_{min}$ ,  $s_{max}$  и  $s_{mid}$  определяются как:

$$s_{min}=0, s_{max}=2^D-1, s_{mid}=2^{D-1}, \quad (3.1)$$

а целые числа со знаком, то  $s_{min}$ ,  $s_{max}$  и  $s_{mid}$  определяются как:

$$s_{min}=-2^{D-1}, s_{max}=2^{D-1}-1, s_{mid}=0. \quad (3.2)$$

Для упрощения записи формул, в случаях, когда пространственные координаты не имеют значения для производимых вычислений, индекс элемента в трехмерном массиве будем записывать как  $s_z(t) \equiv s_{z,y,x}$ ,  $\delta_z(t) \equiv \delta_{z,y,x}$ , где  $t$  определяется как:

$$t = y * N_x + x. \quad (3.3)$$

Соответственно, значение координаты  $x$  можно рассчитать как:

$$x = t \bmod N_x, \quad (3.4)$$

а значение координаты  $y$  как:

$$y = \lfloor t / N_x \rfloor. \quad (3.5)$$

Количество спектральных слоев, которые учитываются при предсказании значений текущего элемента, будем обозначать как  $P$ .

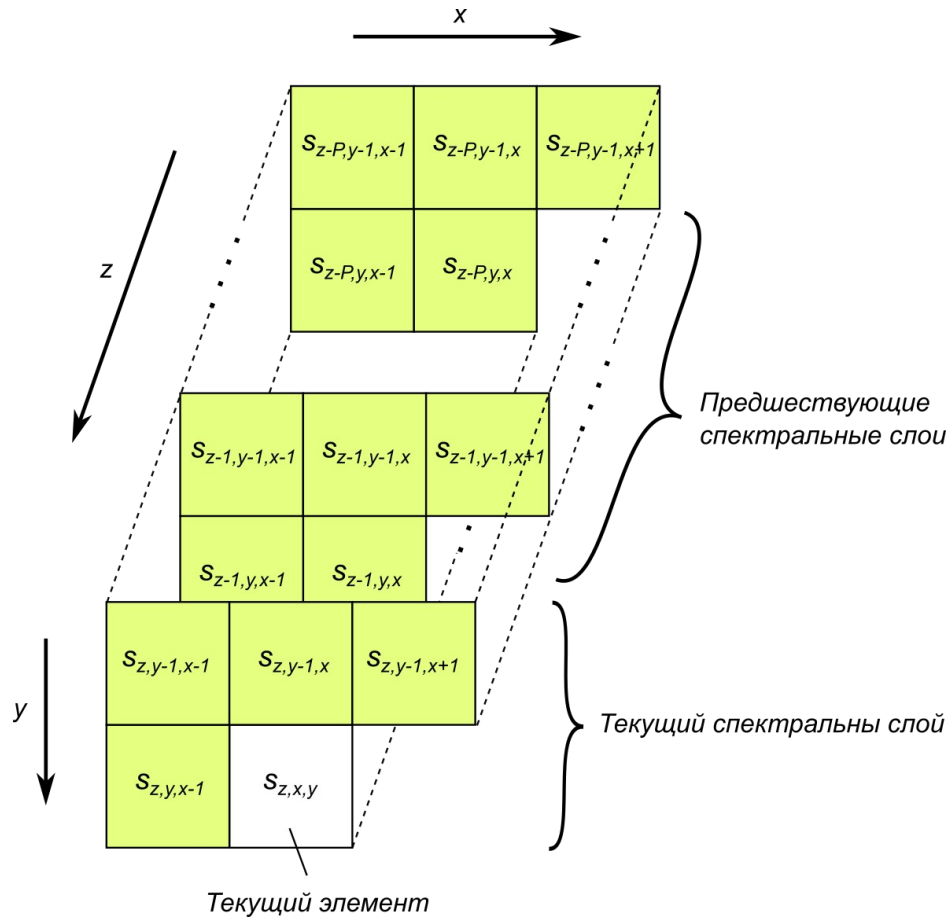


Рисунок 3.1 – Окрестность прогнозирования

Соседними элементами для текущего элемента  $s_{z,y,x}$  будем называть все элементы, находящиеся в окрестности прогнозирования (рисунок 3.1).

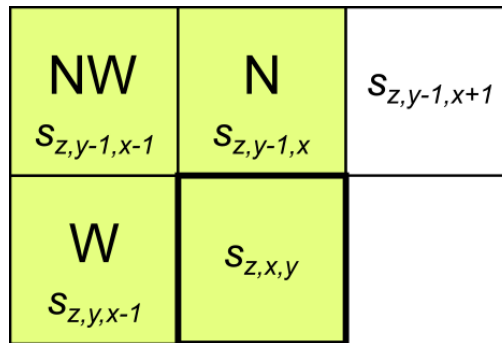


Рисунок 3.2 – N-, W- и NW-соседи элемента  $s_{z,y,x}$

Элемент текущего спектрального слоя с индексом  $s_{z,y-1,x}$  будем называть «северным» соседом и обозначать как  $N$ -сосед, элемент с индексом

$s_{z,y-1,x-1}$  будем называть «северо-западным» соседом и обозначать как *NW-cosced*, а элемент с индексом  $s_{z,y,x-1}$  будем называть «западным» соседом и обозначать как *W-cosced* (рисунок 3.2).

### 3.1 Модуль адаптивного предсказателя

Модуль адаптивного предсказателя производит первичную подготовку гиперспектральных данных для последующего их сжатия энтропийным кодером. С целью облегчения разработки и отладки, данный модуль был разбит на функции, описание которых будет приведено ниже.

Функция `localSum()` выполняет вычисление локальной суммы  $\sigma_{z,y,x}$ , являющейся взвешенной суммой значений предшествующих соседей элемента  $s_{z,y,x}$  в спектре  $z$ . В зависимости от того, какой режим предсказания выбран («по столбцу» или «по соседям»), вычисления, производимые функцией будут различны.

В режиме предсказания «по соседям», локальная сумма  $\sigma_{z,y,x}$  определяется как:

$$\sigma_{z,y,x} = \begin{cases} s_{z,y,x-1} + s_{z,y-1,x-1} + s_{z,y-1,x} + s_{z,y-1,x+1}, & y > 0, 0 < x < N_x - 1 \\ 4 s_{z,y,x-1}, & y = 0, x > 0 \\ 2(s_{z,y-1,x} + s_{z,y-1,x+1}), & y > 0, x = 0 \\ s_{z,y,x-1} + s_{z,y-1,x} + 2 s_{z,y-1,x}, & y > 0, x = N_x - 1, \end{cases} \quad (3.6)$$

а в режиме предсказания «по столбцу»:

$$\sigma_{z,y,x} = \begin{cases} 4 s_{z,y-1,x}, & y > 0 \\ 4 s_{z,y,x-1}, & y = 0, x > 0. \end{cases} \quad (3.7)$$

Функция `d()` вычисляет центральную локальную разницу  $d_{z,y,x}$ , которая определена как:

$$d_{z,y,x} = 4 s_{z,y,x} - \sigma_{z,y,x}. \quad (3.8)$$

Функции `dN()`, `dW()` и `dNW()` вычисляют дирекционные локальные разницы  $d_{z,y,x}^N$ ,  $d_{z,y,x}^W$  и  $d_{z,y,x}^{NW}$  определенные, соответственно как:

$$d_{z,y,x}^N = \begin{cases} 4 s_{z,y-1,x} - \sigma_{z,y,x}, & y > 0 \\ 0, & y = 0, \end{cases} \quad (3.9)$$

$$d_{z,y,x}^W = \begin{cases} 4s_{z,y,x-1} - \sigma_{z,y,x}, & x > 0, y > 0 \\ 4s_{z,y-1,x} - \sigma_{z,y,x}, & x = 0, y > 0 \\ 0, & y = 0, \end{cases} \quad (3.10)$$

$$d_{z,y,x}^{NW} = \begin{cases} 4s_{z,y-1,x-1} - \sigma_{z,y,x}, & x > 0, y > 0 \\ 4s_{z,y-1,x} - \sigma_{z,y,x}, & x = 0, y > 0 \\ 0, & y = 0. \end{cases} \quad (3.11)$$

Функция `getU()` рассчитывает и формирует вектор локальных разниц  $\vec{U}_z(t)$ , необходимый для расчета предсказанного значения  $\hat{s}_z(t)$ . Для «полного» режима предсказания, вектор  $\vec{U}_z(t)$  определяется как:

$$\vec{U}_z(t) = \begin{pmatrix} d_z^N(t) \\ d_z^W(t) \\ d_z^{NW}(t) \\ d_{z-1}(t) \\ d_{z-2}(t) \\ \vdots \\ d_{z-P_z}(t) \end{pmatrix}, \quad (3.12)$$

а для режима «ограниченного» предсказания, вектор  $\vec{U}_z(t)$  определяется, соответственно, как:

$$\vec{U}_z(t) = \begin{pmatrix} d_{z-1}(t) \\ d_{z-2}(t) \\ \vdots \\ d_{z-P_z}(t) \end{pmatrix}. \quad (3.13)$$

Функция `getU()` в качестве входных параметров принимает:

- указатель на трехмерный массив гиперспектральных данных;
- координаты  $x, y, z$  элемента в трехмерном массиве, для которого производится расчет вектора локальных разниц;
- размеры массива  $N_x$  и  $N_y$  по координатам  $x$  и  $y$  соответственно;
- количество предшествующих спектров  $P$ , участвующих в предсказании;

В качестве выходного параметра функция принимает указатель на блок памяти для рассчитываемого вектора  $\vec{U}_z(t)$ .

Функция `weightInitDefault()` производит начальную



инициализацию вектора весов  $\vec{W}_z(t)$  стандартными значениями, который определяется для режима «полного» предсказания как:

$$\vec{W}_z(t) = \begin{pmatrix} \omega_z^N(t) \\ \omega_z^W(t) \\ \omega_z^{NW}(t) \\ \omega_z^{(1)}(t) \\ \omega_z^{(2)}(t) \\ \vdots \\ \omega_z^{(P_z)}(t) \end{pmatrix}, \quad (3.14)$$

а для режима «ограниченного» предсказания как:

$$\vec{W}_z(t) = \begin{pmatrix} \omega_z^{(1)}(t) \\ \omega_z^{(2)}(t) \\ \vdots \\ \omega_z^{(P_z)}(t) \end{pmatrix}, \quad (3.15)$$

где  $\omega_z^N(t)$ ,  $\omega_z^W(t)$ ,  $\omega_z^{NW}(t)$  – весовые коэффициенты соседей текущего элемента трехмерного массива в данном спектральном слое, а  $\omega_z^{(1)}(t)$ ,  $\omega_z^{(2)}(t)$ , ...,  $\omega_z^{(P_z)}(t)$  – весовые коэффициенты предшествующих элементов.

При стандартной инициализации вектора весов  $\vec{W}_z(t)$ , его элементы  $\omega_z^{(1)}(t)$ ,  $\omega_z^{(2)}(t)$ , ...,  $\omega_z^{(P_z)}(t)$  должны иметь следующие значения:

$$\omega_z^{(1)}(1) = \frac{7}{8} \cdot 2^\Omega, \quad \omega_z^{(i)} = \left\lfloor \frac{1}{8} \cdot \omega_z^{(i-1)}(1) \right\rfloor, \quad i = 2, 3, \dots, P_z, \quad (3.16)$$

где  $\Omega$  – параметр, определяющий пределы изменения весовых коэффициентов. Данный параметр задается пользователем и должен удовлетворять условию  $4 \leq \Omega \leq 19$ . Минимально и максимально возможные значения элементов вектора весов  $\omega_{min}$  и  $\omega_{max}$  рассчитываются как:

$$\omega_{min} = -2^{\Omega+2}, \quad \omega_{max} = 2^{\Omega+2} - 1. \quad (3.17)$$

Увеличение числа битов, используемых для представления значений веса обеспечивает увеличение разрешения при расчете прогноза.

Функция `getPredictedScaledVal()` производит вычисление предсказанного масштабированного значения  $\tilde{s}_z(t)$ , определенного как:

$$\tilde{s}_z(t) = \begin{cases} clip\left(\left\lfloor \frac{mod_R[\hat{d}_z(t) + 2^\Omega(\sigma_z(t) - 4s_{mid})]}{2^{\Omega+1}} \right\rfloor + 2s_{mid} + 1, \{2s_{min}, 2s_{max} + 1\}\right), & t > 0 \\ 2s_{z-1}(t), & t = 0, P > 0, z > 0 \\ 2s_{mid}, & t = 0 \wedge (P = 0 \vee z = 0) \end{cases}, \quad (3.18)$$

где  $R$  – размер регистра, определяемый пользователем, должен быть в пределах  $max\{32, D + \Omega + 2\} \leq R \leq 64$ ;

$\hat{d}_z(t)$  – предсказанная центральная локальная разница, определенная как скалярное произведение векторов  $\vec{U}_z(t)$  и  $\vec{W}_z(t)$ :

$$\hat{d}_z(t) = \vec{U}_z(t) \vec{W}_z(t). \quad (3.19)$$

Функция  $clip(x, \{x_{min}, x_{max}\})$  производит «усечение» значения  $x$  по границам  $x_{min}$  и  $x_{max}$ . Функция  $mod_R[x]$  определена как:

$$mod_R[x] = ((x + 2^{R-1}) mod 2^R) - 2^{R-1}. \quad (3.20)$$

Предсказанное значение  $\hat{s}_z(t)$  вычисляется, соответственно, как:

$$\hat{s}_z(t) = \left\lfloor \tilde{s}_z \frac{(t)}{2} \right\rfloor. \quad (3.21)$$

Функция `updateWeight()` производит пересчет вектора весов  $\vec{W}_z(t)$  после вычисления очередного предсказанного значения. Значение вектора  $\vec{W}_z(t+1)$  вычисляется по следующей формуле:

$$\vec{W}_z(t+1) = clip\left(\vec{W}_z(t) + \left\lfloor \frac{1}{2} (sgn^+[e_z(t)] 2^{-\rho(t)} \vec{U}_z(t) + 1) \right\rfloor, \{\omega_{min}, \omega_{max}\}\right), \quad (3.22)$$

где  $e_z(t)$  – ошибка предсказания, определенная как:

$$e_z(t) = 2s_z(t) - \tilde{s}_z(t), \quad (3.23)$$

$\rho(t)$  – масштабирующая экспонента, определяющая скорость изменения значений элементов вектора весов  $\vec{W}_z(t)$ .

В разрабатываемой программе функция `getScalingExp()` выполняет расчет масштабирующей экспоненты. Меньшее значение  $\rho(t)$  порождает более резкое изменение компонентов вектора  $\vec{W}_z(t)$ , и наоборот. Масштабирующая экспонента  $\rho(t)$  вычисляется как:

$$\rho(t) = clip\left(v_{min} + \left\lfloor \frac{t - N_x}{t_{inc}} \right\rfloor, \{v_{min}, v_{max}\}\right) + D + \Omega, \quad (3.24)$$

где  $v_{min}$ ,  $v_{max}$ ,  $t_{inc}$  – параметры, определяемые пользователем.

Функция `getMappedPredictionResidual()` выполняет отображение остатка предсказания (в общем случае значения со знаком) на область значений без знака. Отраженный остаток предсказания  $\delta_z(t)$  определен как:

$$\delta_z(t) = \begin{cases} |\Delta_z(t)| + \theta_z(t), & |\Delta_z(t)| + \theta_z(t) \\ 2|\Delta_z(t)|, & 0 \leq (-1)^{\tilde{s}_z(t)} \Delta_z(t) \leq \theta_z(t), \\ 2|\Delta_z(t)| - 1, & \text{в остальных случаях} \end{cases} \quad (3.25)$$

где  $\Delta_z(t)$  – остаток предсказания, определяемый как разность между истинным значением элемента и его предсказанным значением:

$$\Delta_z(t) = s_z(t) - \hat{s}_z(t), \quad (3.26)$$

а  $\theta_z(t)$  определяется как:

$$\theta_z(t) = \min\{\hat{s}_z(t) - s_{min}, s_{max} - \hat{s}_z(t)\}. \quad (3.27)$$

Функция `getRestoredValue()` выполняет действие, обратное функции `getMappedPredictionResidual()`, а именно, восстанавливает оригинальное значение элемента исходного изображения  $s_z(t)$  из известных предсказанного значения  $\hat{s}_z(t)$  и транслированного остатка предсказания  $\delta_z(t)$ . Остаток предсказания  $\Delta_z(t)$  в этом случае можно найти из выражения (3.23):

$$\Delta_z(t) = \begin{cases} (\theta_z(t) - \delta_z(t)) sgn^+(\hat{s}_z(t) - s_{mid}), & \delta_z(t) > 2\theta_z(t) \\ \left\lfloor \frac{\delta_z(t) + 1}{2} \right\rfloor (-1)^{\tilde{s}_z(t) + \delta_z(t)}, & \delta_z(t) \leq 2\theta_z(t) \end{cases}, \quad (3.28)$$

где  $\theta_z(t)$  вычисляется по формуле (3.22).

Оригинальное значение элемента исходного изображения  $s_z(t)$  из можно вычислить по формуле (3.24):

$$s_z(t) = \Delta_z(t) + \hat{s}_z(t). \quad (3.29)$$

### 3.2 Модуль энтропийного кодера

Отображенные остатки предсказания  $\delta_z(t)$ , согласно рекомендациям стандарта CCSDS [4], должны быть закодированы либо элементарно-адаптивным, либо блочно-адаптивным энтропийным кодером.

Элементарно-адаптивный энтропийный кодер производит кодирование каждого отображенного остатка предсказания  $\delta_z(t)$  в бинарное слово переменной длины. Выбор подходящего кодового слова производится согласно адаптивной статистики, которая состоит из двух ключевых элементов – аккумулятора  $\Sigma_z(t)$  и счетчика  $\Gamma(t)$ , значения которых непрерывно обновляются в процессе кодирования. Отношение  $\Sigma_z(t)/\Gamma(t)$  предоставляет оценку значения среднего отображенного остатка предсказания в спектральном канале. На основе данной оценки и выбирается длина кодового слова.

Начальное значение счетчика  $\Gamma(1)$  должно быть равно:

$$\Gamma(1) = 2^{\gamma_0}, \quad (3.30)$$

где  $\gamma_0$  – определяемое пользователем начальное значение экспоненты счетчика, должно удовлетворять условию  $1 \leq \gamma_0 \leq 8$ .

Для каждого спектрального канала начальное значение аккумулятора  $\Sigma_z(1)$  должно быть равно:

$$\Sigma_z(1) = \left\lfloor \frac{1}{2^7} (3 * 2^{k'_z + 6} - 49) \Gamma(1) \right\rfloor, \quad (3.31)$$

где  $k'_z$  – параметр, определяемый пользователем, должен удовлетворять условию  $0 \leq k'_z \leq D - 2$ . Также, может быть определена константа инициализации аккумулятора  $K$  (таблица 3.3). В таком случае,  $k'_z = K$  для всех  $z$ . Данное уравнение гарантирует, что начальное значение длины кодового слова  $k_z(t)$ , рассчитанное для каждого спектрального канала  $z$ , будет равно  $k'_z$ .

При  $t > 0$ , значение аккумулятора для спектрального канала  $z$  определяется по формуле 3.32, а значение счетчика по формуле 3.33.

$$\Sigma_z(t) = \begin{cases} \Sigma_z(t-1) + \delta_z(t-1), & \Gamma(t-1) < 2^{\gamma'-1} \\ \left\lfloor \frac{\Sigma_z(t-1) + \delta_z(t-1) + 1}{2} \right\rfloor, & \Gamma(t-1) = 2^{\gamma'-1} \end{cases}, \quad (3.32)$$

$$\Gamma(t) = \begin{cases} \Gamma(t-1) + 1, & \Gamma(t-1) < 2^{\gamma'-1} \\ \left\lfloor \frac{\Gamma(t-1) + 1}{2} \right\rfloor, & \Gamma(t-1) = 2^{\gamma'-1} \end{cases}. \quad (3.33)$$

Интервалы изменения счетчика  $\Gamma(t)$  и аккумулятора  $\Sigma_z(t)$  контролируются определяемым пользователем значением параметра  $\gamma'$ , которое должно быть целым из интервала  $\max\{4, \gamma_0 + 1\} \leq \gamma' \leq 9$ .

Функции `getAccum()` и `getCounter()` вычисляют и возвращают значения, соответственно, аккумулятора и счетчика для элемента  $t$  текущего спектра.

Пользователь также определяет значение унарного предела длины  $U_{max}$ , который должен удовлетворять условию  $8 \leq U_{max} \leq 32$ . Процедура элементарно-адаптивного энтропийного кодирования гарантирует, что длина кодового слова для отображенного остатка  $\delta_z(t)$  занимает не более  $U_{max} + D$  бит.

Первое значение отображенного остатка в каждом спектральном канале  $z$  должно быть некодированным, то есть, кодовое слово для  $\delta_z(0)$  – это само число  $\delta_z(0)$ , представленное в виде  $D$ -битного беззнакового целого.

Функция `getCodeWordSize()` производит вычисление оптимальной длины кодового слова  $k_z(t)$ . В качестве аргументов функция принимает текущие значения аккумулятора  $\Sigma_z(t)$  и счетчика  $\Gamma(t)$ .

Для  $t > 0$ , кодовое слово для отображенного остатка предсказания  $\delta_z(t)$  зависит от значений  $k_z(t)$  и  $u_z(t)$ , где  $k_z(t) = 0$ , если:

$$2\Gamma(t) > \Sigma_z(t) + \left\lfloor \frac{49}{2^7} \Gamma(t) \right\rfloor, \quad (3.34)$$

иначе  $k_z(t)$  – наибольшее положительное целое, такое, что

$$\Gamma(t) 2^{k_z(t)} \leq \Sigma_z(t) + \left\lfloor \frac{49}{2^7} \Gamma(t) \right\rfloor, \quad (3.35)$$

а значение  $u_z(t)$  рассчитывается как:

$$u_z(t) = \left\lfloor \frac{\delta_z(t)}{2^{k_z(t)}} \right\rfloor. \quad (3.36)$$

Для элементов с индексом  $t > 0$ , кодовые слова для  $\delta_z(t)$  выбираются исходя из следующих двух правил:

1 Если  $u_z(t) < U_{max}$ , кодовое слово для  $\delta_z(t)$  должно состоять из  $u_z(t)$  нулей, за которыми следует одна единица, а далее – младшие значащие биты двоичного представления  $\delta_z(t)$ .

2 Иначе кодовое слово для  $\delta_z(t)$  должно состоять из  $U_{max}$  нулей, за которыми следует  $D$ -битное двоичное представление числа  $\delta_z(t)$ .

Последнее кодовое слово в сжатом изображении дополняется таким количеством нулей, чтобы итоговый размер сжатого изображения был кратен размеру входного слова.

Функция `encodeGolomb()` выполняет энтропийное кодирование всего массива отображенных остатков предсказаний, полученных от адаптивного предсказателя, формируя, тем самым, основную часть сжатого гиперспектрального изображения. В качестве входных параметров функция принимает:

- указатель на массив отображенных остатков предсказаний;
- указатель на массив выходных кодированных данных;
- указатель на структуру `ImageMetadata`;
- указатель на структуру `PredictorMetadata`;
- указатель на структуру `EncoderMetadata`.

Результатом работы функции `encodeImage()` является массив кодированных данных, а также заполненная структура `EncoderMetadata`, необходимая для восстановления исходного изображения.

### 3.3 Модуль сохранения сжатого изображения

При сохранении сжатого изображения необходимо сформировать его структуру, состоящую из заголовка и, собственно, сжатых данных. Для этой цели служит функция `saveCompressedImage()`, принимающая в качестве параметров:

- `const char * fileName` – имя файла;
- `const void * data` – указатель на массив сжатых данных;
- `const size_t dataSize` – размер сжатых данных в байтах;
- `ImageMetadata * imageMeta` – метаданные изображения;
- `PredictorMetadata * predMeta` – метаданные предсказателя;
- `EncoderMetadata * encoderMeta` – метаданные энтропийного кодера.

В случае успешного сохранения файла функция

`saveCompressedImage()` возвращает нулевое значение и ненулевое – в случае ошибки.

Из перечисленных выше структур метаданных функция `saveCompressedImage()` формирует заголовок сжатого файла. Ниже будет приведено детальное описание данных структур с описанием всех входящих в них полей и их размеров.

Структура `PredictorMetadata` содержит в себе параметры адаптивного предсказателя, необходимые для восстановления исходного изображения. Описание данной структуры представлено в виде таблицы (таблица 3.1).

Таблица 3.1 – Описание полей структуры `PredictorMetadata`

Поле	Ширина (бит)	Описание
1	2	3
Зарезервировано	2	Всегда должно иметь значение «00».
Число спектральных слоев для предсказания	4	Значение параметра $P$ , 4-битовое беззнаковое целое.
Режим предсказания	1	«0» – режим «полного» предсказания; «1» – режим «ограниченного» предсказания.
Зарезервировано	1	Всегда должно иметь значение «0».
Режим вычисления локальной суммы	1	«0» – вычисление «по соседям»; «1» – вычисление «по столбцу».
Зарезервировано	1	Всегда должно иметь значение «0».
Размер регистра	6	Значение параметра $R$ , 6-битовое беззнаковое целое.
Разрешение элементов вектора весов	4	Значение $\Omega$ , 4-битовое беззнаковое целое.
Шаг изменения масштабирующей экспоненты	4	Значение $(\log_2 t_{inc} - 4)$ , 4-битовое беззнаковое целое.
Начальное значение масштабирующей экспоненты	4	Значение $(v_{min} + 6)$ , 4-битовое беззнаковое целое.
Конечное значение масштабирующей экспоненты	4	Значение $(v_{max} + 6)$ , 4-битовое беззнаковое целое.
Зарезервировано	1	Всегда должно иметь значение «0».

Продолжение таблицы 3.1

1	2	3
Метод инициализации вектора весов	1	«0» – стандартные значения вектора; «1» – пользовательские значения вектора.
Флаг таблицы инициализации вектора весов	1	«0» – таблица инициализации не включена в данную структуру; «1» – таблица включена в данную структуру.
Разрешение вектора инициализации	5	Когда выбран стандартный режим инициализации вектора весов, должно иметь значение «00000», иначе должно содержать значение $Q$ , 5-битовое беззнаковое целое.
Таблица инициализации вектора весов	перем. длины	

Структура ImageMetadata содержит сведения об исходном гиперспектральном изображении. Описание данной структуры приведено в таблице 3.2.

Таблица 3.2 – Описание полей структуры ImageMetadata

Поле	Ширина (бит)	Описание
1	2	3
Данные определяемые пользователем	8	Пользователь может назначить значение этого поля произвольно, например, для указания значения определяемого пользователем индекса изображения в последовательности изображений, либо задать некоторую кодовую последовательность, которая будет идентифицировать файл как сжатый.
Размер по координате X	16	Значение $N_x$ , определенное как 16-битное беззнаковое целое.
Размер по координате Y	16	Значение $N_y$ , определенное как 16-битное беззнаковое целое.
Размер по координате Z	16	Значение $N_z$ , определенное как 16-битное беззнаковое целое.



Продолжение таблицы 3.2

Тип данных	1	«0» – элементы изображения являются беззнаковыми целыми; «1» – элементы изображения являются знаковыми целыми.
Зарезервировано	2	Всегда должно иметь значение «00»
Динамический диапазон	4	Значение параметра $D$ , представленное как 4-битное беззнаковое целое
Способ организации данных	1	«0» – каналы, разделенные по строкам (BIL), «1» – поканальная запись (BSQ).
Глубина разделения каналов	16	Когда выбран BIL порядок, данное поле должно содержать значение $M$ , определяемое как 16-битное беззнаковое целое. Когда выбран BSQ порядок, все биты данного поля должны иметь значение «0»
Зарезервировано	2	Всегда должно иметь значение «00»
Размер выходного слова	3	Параметр $B$ , определяемый как 3-битное беззнаковое целое.
Тип энтропийного кодера	1	«0» – sample-adaptive; «1» – block-adaptive.
Зарезервировано	10	Все биты данного поля должны иметь значение «0».

Структура EncoderMetadata содержит параметры работы энтропийного кодера. В зависимости от типа энтропийного кодера (посимвольного или блочного), может быть два типа структур. В таблице 3.3 представлено описание структуры EncoderMetadata для посимвольного энтропийного кодера.

Таблица 3.3 – Структура EncoderMetadata (посимвольный кодер)

Поле	Ширина (бит)	Описание
1	2	3
Предел унарной длины	5	Значение параметра $U_{max}$ , 5-битное беззнаковое целое.
Размерность счетчика масштабирования	3	Значение параметра $(\gamma' - 4)$ , 3-битное беззнаковое целое.

Продолжение таблицы 3.3

1	2	3
Начальное значение счетчика экспоненты	3	Значение параметра $\gamma_0$ , 3-битное беззнаковое целое.
Константа инициализации аккумулятора	4	Когда определена константа инициализации аккумулятора $K$ , данное поле содержит значение $K$ как 4-битное беззнаковое целое. Иначе поле должно содержать все свои биты в «1».
Флаг табличной инициализации аккумулятора	1	«0» – таблица инициализации аккумулятора не включена в состав структуры; «1» – таблица включена в состав структуры.
Таблица инициализации аккумулятора (опционально)	перем. длины	

В таблице 3.4 приведено описание полей структуры EncoderMetadata при блочном типе энтропийного кодера.

Таблица 3.4 – Структура EncoderMetadata (блочный кодер)

Поле	Ширина (бит)	Описание
Зарезервировано	1	Должно всегда иметь значение «0».
Размер блока	2	«00» – размер блока $J = 8$ ; «01» – размер блока $J = 16$ ; «10» – размер блока $J = 32$ ; «11» – размер блока $J = 64$ .
Флаг режима ограниченного кодирования	1	Данное поле должно иметь значение «1» при $D \leq 4$ и использовании ограниченного набора параметров кодирования, «0» – в остальных случаях.
Интервал эталонного образца	12	Значение параметра $r$ , 12-битное беззнаковое целое.

### 3.4 Модуль загрузки сжатого изображения

Модуль загрузки сжатого изображения представлен функцией `loadCompressedImage()`, которая имеет следующие параметры:

- char \* fileName – имя сжатого файла;
- void \* data[] – адрес указателя, по которому будут доступны кодированные данные;
- size\_t \* dataSize – указатель на переменную для сохранения размера кодированных данных в байтах;
- struct ImageMetadata \* imageMeta – указатель на структуру, в которую будут загружены данные об исходном изображении;
- struct PredictorMetadata \* predMeta – указатель на структуру, в которую будут загружены параметры работы адаптивного предсказателя;
- struct EncoderMetadata \* encoderMeta – указатель на структур, в которую будут загружены параметры работы энтропийного кодера.

При успешной загрузке файла сжатого изображения, функция возвращает нулевое значение, а при возникновении ошибок – не нулевое.

### 3.5 Модуль загрузки исходного изображения

Модуль загрузки исходного изображения представлен функцией `loadFromPGM()` – которая имеет следующие параметры:

- char \* fileName – имя загружаемого файла;
- uint16\_t \*data[] – адрес указателя, по которому будет находиться «тело» загруженного изображения;
- unsigned \* sizeX – указатель на переменную для сохранения ширина изображения;
- unsigned \* sizeY – указатель на переменную для сохранения высоты изображения;
- unsigned \* maxValue – указатель на переменную для сохранения максимально возможного значения элемента изображения.

При успешной загрузке файла сжатого изображения, функция возвращает нулевое значение, а при возникновении ошибок – не нулевое.

### 3.6 Модуль сохранения исходного изображений

Модуль сохранения исходного изображения представлен функцией `saveToPGM()`, имеющей следующие параметры:

- uint16\_t data[] – указатель на сохраняемые данные;
- unsigned sizeX – ширина изображения;
- unsigned sizeY – высота изображения;
- unsigned maxValue – максимально значение элемента изображения.

## 4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

При разработке программных модулей частично была применена парадигма «разработки через тестирование» (test-driven development, TDD), которая заключалась в том, что сначала для некоторой функции писался тест, а потом уже сам код функции. Для написания модульных тестов был применен фреймворк Catch2, который распространяется в виде одного-единственного заголовочного файла catch.hpp, в котором в виде макросов описаны тестирующие конструкции. Наборы входных и выходных данных генерировались с помощью табличных процессоров.

Рассмотрим ключевые алгоритмы, реализованные в разрабатываемом программном модуле.

### 4.1 Алгоритм функции вычисления локальной суммы

Как было сказано ранее, под локальной суммой  $\sigma_{z,y,x}$  понимается сумма значений соседей некоторого пикселя  $s_{z,y,x}$  изображения. В качестве исходных данных имеем:

- uint16\_t \* band – указатель на спектральный слой;
- uint32\_t sizeY – размер спектрального слоя по оси Y;
- uint32\_t sizeX – размер спектрального слоя по оси X;
- uint32\_t y, x – координаты целевого пикселя в спектральном слое.

Возвращаемое значение имеет тип int. Для данной функции был написан следующий тест:

```
TEST_CASE( "getLocalSum", "[local_sum]" ) {
    uint16_t ms[] = {
        7, 2, 6, 7, 5, 1, 8,
        5, 4, 9, 9, 9, 6, 3,
        5, 2, 6, 5, 6, 1, 7,
        2, 8, 8, 8, 7, 6, 5,
        8, 1, 9, 7, 3, 9, 9
    };
    int sizeX = 7;
    int sizeY = 5;
    REQUIRE ( getLocalSum(ms, sizeY, sizeX, 0, 1) == 28 );
    REQUIRE ( getLocalSum(ms, sizeY, sizeX, 1, 0) == 18 );
    REQUIRE ( getLocalSum(ms, sizeY, sizeX, 1, 1) == 20 );
    REQUIRE ( getLocalSum(ms, sizeY, sizeX, 0, 6) == 4 );
    REQUIRE ( getLocalSum(ms, sizeY, sizeX, 2, 3) == 33 );
    REQUIRE ( getLocalSum(ms, sizeY, sizeX, 4, 6) == 25 );
    REQUIRE ( getLocalSum(ms, sizeY, sizeX, 4, 0) == 20 );
    REQUIRE ( getLocalSum(ms, sizeY, sizeX, 2, 6) == 13 );
}
```

Опишем данный алгоритм по шагам.

Шаг 1. Создадим переменные  $s_{NW}$ ,  $s_N$ ,  $s_{NE}$ ,  $s_W$  типа `int`, в которых будем хранить, соответственно, значение «северо-западного» (northwest), «северного» (north), «северо-восточного» (northeast) и «западного» (west) соседей. Также, создадим переменную  $sum$ , в которой будет храниться локальная сумма.

Шаг 2. Произведем проверку корректности входных данных: если указатель  $band$  равен нулю, либо если значение  $x$  больше или равно значению  $sizeX$ , либо если значение  $y$  больше или равно значению  $sizeY$ , либо  $y$  и  $x$  одновременно равны нулю, тогда присваиваем переменной  $sum$  значение  $-1$  и переходим к шагу 8. Иначе к шагу 4

Шаг 3. Если  $y$  равен нулю, тогда запишем в переменную  $s_W$  значение «западного» соседа, затем присвоим переменной  $sum$  значение  $s_W * 4$  и переходим к шагу 8. Иначе к шагу 5.

Шаг 5. Если значение переменной  $x$  больше нуля, тогда запишем в переменные  $s_{NW}$ ,  $s_N$ ,  $s_{NE}$ ,  $s_W$  значения соответствующих соседей, присвоим переменной  $sum$  значение суммы переменных  $s_{NW}$ ,  $s_N$ ,  $s_{NE}$ ,  $s_W$  и перейдем к шагу 8. Иначе к шагу 6.

Шаг 6. Если  $x$  равен нулю, тогда присвоим переменным  $s_N$  и  $s_{NE}$  значения соответствующих соседей, присвоим переменной  $sum$  удвоенное значение суммы переменных  $s_N$  и  $s_{NE}$  и перейдем к шагу 8. Иначе к шагу 7.

Шаг 7. Если значение  $x$  равно значению  $sizeX - 1$ , тогда присвоим переменным  $s_{NW}$ ,  $s_N$ ,  $s_W$  значения соответствующих соседей, присвоим переменной  $sum$  сумму  $s_W$ ,  $s_{NW}$  и удвоенного значения  $s_N$ .

Шаг 8. Выход из функции с возвратом значения переменной  $sum$ . Отрицательное значение переменной  $sum$  будет сигнализировать об ошибке входных данных.

## 4.2 Алгоритм функции вычисления вектора локальных разниц

Вектор локальных разниц  $\vec{U}_z(t)$  для некоторого пикселя гиперспектрального изображения  $s_{z,y,x}$  представляет собой массив, состоящий из результатов вычитания рассчитанной локальной суммы  $\sigma_{z,y,x}$  из каждого из соседних элементов, участвующих в предсказании значения данного пикселя. Размер вектора  $\vec{U}_z(t)$  определяется задаваемым пользователем параметром  $P$ , который определяет количество предшествующих спектральных слоев, участвующих в предсказании.

Исходные данные для алгоритма следующие:

- `int * U` – указатель на память, выделенную под вычисляемый вектор  $\vec{U}_z(t)$ ;
- `int P` – количество спектральных слоев, участвующих в

предсказании;

- `uint16_t * image` – указатель на область памяти, хранящую «тело» гиперспектрального изображения;
- `int sizeY` – высота изображения;
- `int sizeX` – ширина изображения;
- `int z, y, x` – координаты пикселя, для которого происходит вычисление вектора локальных разниц.

Приведем описание данного алгоритма по шагам.

Шаг 1. Объявим переменную `bandSize` типа `size_t`, в которой будем хранить полный размер одного спектрального слоя изображения, а также указатель `band` типа `uint16_t *`, в котором будем хранить адрес на область памяти, в которой хранится спектральный слой с индексом `z`, переменную `localSum` типа `int` для хранения локальной суммы.

Шаг 2. Вычислим размеры спектрального слоя и присвоим их переменной `bandSize`.

Шаг 3. Вычислим адрес спектрального слоя с индексом `z` и присвоим его переменной `band`.

Шаг 4. Вычислим значение локальной суммы для пикселя с координатами `z, y, x` и присвоим его переменной `localSum`.

Шаг 5. Элементу массива `U[0]` присвоим значение разницы между учетверенным значением «северного» соседа и значением переменной `localSum`.

Шаг 6. Элементу массива `U[1]` присвоим значение разницы между учетверенным значением «западного» соседа и значением переменной `localSum`.

Шаг 7. Элементу массива `U[2]` присвоим значение разницы между учетверенным значением «северо-западного» соседа и значением переменной `localSum`.

Шаг 8. Если рассчитывается вектор  $\vec{U}_z(t)$  для самого первого спектрального слоя, то есть, если `z` равно нулю, тогда заполним оставшиеся `P` элементов массива `U` нулевыми значениями и переходим к шагу 14.

Шаг 9. Объявим переменные `n, i` типа `int`. Присвоим `i` значение 1.

Шаг 10. Если `i` больше `P`, тогда переходим к шагу 14.

Шаг 11. Если переменная `i` больше `z`, тогда присвоить переменной `n` значение `i`, иначе присвоить значение `z`.

Шаг 12. Присвоить элементу массива `U[i+2]` значение разницы между элементом предшествующего спектрального слоя с координатами `z-n, y, x` и значением переменной `localSum`.

Шаг 13. Увеличить значение переменной `i` и перейти к шагу 10;

Шаг 14. Конец алгоритма.

### 4.3 Алгоритм функции инициализации вектора весов

Вектор весов  $\vec{W}_z(t)$  хранит в себе набор весовых коэффициентов, в соответствии с которыми при предсказании значения некоторого элемента  $s_{z,y,x}$  принимаются во внимание значения элементов-соседей.

В качестве входных данных примем:

- int \* W – указатель на массив весовых коэффициентов;
- int om – значение параметра  $\Omega$ , определяющего пределы изменения весовых коэффициентов;
- int P – количество спектральных слоев, участвующих в предсказании;

Для данной функции был подготовлен следующий тест:

```
TEST_CASE("weightInitDefault", "[weightInitDefault]") {
    int W[10];

    int W_1[6] = {0, 0, 0, 14, 1, 0};
    weightInitDefault(W, 4, 3);

    for(int i = 0; i < 6; i++) {
        INFO("Index: " << i);
        CHECK(W[i] == W_1[i]);
    }

    int W_2[6] = {0, 0, 0, 224, 28, 3};
    weightInitDefault(W, 8, 3);

    for(int i = 0; i < 6; i++) {
        INFO("Index: " << i);
        CHECK(W[i] == W_2[i]);
    }
}
```

Приведем описание данного алгоритма по шагам.

Шаг 1. Присвоить первым трем элементам массива W нулевые значения.

Шаг 2. Если значение параметра P равно нулю, тогда перейти к шагу 8.

Шаг 3. Присвоить элементу массива W[3] начальное значение, согласно формуле (3.14).

Шаг 4. Объявить переменную i и присвоить ей единичное значение.

Шаг 5. Если значение i больше или равно P, тогда перейти к шагу 8.

Шаг 6. Присвоить элементу массива W[i+3] значение предшествующего элемента, уменьшенное в восемь раз.

Шаг 7. Увеличить значение переменной i и перейти к шагу 5.

Шаг 8. Конец алгоритма.

#### 4.4 Алгоритм функции вычисления отображенного остатка предсказания

Отображенный остаток предсказания  $\delta_z(t)$  – это разность между истинным значением элемента  $s_{z,y,x}$  и его предсказанным значением  $\hat{s}_{z,y,x}$ , отображенная на область беззнаковых чисел.

Входные данные для функции следующие:

- int s – истинное значение элемента  $s_{z,y,x}$ ;
- int scale\_s\_pred – удвоенное предсказанное значение  $\tilde{s}_{z,y,x}$ ;
- int s\_min – минимально возможное значение элемента  $s_{z,y,x}$ ;
- int s\_max – максимально возможное значение элемента  $s_{z,y,x}$ .

Возвращаемое значение имеет тип int. Модульный тест для данной функции выглядит следующим образом:

```
TEST_CASE("getMappedPredictionResidual") {  
    CHECK(getMappedPredictionResidual( 9, 8, 0, 63) == 9);  
    CHECK(getMappedPredictionResidual( 9, 17, 0, 63) == 1);  
    CHECK(getMappedPredictionResidual( 9, 22, 0, 63) == 3);  
    CHECK(getMappedPredictionResidual( 8, 3, 0, 63) == 8);  
    CHECK(getMappedPredictionResidual( 4, 0, 0, 63) == 4);  
    CHECK(getMappedPredictionResidual(13, 5, 0, 63) == 13);  
    CHECK(getMappedPredictionResidual( 8, 31, 0, 63) == 14);  
    CHECK(getMappedPredictionResidual( 7, 10, 0, 63) == 4);  
    CHECK(getMappedPredictionResidual(11, 40, 0, 63) == 17);  
}
```

Рассмотрим алгоритм нахождения величины  $\delta_z(t)$  по шагам.

Шаг 1. Вычислить предсказанное значение элемента путем деления scale\_s\_pred на два и присвоить результат новой переменной s\_pred типа int.

Шаг 2. Вычислить остаток предсказания как разность между истинным значением элемента s и его предсказанным значением s\_pred и записать результат в переменную residual типа int.

Шаг 3. Записать абсолютное значение переменной residual в переменную abs\_residual типа int.

Шаг 4. Вычислить разность между предсказанным значением s\_pred и минимальным значением s\_min, записать результат в переменную teta\_a типа int.

Шаг 5. Вычислить разность между максимальным значением s\_max и предсказанным значением s\_pred, записать результат в переменную teta\_b.

Шаг 6. В переменную teta типа int записать меньшее из значений переменных teta\_a и teta\_b.



Шаг 7. В переменную `k` типа `int` записать значение переменной `residual`, если значение переменной `scale_s_pred` четное. Иначе записать в `k` отрицательное значение переменной `residual`.

Шаг 8. Объявить переменную `result` типа `uint16_t`.

Шаг 9. Если значение переменной `abs_residual` больше значения переменной `teta`, присвоить переменной `result` результат суммы значений переменных `abs_residual` и `teta` и перейти к шагу 12.

Шаг 10. Если значение переменной `k` больше или равно нулю, а также если оно меньше или равно значению `teta`, присвоить переменной `result` удвоенное значение переменной `abs_residual` и перейти к шагу 12.

Шаг 11. Присвоить переменной `result` удвоенное значение переменной `abs_residual` и отнять от полученного значения единицу.

Шаг 12. Конец алгоритма. В переменной `result` хранится искомое значение отраженного остатка предсказания.

#### 4.5 Алгоритм функции восстановления исходного значения элемента

Данная функция позволяет найти истинное значение элемента изображения  $s_{z,y,x}$  по известным его предсказанному значению  $\hat{s}_{z,y,x}$  и отображенному остатку предсказания  $\delta_z(t)$ .

Входные данные для работы алгоритма следующие:

- `int mapped_residual` – отображенный остаток предсказания  $\delta_z(t)$ ;
- `int scale_s_pred` – удвоенное предсказанное значение  $\tilde{s}_{z,y,x}$ ;
- `int s_min` – минимально возможное значение элемента  $s_{z,y,x}$ ;
- `int s_max` – максимально возможное значение элемента  $s_{z,y,x}$ ;
- `int s_mid` – середина интервала возможных значений элемента  $s_{z,y,x}$ .

Возвращаемое значение имеет тип `int`. Для данной функции был написан следующий модульный тест:

```
TEST_CASE("getRestoredValue") {
    CHECK(getRestoredValue( 9,  8, 0, 63, 32) == 9);
    CHECK(getRestoredValue( 1, 17, 0, 63, 32) == 9);
    CHECK(getRestoredValue( 3, 22, 0, 63, 32) == 9);
    CHECK(getRestoredValue( 4,  0, 0, 63, 32) == 4);
    CHECK(getRestoredValue( 5,  5, 0, 63, 32) == 5);
    CHECK(getRestoredValue( 4, 10, 0, 63, 32) == 7);
    CHECK(getRestoredValue(17, 40, 0, 63, 32) == 11);
    CHECK(getRestoredValue(13,  5, 0, 63, 32) == 13);
    CHECK(getRestoredValue(14, 31, 0, 63, 32) == 8);
}
```

Опишем алгоритм функции по шагам.

Шаг 1. Вычислить предсказанное значение элемента путем деления  $scale\_s\_pred$  на два и присвоить результат новой переменной  $s\_pred$  типа `int`.

Шаг 2. Вычислить разность между предсказанным значением  $s\_pred$  и минимальным значением  $s\_min$ , записать результат в переменную  $teta\_a$  типа `int`.

Шаг 3. Вычислить разность между максимальным значением  $s\_max$  и предсказанным значением  $s\_pred$ , записать результат в переменную  $teta\_b$ .

Шаг 4. В переменную  $teta$  типа `int` записать меньшее из значений переменных  $teta\_a$  и  $teta\_b$ .

Шаг 5. В переменную  $sign$  типа `int` записать минус единицу, если результат вычитания значения переменной  $s\_mid$  из значения переменной  $s\_pred$  отрицательный, иначе записать единицу.

Шаг 6. В переменную  $k$  типа `int` записать единицу, если сумма значений переменных  $scale\_s\_pred$  и  $mapped\_residual$  четная, иначе записать минус единицу.

Шаг 7. Объявить переменные  $residual$  и  $result$  типа `int`;

Шаг 8. Если значение переменной  $mapped\_residual$  больше удвоенного значения переменной  $teta$ , присвоить переменной  $residual$  результат вычитания из  $teta$  значения  $mapped\_residual$ , умноженного на  $sign$  и перейти к шагу 10.

Шаг 9. Записать в переменную  $residual$  результат деления на два увеличенного на единицу значения переменной  $mapped\_residual$  и умножить его на значение переменной  $k$ .

Шаг 10. Присвоить переменной  $result$  результат суммы значений переменных  $residual$  и  $s\_pred$ .

Шаг 11. Конец алгоритма. В переменной  $result$  хранится искомое восстановленное значение  $s_{z,y,x}$ .

#### 4.6 Алгоритм работы адаптивного предсказателя

Адаптивный предсказатель может выполнять два взаимно обратных действия:

- опираясь на данные исходного гиперспектрального изображения, генерировать трехмерный массив отображенных в беззнаковую область остатков предсказания;
- восстанавливать из массива отображенных остатков предсказания исходное изображение.

Схема алгоритма работы адаптивного предсказателя приведена на

чертеже ГУИР.400201.013 ПД.1. Приведем, также описание работы данного алгоритма по шагам.

В качестве входных данных имеем следующее:

- `uint16_t * in` – указатель на исходные гиперспектральные данные;
- `uint16_t * out` – указатель на область памяти, в которой будут сохраняться результаты работы алгоритма;
- `struct ImageMetadata * imageMeta` – указатель на структуру, содержащую информацию о исходном изображении (таблица 3.2);
- `struct PredictorMetadata * predMeta` – указатель на структуру, в которой содержатся конфигурационные параметры предсказателя (таблица 3.1);
- `int opType` – переменная, задающая режим работы предсказателя: генерация отображенных остатков либо восстановление изображения.

Шаг 1. Переменным `sizeX`, `sizeY`, `sizeZ` типа `int` и присвоить им значение, соответственно, полей `xSize`, `ySize`, `zSize` структуры `imageMeta`.

Шаг 2. Переменной `D` типа `int` присвоить значение поля `dynamicRange` структуры `imageMeta`.

Шаг 3. Если значение `D` оказалось равно нулю, то присвоить переменной `D` значение 16.

Шаг 4. Переменной `P` типа `int` присвоить значение поля `predictionBands` структуры `predMeta`.

Шаг 5. Переменной `R` типа `int` присвоить значение поля `registerSize` структуры `predMeta`.

Шаг 6. Переменной `Om` типа `int` присвоить увеличенное на четыре значение поля `weightComponentResolution` структуры `predMeta`.

Шаг 7. Переменным `v_min` и `v_max` типа `int` присвоить уменьшенные на шесть значения, соответственно полей `wuScalingExpInitialParam` и `wuScalingExpFinalParam` структуры `predMeta`.

Шаг 8. Переменной `t_inc` типа `int` задать единичное значение и произвести операцию «сдвиг влево» на количество разрядов, равное увеличенному на четыре значению поля `wuScalingExpChangeInterval` структуры `predMeta`.

Шаг 9. Переменной `w_min` типа `int` задать умноженный на минус единицу результат операции «сдвиг влево» единицы на количество разрядов, равное увеличенному на два значению переменной `Om`.

Шаг 10. Переменной `w_max` типа `int` задать результата операции «сдвига влево» единицы на количество разрядов, равное увеличенному на два значению переменной `Om`. Уменьшить значение переменной `w_max` на

единицу.

Шаг 11. Объявить переменные  $s\_min$ ,  $s\_max$ ,  $s\_mid$  типа `int`. Переменной  $s\_min$  присвоить нулевое значение. Переменной  $s\_max$  – уменьшенной на единицу результат операции «сдвиг влево» единицы на  $D$  разрядов. В переменную  $s\_max$  записать уменьшенный на единицу результат операции «сдвиг влево» единицы на  $D-1$  разрядов.

Шаг 12. Переменной  $bandSize$  присвоить результат перемножения значений переменных  $sizeY$  и  $sizeX$ .

Шаг 13. Объявить переменные  $curBandIn$ ,  $curBandOut$ ,  $predBase$ ,  $curPredBase$  типа `uint16_t *`. Переменной  $curBandIn$  присвоить значение переменной  $in$ , переменной  $curBandOut$  – значение переменной  $out$ .

Шаг 14. Если значение переменной  $opType$  равно константе `PREDICTOR_MAP`, тогда присвоить переменной  $predBase$  значение  $in$ , иначе – значение  $out$ .

Шаг 15. Переменной  $uwSize$  присвоить увеличенное на три значение переменной  $P$ .

Шаг 16. Вычислить размер необходимой памяти для хранения массивов векторов разниц  $\vec{U}_z(t)$  и весов  $\vec{W}_z(t)$  и присвоить переменной  $uwSizeBytes$  типа `size_t`.

Шаг 17. Запросить у системы память размером  $uwSizeBytes$  под хранение массива векторов  $\vec{U}_z(t)$ . В случае неудачи перейти к шагу 45. В случае успеха присвоить адрес выделенной памяти переменной  $msU$  типа `int *`.

Шаг 18. Запросить у системы память размером  $uwSizeBytes$  под массив векторов весов  $\vec{W}_z(t)$ . В случае неудачи перейти к шагу 45. В случае успеха присвоить адрес выделенной памяти переменной  $msW$  типа `int *`.

Шаг 19. Инициализировать первый элемент массива векторов весов  $\vec{W}_z(t)$  значениями по-умолчанию, применив алгоритм инициализации, описанный в пункте 4.3. Заполнить значением первого элемента массива весь массив векторов весов.

Шаг 20. Переменной  $z$  типа `int` присвоить значение ноль.

Шаг 21. Если значение переменной  $z$  равно или больше значения  $sizeZ$ , то перейти к шагу 45.

Шаг 22. Переменной  $curU$  типа `int *` присвоить результат смещения указателя  $msU$  на  $uwSize$  позиций.

Шаг 23. Переменной  $curW$  типа `int *` присвоить результат смещения указателя  $msW$  на  $uwSize$  позиций.

Шаг 24. Объявить переменную  $scale\_s\_pred$  типа `int`.

Шаг 25. Если значения  $P$  и  $z$  больше нуля, присвоить переменной

`scale_s_pred` удвоенное значение элемента предыдущего спектрального слоя с теми же координатами `y` и `x`. Иначе присвоить `scale_s_pred` удвоенное значение `s_mid`.

Шаг 26. Если значение переменной `opType` равно значению константы `PREDICTOR_MAP`, по алгоритму из пункта 4.4 вычислить отображенный остаток предсказания и записать полученное значение в ячейку `curBandOut[0]`. Иначе по алгоритму из пункта 4.5 вычислить исходное значение элемента и записать в ячейку `curBandOut[0]`.

Шаг 27. Объявить переменные `x`, `y` и `t` типа `int`. Присвоить переменной `t` единичное значение, переменной `y` – нулевое значение.

Шаг 28. Если значение `y` больше или равно значению `sizeY`, перейти к шагу 44.

Шаг 29. Если значение `y` равно нулю, то присвоить переменной `x` единичное значение. Иначе присвоить `x` нулевое значение.

Шаг 30. Если значение `x` больше либо равно `sizeX`, перейти к шагу 43.

Шаг 31. Вычислить по алгоритму из пункта 4.1 локальную сумму. В качестве входных данных использовать значения переменных `curPredBase`, `sizeY`, `sizeX`, `y`, `x`. Результат вычисления записать в переменную `local_sum` типа `int`.

Шаг 32. Заполнить по алгоритму из пункта 4.2 вектор локальных разниц, находящийся по адресу `curU`. В качестве входных данных использовать значения переменных `curU`, `P`, `predBase`, `sizeY`, `sizeX`, `z`, `y`, `x`.

Шаг 33. Вычислить сумму поэлементных произведений векторов `curU` и `curW`. Записать результат вычислений в переменную `d_pred` типа `int`.

Шаг 34. Вычесть из значения переменной `local_sum` учетверенное значение `s_mid`. С полученным результатом произвести операцию «сдвиг влево» на `Om` бит и прибавить значение переменной `d_pred`. Результат проведенных операций записать в переменную `a` типа `int`.

Шаг 35. Провести операцию `mod_R` для переменной `a`. Результат записать в переменную `val` типа `int`. Выполнить операцию «сдвиг вправо» для переменной `val` на количество бит, равное увеличенному на единицу значению переменной `Om`.

Шаг 36. Добавить к переменной `val` увеличенное на единицу удвоенное значение переменной `s_mid`.

Шаг 37. Присвоить переменной `scale_s_pread` результат ограничения значения переменной `val` по нижней границе, равной удвоенному значению `s_mid`, и верхней границе, равной увеличенному на единицу удвоенному значению переменной `s_max`.

Шаг 38. Если значение переменной `opType` равно значению константы `PREDICTOR_MAP`, по алгоритму из пункта 4.4 вычислить отображенный

остаток предсказания и записать полученное значение в ячейку `curBandOut[t]`. Иначе по алгоритму из пункта 4.5 вычислить исходное значение элемента и записать в ячейку `curBandOut[t]`.

Шаг 39. Вычислить значение ошибки предсказания как результат вычитания значения переменной `scale_s_pred` из удвоенного значения ячейки массива `curPredBase` с индексом `t`. Результат присвоить переменной `e` типа `int`.

Шаг 40. По вычислить значение масштабирующей экспоненты  $\rho(t)$ . В качестве входных данных принять значение переменных `D`, `Om`, `v_min`, `v_max`, `t`, `t_inc`, `sizeX`. Результат записать в переменную `ro` типа `int`.

Шаг 41. По алгоритму из пункта 4.x провести обновление вектора весов, адрес которого хранится в переменной `curW`. В качестве входных данных для алгоритма принять значения переменных `curW`, `curU`, `uwSize`, `e`, `ro`, `w_min`, `w_max`.

Шаг 42. Передвинуть указатели `curU` и `curW` на `uwSize` единиц вперед, увеличить значения переменных `x` и `t` на единицу и перейти к шагу 30.

Шаг 43. Увеличить значение переменной `y` на единицу и перейти к шагу 28.

Шаг 44. Передвинуть указатели `curBandIn`, `curBandOut` и `curPredBase` на величину `bandSize` вперед и перейти к шагу 21.

Шаг 45. Если значение указателя `msU` не нулевое, то освободить память по данному адресу.

Шаг 46. Если значение указателя `msW` не нулевое, то освободить память по данному адресу.

Шаг 47. Конец алгоритма.

Результат выполнения алгоритма расположен в памяти по адресу, хранящемуся в указателе `out`.

#### 4.7 Алгоритм работы элементарно-адаптивного энтропийного кодера

Элементарно-адаптивный энтропийный кодер производит кодирование каждого отображенного остатка предсказания  $\delta_z(t)$  в бинарное слово переменной длины. Схема данного алгоритма приведена на чертеже ГУИР.400201.013 ПД.2.

В качестве входных данных имеем следующие:

- `uint16_t * in` – указатель на трехмерный массив отображенных в область положительных значений остатков предсказания;
- `uint32_t * out` – указатель на блок памяти для сохранения результатов работы кодера;
- `size_t * outSize` – указатель на переменную, в которую будет записан итоговый размер выходных данных в байтах;

– `struct ImageMetadata * imageMeta` – указатель на структуру, содержащую информацию о исходном изображении (таблица 3.2);  
– `struct EncoderMetadata * encoderMeta` – указатель на структуру, задающую параметры работы энтропийного кодера (таблица 3.3).

Приведем алгоритм элементарно-адаптивного кодера по шагам.

Шаг 1. Объявить переменные `sizeX`, `sizeY`, `sizeZ` типа `int` и присвоить им, соответственно, значения полей `xSize`, `ySize`, `zSize` структуры `imageMeta`.

Шаг 2. Объявить переменную `D`, присвоить ей значения поля `dynamicRange` структуры `imageMeta`. Если значение `D` равно нулю, то задать `D` значение 16.

Шаг 3. Объявить переменную `bandSize` и присвоить им результат произведения значений переменных `sizeX` и `sizeY`.

Шаг 4. Объявить переменную `curBandIn` типа `uint16_t *` и присвоить ей значение указателя `in`.

Шаг 5. Объявить переменную `curBandOut` типа `uint32_t *` и присвоить ей значение указателя `out`.

Шаг 6. Объявить переменную `U_max` типа `unsigned`, присвоить ей значение битового поля `unaryLengthLimit` структуры `encoderMeta`;

Шаг 7. Объявить переменную `gamma_0` типа `unsigned`, присвоить ей значение битового поля `accumInitConstant` структуры `encoderMeta`;

Шаг 8. Объявить переменную `gamma` типа `unsigned` и присвоить увеличенное на четыре значение битового поля `rescalingCounterSize` структуры `encoderMeta`.

Шаг 9. Объявить переменную `k` типа `unsigned` и присвоить ей значение битового поля `accumInitConstant` структуры `encoderMeta`.

Шаг 10. Объявить переменную `counter` типа `size_t`, присвоить ей результат операции «сдвиг влево» единицы на `gamma_0` разрядов.

Шаг 11. Объявить переменную `accum`, присвоить ей результат выполнения функции `initAccum()`, принимающей в качестве параметров значения переменных `k` и `counter`.

Шаг 12. Объявить переменную `buffer` типа `uint64_t`, `bitCounter`, `z`, `i` типа `uint32_t`. Обнулить указанные переменные.

Шаг 13. Если значение переменной `z` больше или равно значению `sizeZ`, перейти к шагу 32.

Шаг 14. Выполнить операцию «сдвиг влево» переменной `buffer` на `D` бит, увеличить значение `bitCounter` на `D` единиц.

Шаг 15. Прибавить к переменной `buffer` значение `curBandIn[0]`.

Шаг 16. Присвоить переменной `i` единичное значение.

Шаг 17. Если `i` больше или равно `bandSize`, перейти к шагу 31.

Шаг 18. Если значение переменной `counter` меньше `counterLimit`, увеличить значение переменной `accum` на величину `curBandIn[i-1]`, увеличить значение `counter` на единицу. Иначе увеличить значение `accum` на `curBandIn[i-1]`, увеличить значение `counter` на единицу, применить к переменным `accum` и `counter` операцию «сдвиг вправо» на один разряд.

Шаг 19. Переменной `k` присвоить результат работы функции `getCodeWordSize()`, принимающей в качестве параметров значения `counter` и `accum`.

Шаг 20. Присвоить результат деления значения ячейки `curBandIn[i]` на два в степени `k` переменной `u` типа `uint32_t`, а остаток от этого деления – переменной `rem` типа `int32_t`.

Шаг 21. Если значение `u` больше или равно значению `U_max`, присвоить `u` значение `U_max`.

Шаг 22. Выполнить операцию «сдвиг влево» переменной `buffer` на `u` разрядов, увеличить `bitCounter` на величину `u`.

Шаг 23. Если значение `u` больше или равно `U_max`, перейти к шагу 26.

Шаг 24. Выполнить операцию «сдвиг влево» переменной `buffer` на один разряд, увеличить значение переменных `bitCounter` и `buffer` на единицу.

Шаг 25. Выполнить операцию «сдвиг влево» переменной `buffer` на `k` разрядов, увеличить значение `bitCounter` на значение `k`, добавить к переменной `buffer` значение переменной `rem`. Перейти к шагу 27.

Шаг 26. Выполнить операцию «сдвиг влево» переменной `buffer` на `D` разрядов, увеличить значение счетчика `bitCounter` на `D`, добавить к переменной `buffer` значение `curBandIn[i]`.

Шаг 27. Если значение `bitCounter` меньше 32, перейти к шагу 30.

Шаг 28. Уменьшить значение `bitCounter` на 32, записать в переменную `val` типа `uint32_t` результат операции «сдвиг вправо» значения переменной `buffer` на количество разрядов, равное значению `bitCounter`.

Шаг 29. Записать по адресу, хранящемуся в указателе `curBandOut` значение переменной `val` и инкрементировать указатель.

Шаг 30. Инкрементировать переменную `i` и перейти к шагу 17.

Шаг 31. Передвинуть указатель `curBandIn` на `bandSize` элементов вперед, инкрементировать переменную `z` и перейти к шагу 13.

Шаг 32. Если значение переменной `bitCounter` равно нулю, то перейти к шагу 36.

Шаг 33. От 32 отнять значение `bitCounter`, присвоить полученное значение переменной `val`.

Шаг 34. Выполнить «сдвиг влево» значения переменной `buffer` на количество разрядов, равное значению `val`, записать младшие 32 бита



результата по адресу, хранящемуся в указателе `curBandOut`.

Шаг 35. Инкрементировать указатель `curBandOut`.

Шаг 36. Записать по адресу в указателе `outSize` результат вычитания указателя `out` из указателя `curBandOut` умноженный на четыре.

Шаг 37. Конец алгоритма.

Результаты работы алгоритма расположены в памяти по адресу, хранящемуся в указателях `out` и `outSize`.

#### **4.8 Алгоритм работы элементно-адаптивного энтропийного декодера**

Элементно-адаптивный декодер выполняет процесс восстановления трехмерного массива отображенных остатков предсказания из массива кодированных данных. Схема алгоритма работы элементно-адаптивного декодера приведена на чертеже ГУИР.400201.013 ПД.3. Данный алгоритм реализован в функции `decodeGolomb()` имеющей следующие входные параметры:

- `uint32_t * in` – указатель на блок памяти, хранящий кодированные данные;

- `uint16_t * out` – указатель на блок памяти для сохранения результатов работы декодера;

- `struct ImageMetadata * imageMeta` – указатель на структуру, содержащую информацию о исходном изображении (таблица 3.2);

- `struct EncoderMetadata * encoderMeta` – указатель на структуру, хранящую информацию о параметрах, на основе которых производилось кодирование (таблица 3.3).

Опишем алгоритм работы элементно-адаптивного декодера по шагам. Шаги 1-12 аналогичны шагам алгоритма элементно-адаптивного кодера (см. пункт 4.7).

Шаг 13. Объявить переменные `samplesCounter`, `bandsCounter` типа `uint32_t`, `val` типа `int16_t` и присвоить всем нулевое значение.

Шаг 14. Объявить переменную `lastBit` типа `uint64_t`, присвоить ей единичное значение и выполнить операцию «сдвиг влево» на 63 разряда.

Шаг 15. Прибавить к переменной `buffer` значение типа `uint32_t`, находящееся по адресу, хранящемуся в указателе `curBandIn`.

Шаг 16. Выполнить «сдвиг влево» переменной `buffer` на 32 разряда.

Шаг 17. Инкрементировать указатель `curBandIn`.

Шаг 18. Прибавить к переменной `buffer` значение типа `uint32_t`, находящееся по адресу, хранящемуся в указателе `curBandIn`.

Шаг 19. Инкрементировать указатель `curBandIn`.

Шаг 20. Если значение переменной `bandsCounter` больше либо равно значению `sizeZ`, тогда перейти к шагу 47.

Шаг 21. Записать в переменную `val` младшие 16 бит результата операции «сдвиг вправо» 64-битного значения переменной `buffer` на количество разрядов, равное результату вычитания из 64 значения переменной `D`.

Шаг 22. Выполнить «сдвиг влево» переменной `buffer` на `D` разрядов, увеличить значение переменной `bitCounter` на `D` единиц.

Шаг 23. Записать по указателю `curBandOut` значение переменной `val`.

Шаг 24. Присвоить переменной `samplesCounter` единичное значение.

Шаг 25. Если значение `sampleCounter` больше либо равно значению `bandSize`, перейти к шагу 46.

Шаг 26. Если значение переменной `counter` меньше `counterLimit`, увеличить значение переменной `accum` на величину `val`, увеличить значение `counter` на единицу, перейти к шагу 28.

Шаг 27. Увеличить значение `accum` на `val`, увеличить значение `counter` на единицу, применить к переменным `accum` и `counter` операцию «сдвиг вправо» на один разряд.

Шаг 28. Переменной `k` присвоить результат работы функции `getCodeWordSize()`, принимающей в качестве параметров значения `counter` и `accum`.

Шаг 29. Объявить переменную `u` типа `uint16_t` и инициализировать ее нулевым значением.

Шаг 30. Выполнить операцию логического И между переменными `buffer` и `lastBit`. Если результат операции не нулевой, тогда перейти к шагу 33.

Шаг 31. Инкрементировать переменные `u` и `bitCounter`, выполнить операцию «сдвиг влево» значения переменной `buffer` на один разряд.

Шаг 32. Если значение переменной `u` не равно `U_max` перейти к шагу 34. Иначе перейти к шагу 30.

Шаг 33. Если значение переменной `u` не равно `U_max`, перейти к шагу 36.

Шаг 34. Записать в переменную `val` младшие 16 бит результата операции «сдвиг вправо» 64-битного значения переменной `buffer` на количество разрядов, равное результату вычитания из 64 значения переменной `D`.

Шаг 35. Выполнить «сдвиг влево» переменной `buffer` на `D` разрядов, увеличить значение переменной `bitCounter` на `D` единиц и перейти к шагу 39.

Шаг 36. Записать в переменную `val` результат «сдвига влево» значения переменной `u` на `k` разрядов.

Шаг 37. Выполнить «сдвиг влево» значения переменной `buffer` и инкрементировать значение переменной `bitCounter`.

Шаг 38. Если значение переменной `k` больше нуля, то прибавить к переменной `val` младшие 16 бит результата «сдвига вправо» значения переменной `buffer` на количество разрядов, равное результату вычитания из 64 значения переменной `k`, выполнить «сдвиг влево» значения переменной `buffer` и увеличить значение переменной `bitCounter` на `k`.

Шаг 39. Если значение переменной `bitCounter` меньше 32, перейти к шагу 44.

Шаг 40. Уменьшить значение переменной `bitCounter` на 32.

Шаг 41. Выполнить «сдвиг вправо» значения переменной `buffer` на количество разрядов, равное значению `bitCounter`.

Шаг 42. Прибавить к переменной `buffer` значение типа `uint32_t`, находящееся по адресу, хранящемуся в указателе `curBandIn` и инкрементировать значение указателя.

Шаг 43. Выполнить «сдвиг влево» значения переменной `buffer` на количество разрядов, равное значению `bitCounter`.

Шаг 44. Записать значение переменной `val` в ячейку массива `curBandOut` с индексом, равным значению переменной `samplesCounter`.

Шаг 45. Инкрементировать значение переменной `samplesCounter` и перейти к шагу 25.

Шаг 46. Инкрементировать значение переменной `bandsCounter`, сместить указатель `curBandOut` вперед на `bandSize` позиций и перейти к шагу 20.

Шаг 47. Конец алгоритма.

Результат работы алгоритма находится в памяти, адресуемой указателем `out`.

## 5 ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ

Основными задачами, которые должно выполнять разработанное приложение, является сжатие и восстановление гиперспектральных изображений. Соответственно, проведем тестирование работы программы в каждом из режимов по отдельности. Также, большой интерес представляет скорость и эффективность работы программы в сравнении с распространенными универсальными программами-архиваторами, использующими различные алгоритмы сжатия.

### 5.1 Описание исходных данных

Для начала протестируем процесс сжатия изображений. В качестве гиперспектрального изображения примем набор файлов в формате PGM, каждый из которых хранит в себе данные фотосъемки одного и того же объекта, полученную в разные моменты времени. Каждый отдельный PGM-файл играет роль отдельного спектрального канала гиперспектрального изображения. Цветовая глубина для полученных снимков составляет 11 бит, а это означает, что значения пикселей изображения могут изменяться в пределах от 0 до 4095. Линейный размер одного изображения составляет 745x1024 пикселей.

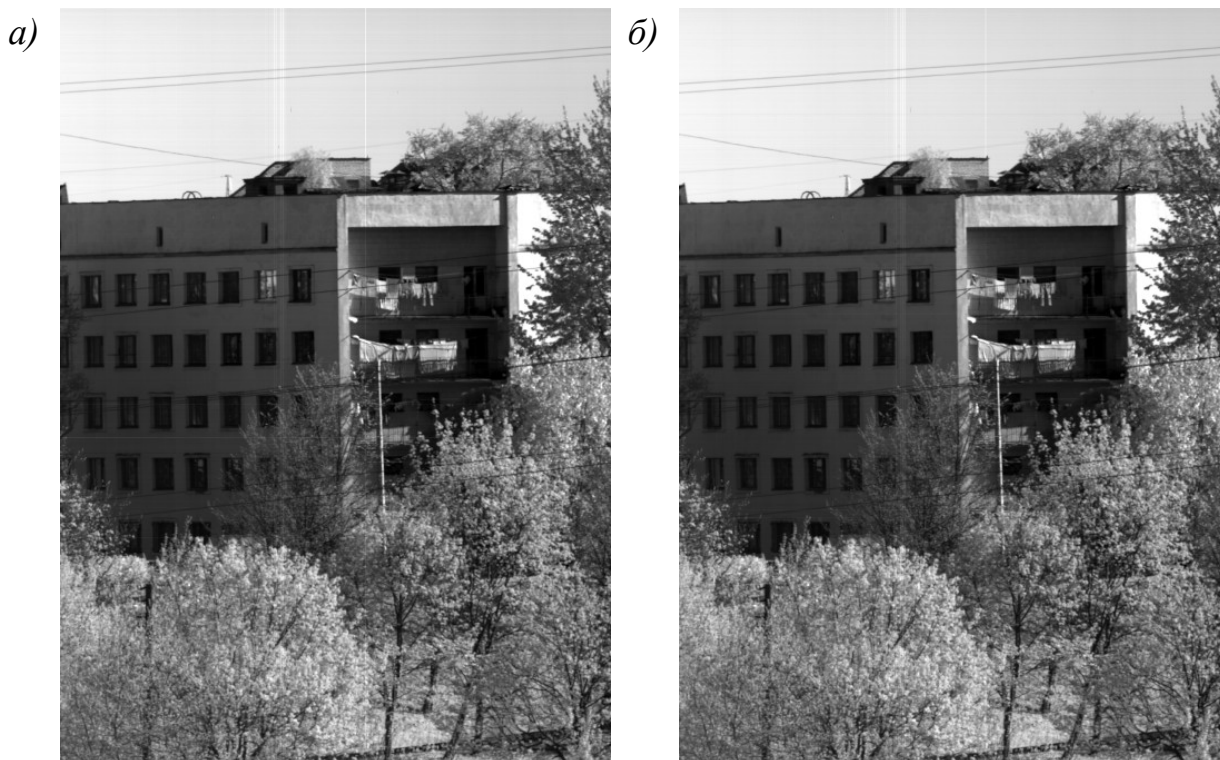


Рисунок 5.1 – Визуализация данных из PGM-файлов:  
а – 0000.pgm; б – 0001.pgm

Для наглядности, на рисунке 5.1 приведена визуализация данных из PGM-файлов, динамический диапазон которых был сжат в интервал от 0 до 255, чтобы иметь возможность представить их как изображение в оттенках серого. Человеческому глазу трудно найти различия между изображением *а* и изображением *б*, но они имеются. В этом можно убедиться, глядя на результат «вычитания» этих изображений друг из друга с последующим инвертированием цветов (рисунок 5.2). На изображении с рисунка 5.2 четко видны горизонтальные и вертикальные полосы, являющиеся следствием помех, повлиявших на качество съемки в разные моменты времени. При абсолютно идентичных изображениях мы бы наблюдали лишь чистый лист бумаги.

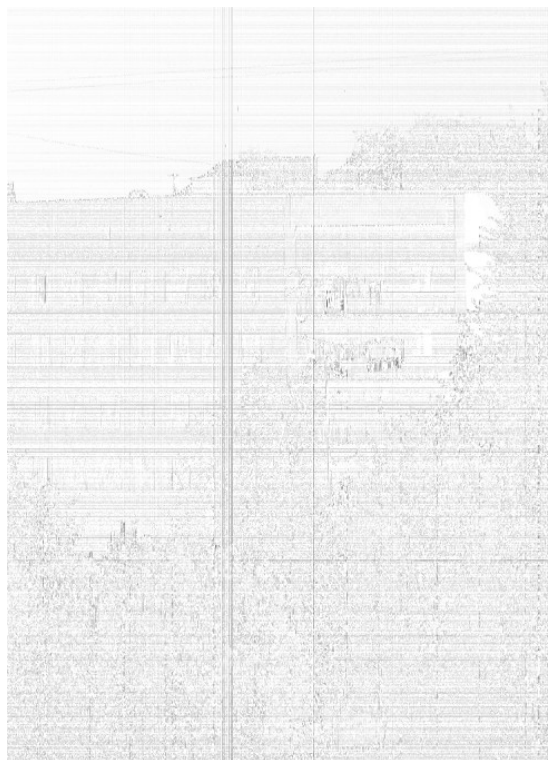


Рисунок 5.2 – Результат «вычитания» изображения *а* из *б*

За счет присутствия этих незначительных различий данных в тестируемом на сжатие наборе файлов и будет достигаться эффект многослойного по своей структуре гиперспектрального изображения.

## 5.2 Тестирование работы программы в режиме сжатия

Откроем терминал, перейдем в папку с PGM-файлами и выполним следующую команду:

```
> fl-compressor --compress 0000.pgm 0001.pgm 0002.pgm  
--output compressed.fl
```

Тем самым мы предложим программе fl-compressor выполнить сжатие (ключ `--compress`) файлов с именами 0000.pgm 0001.pgm 0002.pgm 0003.pgm, и сохранить результат в файле с именем test\_compress.fl (ключ `--output`).

После работы программы в терминал выведена информация, изображенная на рисунке 5.3.

```
Opening files...
Start compression 3 files totaling 4577280 bytes (4.37 MB)...
Prediction...
100% [#####]
Prediction done.
Encoding...
100% [#####]
Encoding done.
Compression done.
Elapsed time is 0.531250 seconds.
Compressed file size is 1976068 bytes (1.88 MB).
Compression ratio is 43.17%.
```

Рисунок 5.3 – Результаты сжатия трех файлов

Анализируя вывод программы, можно заключить, что за время, приблизительно равное 0,5 секунды было сжато три файла общим объемом 4,37 Мбайт. Объем сжатого файла составил 1,88 Мбайт, степень сжатия составила 43,17%.

Выполнив в терминале команду `ls *.fl` убедимся, что в папке появился сжатый файл compressed.fl, размер которого в точности совпадает с указанным размером в выводе программы (рисунок 5.4).

```
PS C:\Users\user\test_compression> ls *.fl

Каталог: C:\Users\user\test_compression

Mode                LastWriteTime         Length Name
----                -
-a----           28.05.2019    22:08         1976068 compressed.fl
```

Рисунок 5.4 – Результаты команды `ls *.fl`

Проведем тестирование программы на сжатие больших объемов входных данных. Для этого выполним следующую команду:

```
> fl-compressor --compress --output test_large.fl  
pgm_files/*.pgm
```

Данная команда предлагает программе воспринять все файлы из папки `pgm_files`, имеющие расширение `.pgm` как слои гиперспектрального изображения и выполнить их сжатие в файл `test_large.fl`. Результаты выполнения команды приведены на рисунке 5.5.

```
Opening files...  
Start compression 256 files totaling 390594560 bytes (372.50 MB)...  
Prediction...  
100% [#####]  
Prediction done.  
Encoding...  
100% [#####]  
Encoding done.  
Compression done.  
Elapsed time is 39.281250 seconds.  
Compressed file size is 157491268 bytes (150.20 MB).  
Compression ratio is 40.32%.
```

Рисунок 5.5 – Результаты сжатия 256 файлов

Вывод программы (рисунок 5.5) говорит об успешном завершении операции. Было сжато 256 файлов общим объемом 372,50 Мбайт. Размер сжатого файла составил 150,20 Мбайт, что составляет 40,32% от общего размера исходных файлов. Затраченное время приблизительно равно 39 секундам. Выполнив команду `ls *.fl` убеждаемся в существовании сжатого файла с именем `test_large.fl` (рисунок 5.6).

Каталог: C:\Users\user\test_compression			
Mode	LastWriteTime		Length Name
----	-----		-----
-a----	28.05.2019	22:27	1976068 compressed.fl
-a----	28.05.2019	22:58	157491268 test_large.fl

Рисунок 5.6 – Проверка существования файла `test_large.fl`

Таким образом, две проведенные операции завершились генерацией файлов, имеющих меньший размер, чем исходные данные, что дает предпосылки говорить о корректности проведения операции сжатия. Полную уверенность в этом можно получить только при успешном выполнении

обратной операции – операции восстановления исходных изображений из сжатого файла.

### 5.3 Тестирование работы программы в режиме восстановления

Выполним восстановление сжатых гиперспектральных изображений из файлов, полученных в пункте 5.2. Для этого откроем терминал, перейдем в папку со сжатыми файлами, создадим папку `restored_files` и выполним следующую команду:

```
> fl-compressor --decompress compressed.fl --output  
.\restored_files\
```

Тем самым, предложим программе `fl-compressor` выполнить восстановление сжатых данных из файла `compressed.fl` в папку `restored_files`, которая находится в текущем каталоге. Результаты работы программы отражены на рисунке 5.7.

```
Starting decompression compressed.fl...  
Decoding...  
100% [#####]  
Decoding done.  
Restoration...  
100% [#####]  
Restoration done.  
Decompression done.  
Decompressed 3 files totaling 4.37 MB  
Elapsed time is 0.468750 seconds.
```

Рисунок 5.7 – Декомпрессия файла `compressed.fl`

Как можно заметить из вывода программы (рисунок 5.7), было восстановлено 3 файла общим объемом 4,37 Мбайт.

Каталог: C:\Users\user\test_compression\restored_files				
Mode	LastWriteTime		Length	Name
----	-----		-----	----
-a----	01.06.2019	16:44	1525777	0000.pgm
-a----	01.06.2019	16:44	1525777	0001.pgm
-a----	01.06.2019	16:44	1525777	0002.pgm

Рисунок 5.8 – Каталог с восстановленными файлами



Убедиться в существовании восстановленных файлов, можно выполнив команду `ls .\restored_files` (рисунок 5.8). Выполним сравнение исходных и восстановленных файлов. Для этого воспользуемся утилитой File Checksum Integrity Verifier (FCIV) от Microsoft.

FCIV – это утилита командной строки, которая вычисляет и проверяет криптографические значения хеш-файлов. FCIV может вычислять криптографические значения хеша MD5 или SHA-1. Эти значения могут быть отображены на экране или сохранены в формате XML для последующего использования и проверки.

Для расчета MD5-хешей восстановленных файлов выполним команду:

```
> fciv.exe .\restored_files\
```

Данная команда рассчитает MD5-хеши для всех файлов, находящихся в папке `restored_files`. Результаты выполнения команды приведены на рисунке 5.9.

```
PS C:\Users\user\test_compression> fciv.exe .\restored_files\  
//  
// File Checksum Integrity Verifier version 2.05.  
//  
c47116ddd6acda7d8626bdaacdc01f0c .\restored_files\0000.pgm  
819ffeed8f9fbc6b5a644df63571da0d .\restored_files\0001.pgm  
687680c2a700d99bbcb9636af41c4cf3 .\restored_files\0002.pgm
```

Рисунок 5.9 – Значения MD5-хешей для восстановленных файлов

Теперь рассчитаем MD5-хеши для оригинальных файлов, а после сравним получившиеся результаты. Для этого выполним команду:

```
> fciv.exe .\pgm_files\
```

Результаты выполнения команды приведены на рисунке 5.10.

```
PS C:\Users\user\test_compression> fciv.exe .\pgm_files\  
//  
// File Checksum Integrity Verifier version 2.05.  
//  
c47116ddd6acda7d8626bdaacdc01f0c .\pgm_files\0000.pgm  
819ffeed8f9fbc6b5a644df63571da0d .\pgm_files\0001.pgm  
687680c2a700d99bbcb9636af41c4cf3 .\pgm_files\0002.pgm
```

Рисунок 5.10 – Значения MD5-хешей для оригинальных файлов

Анализируя результаты расчетов MD5-хешей для восстановленных и

оригинальных файлов, легко заметить, что хеши идентичны. А это значит, что разработанный программный модуль корректно выполняет две главные его задачи – сжатие трехмерных гиперспектральных данных без потерь, а также их восстановление в первоначальное состояние.

Также убедимся, что программа способна выполнять восстановление данных большого объема. Для этого выполним декомпрессию файла `test_large.fl`, полученного в пункте 5.2 в ходе тестирования сжатия 256 файлов. Откроем в терминале папку с файлом `test_large.fl`, создадим папку `restored_large` и выполним следующую команду:

```
> fl-compressor --decompress test_large.fl --output  
.\restored_large\
```

Результаты выполнения команды приведены на рисунке 5.11. Вывод программы информирует нас о том, что в процессе декомпрессии файла `test_large.fl` было восстановлено 256 файлов, общий объем которых составил 372,50 Мбайт. Затраченное время составило 36 секунд.

```
Starting decompression test_large.fl...  
Decoding...  
100% [#####]  
Decoding done.  
Restoration...  
100% [#####]  
Restoration done.  
Decompression done.  
Decompressed 256 files totaling 372.50 MB  
Elapsed time is 36.296875 seconds.
```

Рисунок 5.11 – Декомпрессия файла `test_large.fl`

Убедимся в наличии восстановленных файлов можно, выполнив команду `ls .\restored_large\ | sort -desc`, которая выведет список файлов в указанном каталоге, начиная с конца (рисунок 5.12).

```
Каталог: C:\Users\user\test_compression\restored_large
```

Mode		LastWriteTime	Length	Name
----		-----	-----	----
-a----	01.06.2019	21:02	1525777	0255.pgm
-a----	01.06.2019	21:02	1525777	0254.pgm
-a----	01.06.2019	21:02	1525777	0253.pgm

Рисунок 5.12 – Фрагмент содержимого каталога `restored_large`

## 5.4 Тестирование работы программы с различными параметрами сжатия

Разработанная программа имеет несколько настраиваемых параметров сжатия, среди которых можно выделить:

1 Предел длины унарной части кодового слова  $U_{max}$ , задаваемый ключом `-U`. Может изменяться в пределах от 1 до 20;

2 Количество спектральных слоев, используемых адаптивным предсказателем  $P$ . Значение параметра  $P$  задается ключом `-P` и может изменяться в пределах от 0 до 15.

3 Скорость и пределы адаптации вектора весов  $\vec{W}_z(t)$  к набору входных данных. Скорость адаптации  $t_{inc}$  задается ключом `-t_inc`, а верхний  $v_{max}$  и нижний  $v_{min}$  пределы ключами `-nu_max` и `-nu_min` соответственно.

Для определения зависимости степени сжатия от предела длины унарной части кодового слова, было выполнено сжатие 20 тестовых PGM-файлов с последовательными номерами, выполняющих роль спектральных слоев гиперспектрального изображения, во всем диапазоне допустимых значений параметра  $U_{max}$ . В качестве примера, приведем команду для сжатия 20 файлов с величиной  $U_{max}$ , равной 10:

```
> fl-compressor --compress .\pgm_files\000*.pgm .\pgm_files\001*.pgm --output test_U.fl -U 10
```

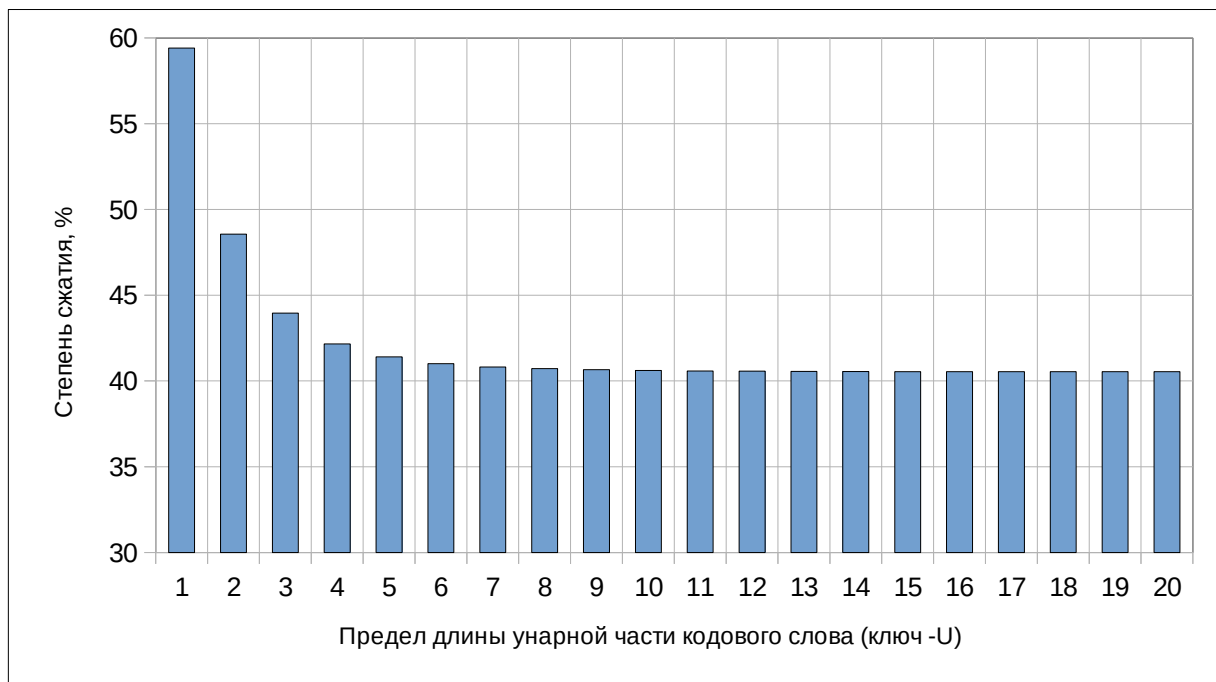


Рисунок 5.13 – Зависимость степени сжатия от параметра  $U_{max}$

Результаты выполнения данного теста отражены на рисунке 5.13. Анализируя полученные результаты, можно сделать вывод о том, что для тестируемых изображений, наилучшая степень компрессии (порядка 40,5%) достигается при максимальной величине параметра  $U_{max}$ , равной 20, а наихудшая (порядка 59,0%) – при минимальной, равной единице. Приемлемая степень компрессии (порядка 40,7%) достигается уже при величине  $U_{max}$  равной 8, принятой в качестве значения по-умолчанию. Также необходимо отметить, что при проведении данного теста никакой зависимости времени выполнения тестов от величины параметра  $U_{max}$  выявлено не было. Среднее время сжатия составило порядка 3,2 секунды.

Аналогично проведем исследование зависимости степени и времени сжатия 20 тестовых PGM-файлов с последовательными номерами от количества спектральных слоев  $P$ , используемых адаптивным предсказателем. Напомним, что адаптивный предсказатель может использовать до 15 предшествующих спектральных слоев для прогнозирования значений. В качестве примера, приведем команду для сжатия 20 файлов с величиной  $P$ , равной 8:

```
> fl-compressor --compress .\pgm_files\000*.pgm .\pgm_files\
001*.pgm --output test_P.fl -P 8
```

Зависимость степени сжатия тестовых файлов от величины параметра  $P$  приведена на рисунке 5.14.

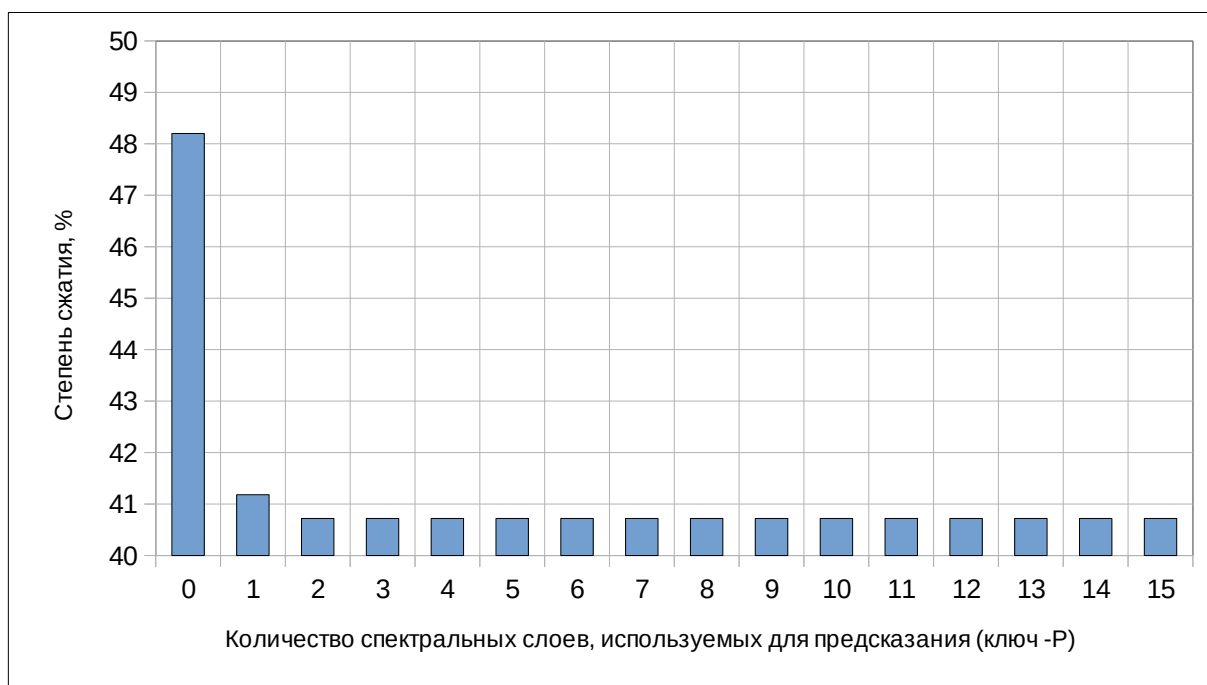


Рисунок 5.14 – Зависимость степени сжатия от параметра  $P$

Изучение полученных результатов приводит к выводу о том, что

наилучшая степень компрессии (40,72%) достигается уже при величине  $P=2$ . Наихудший результат (48,20%) был получен при  $P=0$ .

Зависимость времени сжатия от величины параметра  $P$  приведена на рисунке 5.15.



Рисунок 5.15 – Зависимость времени сжатия от параметра  $P$

Как видно из диаграммы на рисунке 5.15, наименьшее время сжатия (порядка 2,7 секунды) достигается при  $P=0$ , а наибольшее (6,2 секунды) – при  $P=15$ . Такой результат был вполне ожидаем, так как при большем значении параметра  $P$ , необходимо выполнить большее количество вычислений.

Анализируя диаграммы на рисунках 5.14 и 5.15 легко заметить, что, для данного набора тестовых данных, оптимальным значением количества спектральных слоев, используемых для предсказания значений, будет величина параметра  $P=2$ , так как при этом достигается наилучшая степень компрессии при незначительном возрастании времени сжатия. Поэтому, величина  $P=2$  была принята по-умолчанию.

Изучим влияние изменения параметров  $v_{min}$ ,  $v_{max}$  и  $t_{inc}$  на степень сжатия 20 тестовых PGM-файлов с последовательными номерами. Допустимые диапазоны изменения данных параметров – от 0 до 15. Примем  $v_{min}=0$ ,  $v_{max}=15$ , а  $t_{inc}$  будем изменять во всем допустимом диапазоне значений. Будем использовать следующую команду:

```
> fl-compressor --compress .\pgm_files\000*.pgm .\pgm_files\
001*.pgm --output test_tinc.fl -nu_min 0 -nu_max 15 -t_inc 0
```

Зависимость степени сжатия 20 тестовых файлов от величины параметра  $t_{inc}$ , задающего интервал адаптации вектора весов ко входным данным, приведена на рисунке 5.16.

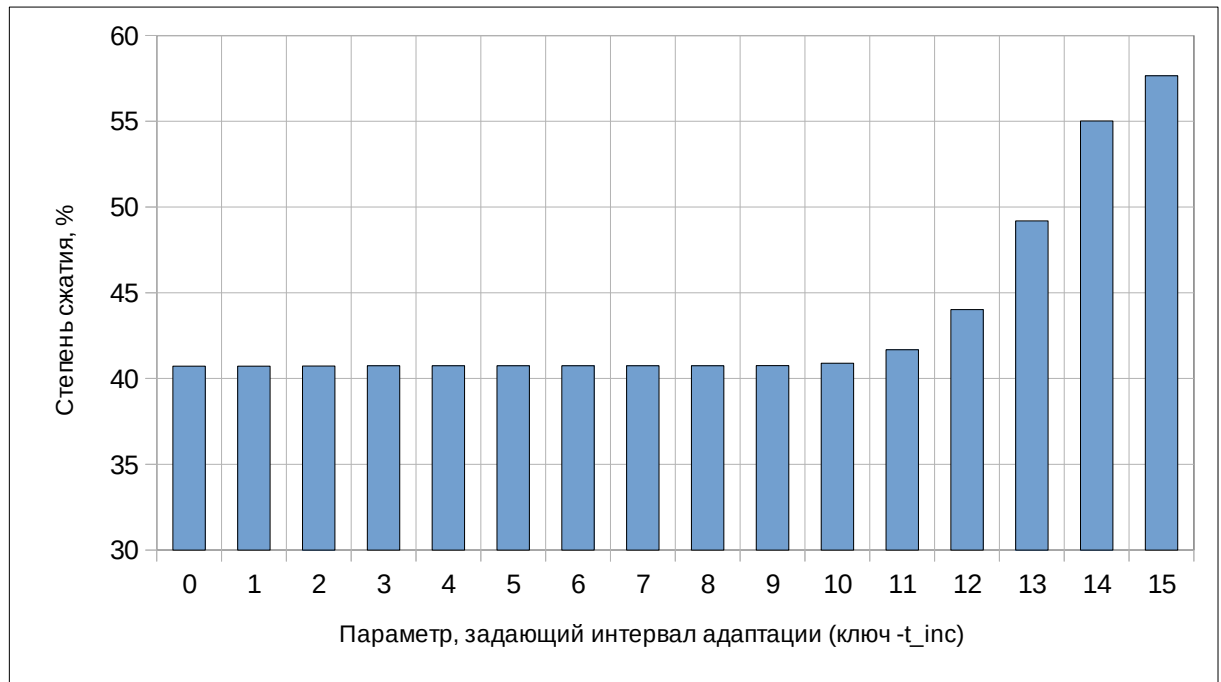


Рисунок 5.16 – Зависимость степени сжатия от параметра  $t_{inc}$

Анализируя полученные результаты, отметим что после того, как величина параметра  $t_{inc}$  становится больше 10, результаты сжатия начинают ухудшаться. Предположительно это связано с тем, при  $t_{inc} > 10$  адаптация становится слишком медленной, порождая большие ошибки предсказаний, что, в свою очередь, порождает коды Голомба-Райса большей длины, увеличивающие общий размер сжатого файла. В качестве значения по умолчанию было принято  $t_{inc} = 0$ .

### 5.5 Тестирование программы на ввод некорректных данных

Проведем тестирование на обработку программой некорректных аргументов командной строки, введенных пользователем. Для начала попытаемся сжать несуществующие файлы. Для этого выполним следующую команду:

```
> fl-compressor --compress .\pgm_files\1000.pgm .\pgm_files\2000.pgm .\pgm_files\3000.pgm --output test.fl
```

Вывод программы после выполнения данной команды представлен на рисунке 5.17.

```
Opening files...
Cannot load .\pgm_files\1000.pgm file.
Cannot load .\pgm_files\2000.pgm file.
Cannot load .\pgm_files\3000.pgm file.
No files has been uploaded.
Compression failed.
```

Рисунок 5.17 – Обработка попытки сжатия несуществующих файлов

Как следует из вывода программы (рисунок 5.17), попытка поиска и загрузки всех указанных пользователем файлов не удалась, поэтому программа завершилась с сообщением об ошибке.

Теперь попытаемся провести операцию сжатия существующих файлов, но неизвестного программе формата. Воспользуемся файлами, созданными после проведения экспериментов в пункте 5.4. Для этого выполним следующую команду:

```
> fl-compressor --compress test_P.fl test_t_inc.fl test_U.fl
--output test.fl
```

То есть, попытаемся сжать уже сжатые файлы. Результаты выполнения команды приведены на рисунке 5.18.

```
Opening files...
File test_large.fl has an invalid format. Must be PGM.
Cannot load test_large.fl file.
File test_t_inc.fl has an invalid format. Must be PGM.
Cannot load test_t_inc.fl file.
File test_U.fl has an invalid format. Must be PGM.
Cannot load test_U.fl file.
No files has been uploaded.
Compression failed.
```

Рисунок 5.18 – Обработка попытки сжатия файлов неверного формата

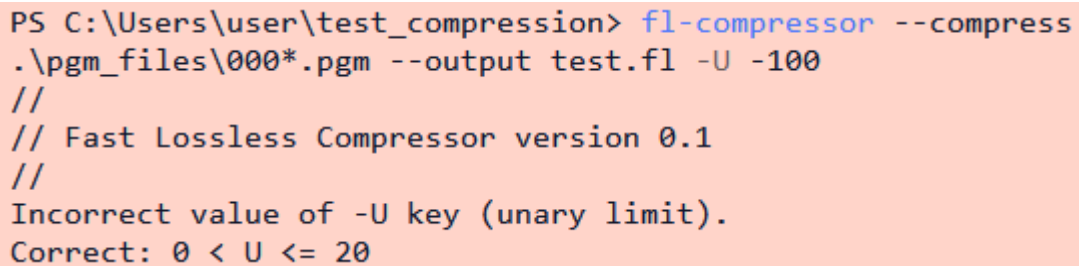
Изучая вывод программы можно заключить, что было совершено три попытки открытия файлов, каждый из которых имел неверный формат. В итоге программе не удалось выполнить загрузку ни одного файла, и она завершилась с сообщением об ошибке. В версии программы, тестируемой на данный момент, имеется возможность загружать в качестве слоев гиперспектрального изображения только файлы формата PGM.

Проведем тестирование разработанной программы на корректность обработки нарушения допустимых значений конфигурационных параметров, передаваемых посредством ключей-аргументов командной строки, детальное изучение которых было проведено в пункте 5.4.

Проверим обработку указания недопустимых значений для параметра  $U_{max}$ . Напомним, что допустимо указание положительного целого значения от 1 до 20. Выполним проверку на отрицательные значения, введя команду:

```
> fl-compressor --compress .\pgm_files\000*.pgm --output  
test.fl -U -100
```

Результат выполнения команды представлен на рисунке 5.19.



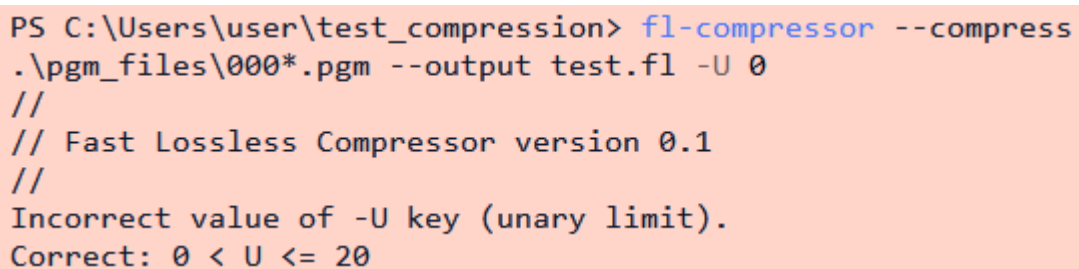
```
PS C:\Users\user\test_compression> fl-compressor --compress  
.\pgm_files\000*.pgm --output test.fl -U -100  
//  
// Fast Lossless Compressor version 0.1  
//  
Incorrect value of -U key (unary limit).  
Correct: 0 < U <= 20
```

Рисунок 5.19 – Обработка отрицательного значения ключа -U

Проведем тест программы на введение нулевого значения параметра  $U_{max}$ , выполнив следующую команду:

```
> fl-compressor --compress .\pgm_files\000*.pgm --output  
test.fl -U 0
```

Результат выполнения команды представлен на рисунке 5.20.



```
PS C:\Users\user\test_compression> fl-compressor --compress  
.\pgm_files\000*.pgm --output test.fl -U 0  
//  
// Fast Lossless Compressor version 0.1  
//  
Incorrect value of -U key (unary limit).  
Correct: 0 < U <= 20
```

Рисунок 5.20 – Обработка нулевого значения ключа -U

Также проведем тест на ввод положительного значения, превышающего установленный лимит, выполнив команду в терминале:

```
> fl-compressor --compress .\pgm_files\000*.pgm --output  
test.fl -U 500
```

Результаты выполнения команды представлены на рисунке 5.21. Анализируя проведенные тесты можно заключить, что программа корректно



обрабатывает ошибки пользовательского ввода значения параметра  $U_{max}$ .

```
PS C:\Users\user\test_compression> fl-compressor --compress
.\pgm_files\000*.pgm --output test.fl -U 500
//
// Fast Lossless Compressor version 0.1
//
Incorrect value of -U key (unary limit).
Correct: 0 < U <= 20
```

Рисунок 5.21 – Обработка некорректного значения ключа  $-U$

Протестируем программу на обработку некорректного пользовательского ввода значения параметра  $P$ , указывающего модулю адаптивного предсказателя, какое количество предшествующих спектральных слоев использовать при выполнении предсказания значений элементов трехмерного изображения. Корректными значениями для параметра  $P$  являются целые неотрицательные числа в диапазоне от 0 до 15.

Для проведения теста на отрицательные значения выполним следующую команду:

```
> fl-compressor --compress .\pgm_files\000*.pgm --output
test.fl -P -100500
```

Результаты выполнения команды представлены на рисунке 5.22.

```
PS C:\Users\user\test_compression> fl-compressor --compress
.\pgm_files\000*.pgm --output test.fl -P -100500
//
// Fast Lossless Compressor version 0.1
//
Incorrect value of -P key
(count of previous band for prediction)).
Correct: 0 <= P <= 15.
```

Рисунок 5.22 – Обработка отрицательного значения ключа  $-P$

Для проведения теста на большие положительные числа, превышающие заданный предел, выполним следующую команду:

```
> fl-compressor --compress .\pgm_files\000*.pgm --output
test.fl -P 666
```

Результаты выполнения команды представлены на рисунке 5.23.

```

PS C:\Users\user\test_compression> fl-compressor --compress
.\pgm_files\000*.pgm --output test.fl -P 666
//
// Fast Lossless Compressor version 0.1
//
Incorrect value of -P key
(count of previous band for prediction)).
Correct: 0 <= P <= 15.

```

Рисунок 5.23 – Обработка большого положительного значения ключа  $-P$

Из представленных выводов программы (рисунки 5.22, 5.23) делаем вывод, что некорректные значения параметра  $P$  успешно детектируются.

В том же ключе проведем тестирование некорректного ввода значений параметров  $v_{min}$ ,  $v_{max}$  и  $t_{inc}$ . Для всех перечисленных параметров корректными значениями являются целые числа в диапазоне от 0 до 15.

На рисунке 5.24 представлен вывод программы при вводе пользователем отрицательного значения параметра  $t_{inc}$ .

```

PS C:\Users\user\test_compression> fl-compressor --compress
.\pgm_files\000*.pgm --output test.fl -t_inc -1000
//
// Fast Lossless Compressor version 0.1
//
Incorrect value of -t_inc key
(weight update scaling exponent change interval).
Correct: 0 <= t_inc <= 15.

```

Рисунок 5.25 – Обработка отрицательных значений параметра  $t_{inc}$

Обработка большого значения параметра  $t_{inc}$ , превышающего установленный лимит, представлена на рисунке 5.26.

```

PS C:\Users\user\test_compression> fl-compressor --compress
.\pgm_files\000*.pgm --output test.fl -t_inc 500
//
// Fast Lossless Compressor version 0.1
//
Incorrect value of -t_inc key
(weight update scaling exponent change interval).
Correct: 0 <= t_inc <= 15.

```

Рисунок 5.26 – Обработка большого значения параметра  $t_{inc}$ .

На рисунке 5.27 показан вывод программы при передаче ей

некорректных значений параметра  $v_{min}$ .

```
PS C:\Users\user\test_compression> fl-compressor --compress
.\pgm_files\000*.pgm --output test.fl -nu_min -300
//
// Fast Lossless Compressor version 0.1
//
Incorrect value of -nu_min key
(weight update scaling exponent initial parameter).
Correct: 0 <= nu_min <= 15.
PS C:\Users\user\test_compression> fl-compressor --compress
.\pgm_files\000*.pgm --output test.fl -nu_min 1000
//
// Fast Lossless Compressor version 0.1
//
Incorrect value of -nu_min key
(weight update scaling exponent initial parameter).
Correct: 0 <= nu_min <= 15.
```

Рисунок 5.26 – Обработка нарушения диапазона параметра  $v_{min}$

Аналогично проведем тест передачи некорректных значений  $v_{max}$  параметра. Результаты теста отображены на рисунке 5.27.

```
PS C:\Users\user\test_compression> fl-compressor --compress
.\pgm_files\000*.pgm --output test.fl -nu_max -400
//
// Fast Lossless Compressor version 0.1
//
Incorrect value of -nu_max key
(weight update scaling exponent final parameter).
Correct: 0 <= nu_max <= 15.
PS C:\Users\user\test_compression> fl-compressor --compress
.\pgm_files\000*.pgm --output test.fl -nu_max 700
//
// Fast Lossless Compressor version 0.1
//
Incorrect value of -nu_max key
(weight update scaling exponent final parameter).
Correct: 0 <= nu_max <= 15.
```

Рисунок 5.27 – Обработка нарушения диапазона параметра  $v_{max}$

По результатам проведенных тестов можно заключить, что в целом в программе имеются первичные механизмы детектирования ошибок пользовательского ввода.

## 5.6 Сравнение с результатами универсальных алгоритмов сжатия

Популярные программы-архиваторы, такие как WinZip, WinRAR, 7-Zip предназначены для сжатия практически любого набора файлов и данных, вне зависимости от содержимого, типа и размера. Причем, как и разработанный программный модуль, универсальные архиваторы выполняют сжатие входных данных без потерь по очевидным причинам. Данный факт дает повод к сравнению результатов сжатия тестового набора файлов (см. пункт 5.1) демонстрационной программой с результатами работы программ-архиваторов, использующих различные алгоритмы сжатия.

Для проведения теста были выбраны архиваторы WinRAR и 7-Zip как наиболее популярные и эффективные. Каждый из них имеет в своем составе несколько алгоритмов сжатия, возможность выбора которых предоставляется пользователю в момент создания архива.

Архиватор WinRAR известен тем, что имеет свой собственный формат архива – RAR, а также собственные алгоритмы сжатия, последние версии которых носят названия RAR4 и RAR5.

Архиватор 7-Zip позволяет создавать архивы форматов 7z и zip, при этом доступны следующие алгоритмы сжатия:

- LZMA2;
- LZMA;
- BZip2;
- PPMd;
- Deflate;
- Deflate64.

Разработанный программный модуль реализует алгоритм, который в рекомендациях стандарта CCSDS носит неофициальное название «Fast Lossless» и имеет аббревиатуру FL. Поэтому, для удобства, в последующем будем придерживаться тех же обозначений, а также присваивать расширение .fl для сжатых файлов.

В качестве тестового набора данных, были взяты 20 PGM-файлов с последовательными номерами, которые будут играть роль единого гиперспектрального изображения с 20-ю спектральными каналами.

Сжатие тестового набора данных демонстрационной программой производилось с параметрами по-умолчанию. Команда для осуществления операции имела следующий вид:

```
> fl-compressor --compress .\pgm_files\000*.pgm .\pgm_files\001*.pgm --output FL.fl
```

Результаты работы демонстрационной программы представлены на рисунке 5.28. Вывод программы сигнализирует об успешном сжатии 20 файлов, общим объемом 29,10 Мбайт. Размер сжатого файла достиг 11,85 Мбайт, что соответствует 40,72% от суммарного объема входных

данных, а затраченное время составило не многим более трех секунд.

```
//
// Fast Lossless Compressor version 0.1
//
Opening files...
Start compression 20 files totaling 30515200 bytes (29.10 MB)
...
Prediction...
100% [#####]
Prediction done.
Encoding...
100% [#####]
Encoding done.
Compression done.
Elapsed time is 3.109375 seconds.
Compressed file size is 12425216 bytes (11.85 MB).
Compression ratio is 40.72%.
```

Рисунок 5.28 – Сжатие тестового набора файлов демонстрационной программой

Результаты сжатия тестовых данных программами-архиваторами приведены на рисунке 5.29. Названия файлов соответствуют примененным алгоритмам сжатия. Все алгоритмы применялись со стандартными настройками, выбирался уровень сжатия «Нормальный». Также, проводились замеры времени, затрачиваемого на создание каждого архива.

Имя	Тип	Размер
7z Archive (4)		
BZip2.7z	7z Archive	15 065 КБ
LZMA.7z	7z Archive	16 447 КБ
LZMA2.7z	7z Archive	16 449 КБ
PPMd.7z	7z Archive	15 316 КБ
rar Archive (2)		
RAR4.rar	rar Archive	15 809 КБ
RAR5.rar	rar Archive	16 130 КБ
zip Archive (2)		
Deflate.zip	zip Archive	19 822 КБ
Deflate64.zip	zip Archive	19 702 КБ

Рисунок 5.29 – Результаты сжатия тестового набора файлов программами-архиваторами

Вычислив степень сжатия каждого из архивов, как отношение размера архивного файла к суммарному размеру исходных данных, представим полученные результаты в виде диаграммы (рисунок 5.30). Вертикальная ось соответствует достигнутой степени сжатия исходных данных, в горизонтальной оси располагаются названия примененных алгоритмов.

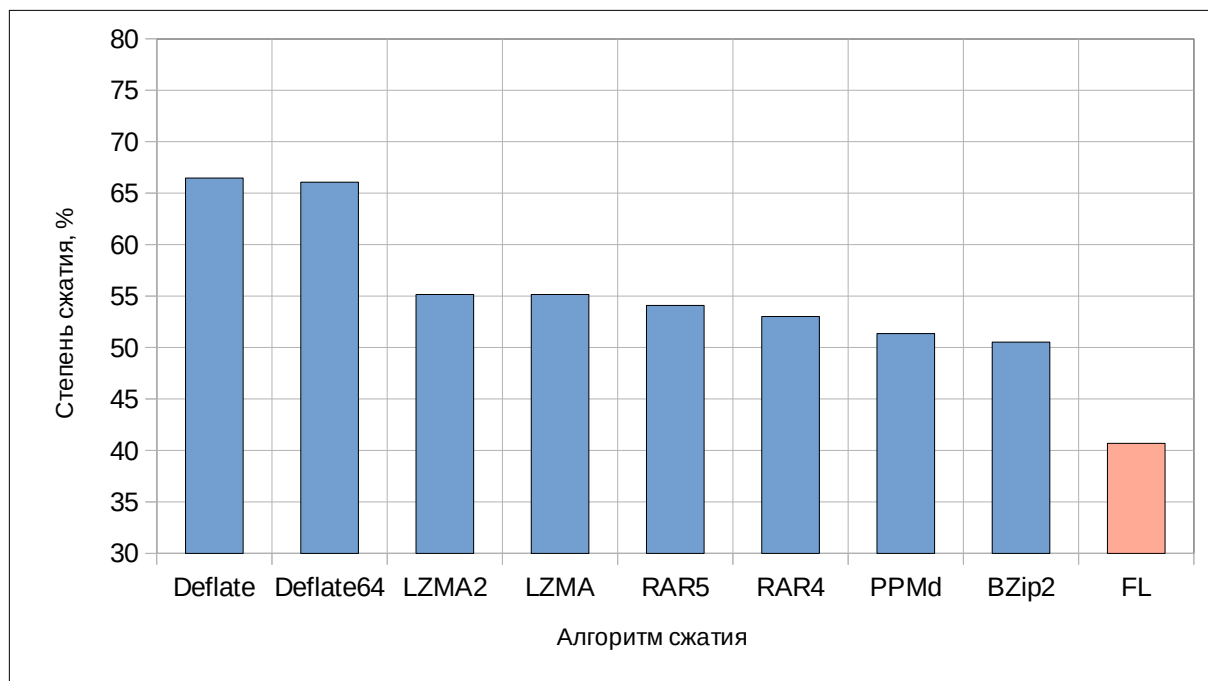


Рисунок 5.30 – Результаты сжатия тестовых данных различными алгоритмами

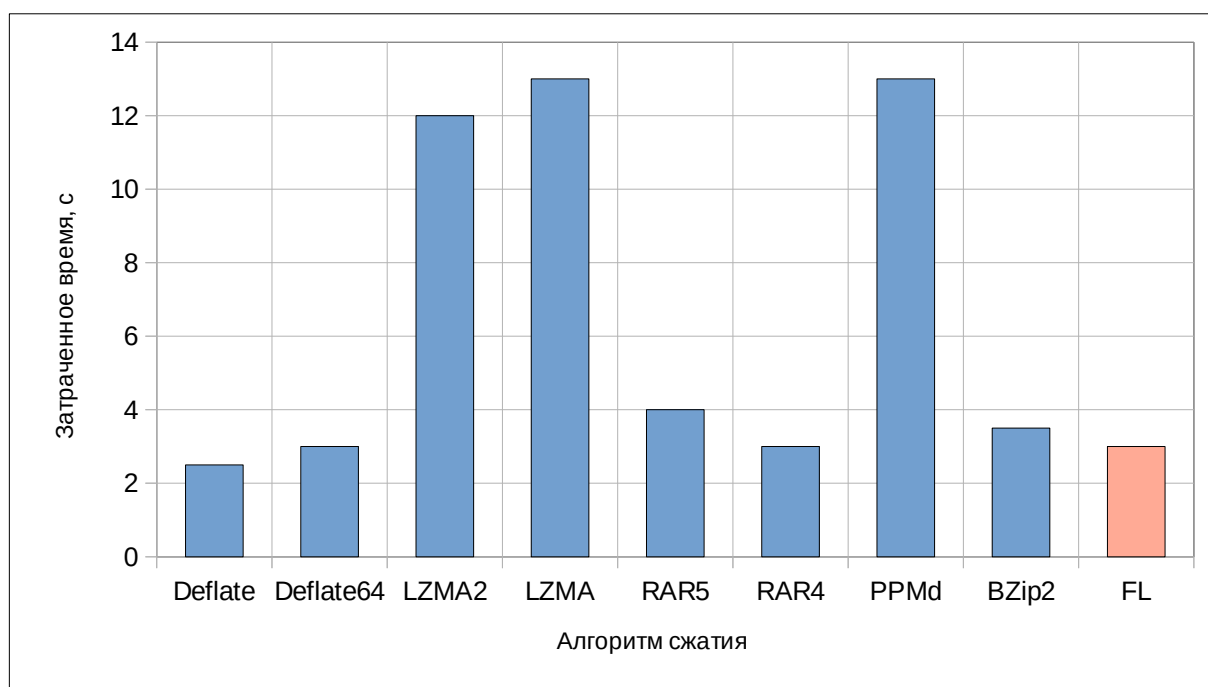


Рисунок 5.31 – Время сжатия тестовых данных различными алгоритмами

Количество времени, затраченное каждым из алгоритмов на выполнение операции, представлено на рисунке 5.31 в виде диаграммы.

Анализируя полученные результаты легко заметить, что реализованный в разработанном программном модуле алгоритм сжатия «Fast Lossless» выполняет сжатие тестовых данных, как минимум, на 10% эффективнее, чем любой из участвовавших в тестировании универсальных алгоритмов. Наихудшие результаты по степени сжатия показали алгоритмы Deflate и Deflate64, применяемые по-умолчанию в большинстве архиваторов при создании архивов формата ZIP.

Что же касается скорости сжатия, то алгоритм «Fast Lossless» оказался среди лидеров и вполне оправдал свое название. Алгоритмы LZMA2, LZMA и RPPMd по скорости оказались самыми медленными, причем время выполнения операции у них в 3-4 раза превысило показатели других тестируемых алгоритмов.

По совокупности показателей, наиболее близким к реализованному в проекте «Fast Lossless» алгоритму при сжатии тестового набора данных оказался алгоритм BZip2.

## 6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

У разработанного программного продукта имеются две категории пользователей:

- разработчики, использующие разработанную библиотеку функций в свои проектах;
- конечные пользователи, которые могут применять написанную в рамках проекта демонстрационную программу для сжатия файлов, имеющих трехмерную структуру.

### 6.1 Руководство разработчика

Данное руководство содержит сведения, необходимые для использования разработанной в рамках проекта библиотеке функций, выполняющих сжатие и восстановление трехмерных данных.

#### 6.1.1 Требования к программному обеспечению

Для использования разработанной библиотеки функций необходимо выполнить следующие требования:

- операционная система, позволяющая выполнять операции с 64-битными целыми числами;
- компилятор языка C, поддерживающий стандарт C99;
- компилятор языка C++, поддерживающий стандарт C++11, необходимый для сборки и запуска модульного тестирования.

#### 6.1.2 Поддерживаемый интерфейс

Набор пользовательских функций реализованной библиотеки представлен ниже.

`1 void runPredictor()` – функция, выполняющая подготовку входных данных к энтропийному кодированию, а также, выполняющая восстановление данных, полученных после работы энтропийного декодера. Имеет следующий набор параметров:

- `uint16_t * in` – указатель на массив исходных данных;
- `uint16_t * out` – указатель на память для сохранения выходных данных;
- `struct ImageMetadata * imageMeta` – указатель на структуру данных, содержащую информацию об изображении;
- `struct PredictorMetadata * predMeta` – указатель на структуру, содержащую конфигурационные параметры адаптивного предсказателя;
- `int opType` – флаг, определяющий тип операции: сжатие или



восстановление.

**2** void encodeGolomb() – функция, выполняющая энтропийное кодирование. Имеет следующий набор параметров:

- uint16\_t \* in – указатель на входные данные;
- uint32\_t \* out – указатель на память для сохранения выходных данных;
- size\_t \* outSize – актуальный размер выходных данных в байтах;
- struct ImageMetadata \* imageMeta – указатель на структуру, содержащую информацию об исходном трехмерном изображении;
- struct EncoderMetadata \* encoderMeta – указатель на структуру, содержащую конфигурационные параметры энтропийного кодера;

**3** void decodeGolomb() – функция, выполняющая восстановление данных, полученных после энтропийного кодирования. Имеет следующие параметры:

- uint32\_t \* in – указатель на массив входных данных;
- uint16\_t \* out – указатель на память для сохранения выходных данных;
- struct ImageMetadata \* imageMeta – указатель на структуру, содержащую информацию об исходном трехмерном изображении;
- struct EncoderMetadata \* encoderMeta – указатель на структуру, содержащую конфигурационные параметры энтропийного кодера.

**4** int loadFromPGM() – функция, выполняющая загрузку изображений из файлов формата PGM. Имеет следующие параметры:

- char \* fileName – имя загружаемого файла;
- uint16\_t \* data[] – адрес указателя, по которому будет находиться «тело» загруженного изображения;
- unsigned \* sizeX – указатель на переменную для сохранения ширины изображения;
- unsigned \* sizeY – указатель на переменную для сохранения высоты изображения;
- unsigned \* maxValue – указатель на переменную для сохранения максимально возможного значения элемента изображения.

**5** int saveToPGM() – функция, выполняющая сохранение изображения в файл формата PGM. Имеет следующие параметры:

- uint16\_t data[] – указатель на массив с «телом» восстановленного изображения;
- unsigned sizeX – ширина изображения;
- unsigned sizeY – высота изображения;
- unsigned maxValue – максимально возможное значение элемента изображения.

**6** int saveCompressedImage() – функция, выполняющая

сохранение сжатого файла. Имеет следующие параметры:

- char \* fileName – имя файла для сохранения;
- void \* data – указатель на данные для сохранения;
- size\_t dataSize – размер данных в байтах;
- struct ImageMetadata \* imageMeta – указатель на структуру, содержащую информацию об исходном изображении;
- struct PredictorMetadata \* predMeta – указатель на структуру, содержащую параметры работы адаптивного предсказателя;
- struct EncoderMetadata \* encoderMeta – указатель, содержащий информацию о параметрах работы энтропийного кодера.

7 int loadCompressedImage() – функция, выполняющая загрузку сжатого изображения. Имеет следующие параметры:

- char \* fileName – имя сжатого файла;
- void \* data[] – адрес указателя, по которому будут доступны кодированные данные;
- size\_t \* dataSize – указатель на переменную для сохранения размера кодированных данных в байтах;
- struct ImageMetadata \* imageMeta – указатель на структуру, в которую будут загружены данные об исходном изображении;
- struct PredictorMetadata \* predMeta – указатель на структуру, в которую будут загружены параметры работы адаптивного предсказателя;
- struct EncoderMetadata \* encoderMeta – указатель на структур, в которую будут загружены параметры работы энтропийного кодера.

## 6.2 Руководство конечного пользователя

Данное руководство посвящено демонстрационной программе, выполняющей сжатие и восстановление набора PGM-файлов, которые трактуются как спектральные слои гиперспектрального изображения. Программа не имеет графического интерфейса. Все взаимодействие с программой осуществляется через терминал посредством запуска исполняемого файла с передачей ему аргументов командной строки.

Программа имеет два режима работы: сжатие и восстановление, которые определяются, соответственно, ключами `--compress` и `--decompress`.

Для режима сжатия формат команды выглядит следующим образом:

```
> .\fl-compressor.exe --compress <список файлов> --output  
<имя выходного файла> [опции]
```

Список файлов можно задавать как простым перечислением, так и с

использованием масок. Допустимы изображения в формате PGM, имеющие одинаковые размеры и количество бит на пиксель. Имя выходного файла может быть любым, но рекомендуется использовать для выходного файла расширение .fl. Вместо просто имени файла можно задать абсолютный или относительный путь до него, но важно убедиться в существовании данного пути, так как программа не может создавать каталоги. В качестве примера приведем следующую команду:

```
> .\fl-compressor.exe --compress 0000.pgm 0001.pgm 0002.pgm  
--output compressed.fl
```

После выполнения данной команды, программа попытается найти в текущем каталоге файлы 0000.pgm, 0002.pgm и 0003.pgm, выполнить их сжатие с параметрами по-умолчанию и сохранить результат в файл compressed.fl. При невозможности открыть какой-либо из указанных файлов, программа выведет соответствующее сообщение об ошибке, но не завершится, а выполнит сжатие успешно загруженных файлов. Пример выполнения указанной команды приведен на рисунке 6.1.

```
//  
// Fast Lossless Compressor version 0.1  
//  
Opening files...  
Start compression 3 files totaling 4577280 bytes (4.37 MB)...  
Prediction...  
100% [#####]  
Prediction done.  
Encoding...  
100% [#####]  
Encoding done.  
Compression done.  
Elapsed time is 0.515625 seconds.  
Compressed file size is 1976068 bytes (1.88 MB).  
Compression ratio is 43.17%.
```

Рисунок 6.1 – Пример работы программы в режиме сжатия

Программа в процессе работы выводит различные информационные сообщения, отражающие ход выполнения операций. Так, из приведенного примера видно, что программа сначала выполняет открытие указанных файлов, потом считает их общее количество и суммарный объем. Далее выполняются внутренние операции предсказания и кодирования. Прогресс в выполнении данных операций выражается в виде постепенно заполняющихся символьных полос. После успешного выполнения операции сжатия, программа выведет информацию о затраченном времени, объеме сжатого

файла и достигнутой степени компрессии. Так, в приведенном на рисунке 6.1 примере, программа выполнила сжатие трех файлов, общий объем которых составил порядка 4,37 Мбайт, на выполнение операции было затрачено примерно полсекунды, объем сжатого файла составил 1,88 Мбайт, что соответствует степени сжатия 43,17%.

Для режима сжатия, через аргументы командной строки можно сообщить программе требуемые параметры работы модуля предсказания и энтропийного кодера путем указания необходимых числовых значений после соответствующих аргументов-ключей. Допустимы следующие ключи:

- ключ `-U` определяет предел длины унарной части кодового слова  $U_{max}$  (допустимы значения от 1 до 20);
- ключ `-P` задает количество спектральных слоев, используемых адаптивным предсказателем  $P$ , (допустимы значения от 0 до 15);
- ключ `-t_inc` определяет скорость адаптации вектора весов ко входному набору данных  $t_{inc}$  (допустимы значения от 0 до 15);
- ключ `-nu_min` устанавливает нижнюю границу значения масштабирующей экспоненты  $v_{min}$  (допустимы значения от 0 до 15);
- ключ `-nu_max` устанавливает верхнюю границу значения масштабирующей экспоненты  $v_{max}$  (допустимы значения от 0 до 15);
- ключ `-Om` определяет ширину диапазона значений весового вектора  $\Omega$  (допустимы значения от 0 до 15);
- ключ `-G` определяет начальное значение величины счетчика элементарно-адаптивного энтропийного кодера  $\Gamma(t)$  (допустимы значения от 1 до 16);
- ключ `-K` устанавливает начальное значение константы инициализации аккумулятора элементарно-адаптивного энтропийного кодера  $\Sigma_z(t)$  (допустимы значения от 0 до 15).

Например, если требуется выполнить сжатие всех PGM-файлов в папке `pgm_files` в файл `compressed.fl` с параметрами  $U_{max}=10$  и  $P=5$ , команда будет выглядеть следующим образом:

```
> .\fl-compressor.exe --compress .\pgm_files\*.pgm --output  
compressed.fl -U 10 -P 5
```

Для режима восстановления формат команды выглядит следующим образом:

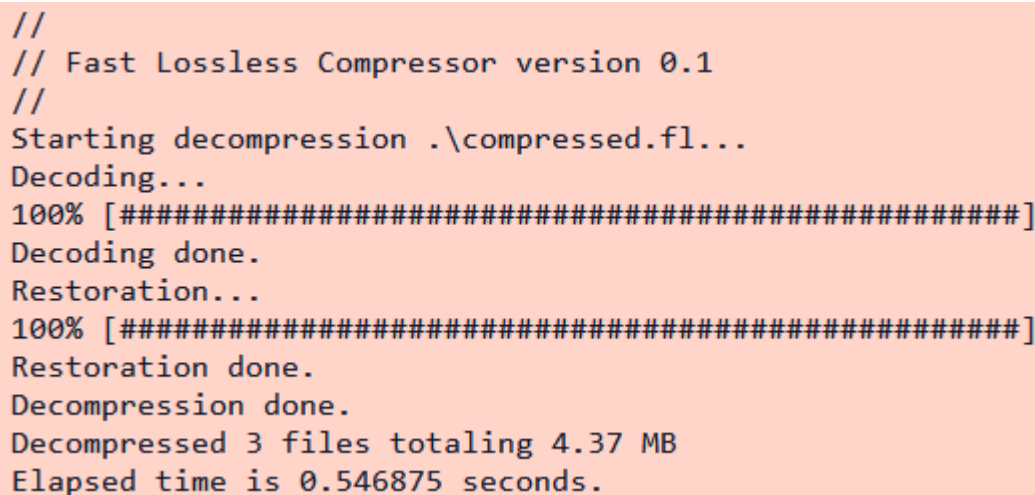
```
> .\fl-compressor.exe --decompress <путь до сжатого файла>  
--output <путь к папке назначения>
```

Программа за один запуск может распаковать лишь один сжатый файл. В папке назначения будут созданы файлы с именем в формате <номер спектрального слоя>.pgm. Например, команда для распаковки файла

compressed.fl в каталог restored будет выглядеть следующим образом:

```
> .\fl-compressor.exe --decompress compressed.fl  
--output .\restored\
```

Результаты выполнения данной команды приведены на рисунке 6.2.



```
//  
// Fast Lossless Compressor version 0.1  
//  
Starting decompression .\compressed.fl...  
Decoding...  
100% [#####]  
Decoding done.  
Restoration...  
100% [#####]  
Restoration done.  
Decompression done.  
Decompressed 3 files totaling 4.37 MB  
Elapsed time is 0.546875 seconds.
```

Рисунок 6.2 – Пример работы программы в режиме восстановления

Вывод программы в режиме восстановления (рисунок 6.2) говорит нам о том, что было успешно выполнено декодирование и восстановление исходных PGM-файлов, игравших роль гиперспектрального изображения. За полсекунды было восстановлено три файла общим объемом 4,37 Мбайт.

Для работы программы в режиме восстановления не существует каких-либо дополнительных ключей-аргументов.

## **7 ЭКОНОМИЧЕСКОЕ            ОБОСНОВАНИЕ            РАЗРАБОТКИ ПРОГРАММНОГО    МОДУЛЯ    СЖАТИЯ    ТРЕХМЕРНЫХ ДАННЫХ**

### **7.1 Характеристика программного модуля сжатия трехмерных данных**

Разрабатываемый программный модуль представляет собой реализацию алгоритма сжатия гиперспектральных и мультиспектральных изображений, основанного на рекомендациях стандарта CCSDS. В виде гиперспектральных и мультиспектральных изображений хранятся данные дистанционного зондирования земли (ДДЗ), снимки орбитальных телескопов, космических исследовательских зондов. Следование рекомендациям стандарта CCSDS при разработке программного модуля означает возможность его широкого применения во всех системах, поддерживающих данный стандарт.

Разрабатываемый программный модуль выполняет следующие функции:

- производит сжатие одного гиперспектрального или мультиспектрального изображения, либо пакета изображений;
- производит восстановление одного изображения, либо пакета изображений;
- при сжатии изображений имеется возможность указать параметры сжатия путем передачи их посредством аргументов командной строки;
- отображает информацию о производимых операциях посредством псевдографического интерфейса.

Разрабатываемый программный модуль предназначен для использования в аэрокосмической отрасли при организации передачи гиперспектральной информации от искусственных спутников Земли, производящих съемку земной поверхности во множестве спектральных диапазонов, зондов, выполняющих съемку и исследование других космических объектов (планет, комет, астероидов). Также разрабатываемый программный модуль может быть использован в системах организации хранения гиперспектральных данных.

К преимуществам реализуемого программного модуля можно отнести:

- сжатие данных без потерь;
- невысокая вычислительная сложность;
- низкие требования к аппаратному обеспечению;
- не имеет зависимостей от сторонних библиотек;
- возможность собрать исполняемый файл практически под любую операционную систему и оборудование, ввиду использования языка C.

## 7.2 Расчет затрат на разработку программного средства организации питания сотрудников

Основная заработная плата исполнителей ( $З_o$ ) рассчитывается по формуле (7.1):

$$З_o = \sum_{i=1}^n T_{\text{чи}} * TP_{\text{чи}} * K_i, \quad (7.1)$$

где  $n$  – количество исполнителей, занятых разработкой;  
 $T_{\text{чи}}$  – часовая заработная плата  $i$ -го исполнителя, руб./ч.;  
 $TP_{\text{чи}}$  – трудоемкость работ  $i$ -го исполнителя, ч.;  
 $K_i$  – коэффициент премирования.

Результаты расчета основной заработной платы исполнителей представлены в таблице 7.1. За норму рабочего времени на 2019 год примем 170 часов в месяц, коэффициент премирования возьмем равным 1,5.

Таблица 7.1 – Расчет основной заработной платы исполнителей

Исполнитель	Месячная заработная плата ( $T_m$ ), руб.	Часовая заработная плата ( $T_{\text{ч}}$ ), руб.	Трудоемкость работ ( $TP$ ), ч.	Коэффициент премий ( $K$ )	Заработная плата ( $З$ ), руб.
Руководитель проекта	2500,00	14,71	40	1,5	7058,82
Инженер- программист	1500,00	8,82	300	1,5	31764,71
Основная заработная плата ( $З_o$ )					38823,53

Величину дополнительной заработной платы исполнителей вычислим по формуле (7.2):

$$З_d = \frac{З_o * H_d}{100}, \quad (7.2)$$

где  $H_d$  – норматив дополнительной заработной платы (10%).

Дополнительная заработная плата составит:

$$З_d = 38823,53 * 10 / 100 = 3882,35 \text{ руб.}$$

Отчисления в фонд социальной защиты населения и на обязательное страхование ( $З_{сз}$ ) определяются в соответствии с действующими законодательными актами по формуле (7.3):

$$З_{сз} = \frac{(З_о + З_д) * Н_{сз}}{100}, \quad (7.3)$$

где  $Н_{сз}$  – норматив отчислений в фонд социальной защиты населения (34,0%) и на обязательное страхование (0,6%), суммарно 34,6%.

Размер отчислений в фонд социальной защиты населения и на обязательное страхование составит:

$$З_{сз} = (38823,53 + 3882,35) * 34,6 / 100 = 14776,24 \text{ руб.}$$

Расходы по статье «Машинное время» ( $P_m$ ) определим по формуле (7.4):

$$P_m = Ц_m * TP_o, \quad (7.4)$$

где  $Ц_m$  – цена одного часа машинного времени, м-ч, 1,00 руб.;

$TP_o$  – общая трудоемкость работ, ч.

Расходы на использование машинного времени составят:

$$P_m = 1,00 * (40 + 300) = 340,00 \text{ руб.}$$

Полная сумма затрат на разработку программного модуля ( $З_p$ ) найдем путем суммирования всех рассчитанных статей затрат:

$$З_p = 38823,53 + 3882,35 + 14776,24 + 340,00 = 57822,12 \text{ руб.}$$

### **7.3 Расчет экономической эффективности реализации на рынке программного модуля сжатия трехмерных данных**

Разрабатываемый программный модуль планируется распространять через сеть Интернет путем продажи заинтересованным организациям лицензий на пользование продуктом сроком на 1 год по цене 1500 рублей. Предполагается, что в среднем в год лицензии на пользование продуктом будут приобретать 25 организаций. Таким образом, чистая прибыль, полученная от реализации программного модуля на рынке ( $П_q$ ) будет рассчитываться по формуле (7.5):

$$П_q = (Ц * N - НДС) * (1 - Н_{п}), \quad (7.5)$$

где  $Ц$  – цена одной лицензии, руб.;

$N$  – ожидаемое количество приобретенных лицензий;

НДС – налог на добавленную стоимость, руб.;



$H_n$  – ставка налога на прибыль.

Сумму налога на добавленную стоимость рассчитаем по формуле (7.6):

$$HДС = \frac{Ц * N * H_{дс}}{(100\% + H_{дс})}, \quad (7.6)$$

где  $H_{дс}$  – ставка налога на добавленную стоимость согласно действующему законодательству, (20%).

Таким образом, величина налога на добавленную стоимость составит:

$$HДС = (1500,00 * 25 * 20) / (100 + 20) = 6250,00 \text{ руб.}$$

Прибыль составит:

$$П_ч = (1500,00 * 25 - 6250,00) * (1 - 0,18) = 25625,00 \text{ руб.}$$

#### **7.4 Расчет показателей эффективности инвестиций в разработку программного модуля сжатия трехмерных данных**

Сравнивая величину годового экономического эффекта в виде прогнозируемой прибыли ( $П_ч$ ) с величиной инвестиций (полной суммы затрат на разработку ( $З_p$ )), можно сделать вывод, что инвестиции не окупятся за один год. Поэтому, для расчета эффективности инвестиций необходимо выполнить расчеты чистого дисконтированного дохода (ЧДД), срока окупаемости ( $T_{ок}$ ) и рентабельности инвестиций ( $P_{и}$ ).

Чистый дисконтированный доход (ЧДД) рассчитывается по формуле (7.7):

$$ЧДД = \sum_{t=1}^n (P_t \alpha_t - Z_t \alpha_t), \quad (7.7)$$

где  $n$  – расчетный период, лет;

$P_t$  – результат (экономический эффект), полученный в году  $t$ , руб.;

$Z_t$  – затраты (инвестиции) в году  $t$ , руб.;

$\alpha_t$  – коэффициент дисконтирования, определяемый по формуле (7.8):

$$\alpha_t = \frac{1}{(1 + E_n)^t}, \quad (7.8)$$

где  $E_n$  – норма дисконта, на расчетный 2019 год равная 0,15;

$t$  – порядковый номер года в расчетном периоде (шаг расчета).

Срок окупаемости проекта – момент, когда суммарный дисконтированный результат (эффект) станет равным или превысит дисконтированную сумму инвестиций. То есть, определяется через какой период времени инвестиционный проект начнет приносить инвестору прибыль.

Рентабельность инвестиций ( $P_{и}$ ) рассчитывается как отношение суммы дисконтированных результатов (эффектов) к осуществленным инвестициям (7.9):

$$P_{и} = \sum_{t=0}^n P_t \alpha_t / \sum_{t=0}^n Z_t \alpha_t. \quad (7.9)$$

Расчет показателей эффективности представлен в таблице 7.2. За нулевой шаг расчета был принят 2019 год.

Таблица 7.2 – Расчет показателей эффективности инвестиций

Показатель	Шаги расчета				
	0	1	2	3	4
Результат					
1. Прирост чистой прибыли, руб.	0,00	25625,00	25625,00	25625,00	25625,00
Затраты (инвестиции)					
2. Инвестиции в разработку, руб.	-57822,12	0,00	0,00	0,00	0,00
3. Всего инвестиций, руб.	-57822,12	0,00	0,00	0,00	0,00
Экономический эффект					
4. Чистый поток наличности (ЧПН), руб.	-57822,12	25625,00	25625,00	25625,00	25625,00
5. ЧПН нарастающим итогом, руб.	-57822,12	-32197,12	-6572,12	19052,88	44677,88
6. Коэффициент дисконтирования	1,000	0,870	0,756	0,658	0,572
7. Дисконтированный ЧПН, руб.	-57822,12	22282,61	19376,18	16848,85	14651,18
8. Чистый дисконтированный доход (ЧДД), руб.	15337,00				
9. Внутренняя норма доходности (ВНД)	28%				
10. Индекс рентабельности	1,27				
11. Срок окупаемости	2 года 4 месяца				

Анализируя таблицу 7.2 можно сделать вывод, что разработка программного модуля сжатия гиперспектральных данных является экономически целесообразной, так как ЧДД является величиной положительной и равен 14651,18 руб., ВНД превышает ставку дисконта, а индекс рентабельности больше 1,0. Инвестиции в разработку окупятся спустя 2 года 4 месяца после выхода продукта на рынок.

## ЗАКЛЮЧЕНИЕ

В рамках выполненного дипломного проекта была разработана библиотека функций, с помощью которых можно выполнять сжатие без потерь и восстановление гиперспектральных, мультиспектральных и иных данных, имеющих трехмерную структуру.

Следуя рекомендациям стандарта CCSDS, были реализованы:

- алгоритм адаптивного предсказателя значений элементов трехмерных изображений;
- элементарно-адаптивный энтропийный кодер;
- элементарно-адаптивный энтропийный декодер;
- структуры для хранения метаданных трехмерного изображения, адаптивного предсказателя и энтропийного кодера.

Для демонстрации возможностей разработанной библиотеки функций была написана простая консольная программа fl-compressor, способная производить сжатие и восстановление трехмерных изображений. Взаимодействие с программой осуществляется через передачу ей соответствующих аргументов командной строки.

Было проведено сравнение результатов сжатия тестовых данных разработанной программой с результатами архивации этих данных популярными программами-архиваторами, использующими различные универсальные алгоритмы сжатия (LZMA, LZMA2, BZip2, PPMd, RAR4, RAR5, Deflate, Deflate64). Сравнение показало, что разработанная программа на 10-15% эффективнее сжимает трехмерные данные, при сохранении высокой скорости работы. Данный факт позволяет надеяться на коммерческий успех разработанного программного модуля и интерес к нему со стороны специализированных организаций.

В целом можно считать, что проект выполнен в полном объеме. Среди дальнейших путей улучшения можно выделить следующие:

- провести оптимизацию вычислений;
- расширить набор поддерживаемых форматов входных данных;
- реализовать графический пользовательский интерфейс.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- [1] Motta, G. Hyperspectral Data Compression / G. Motta, F. Rizzo, James A. Storer. – Springer, 2005.
- [2] Райкунов, Г. Г. Гиперспектральное дистанционное зондирование в геологическом картировании / Под науч. ред. докт. техн. Наук, проф. Г. Г. Райкунова. – М.: ФИЗМАТЛИТ, 2014. – 136 с.
- [3] Панфилов, И. П. Теория электрической связи / И. П. Панфилов, В. Е. Дырда. – М.: Радио и связь, 1991. – 344 с.
- [4] Lossless Multispectral & Hyperspectral Image Compression. Issue 1. Recommendation for Space Data System Standards (Blue Book), CCSDS 123.0-B-1. Washington, D.C.: CCSDS, May 2012.
- [5] Садыхов, Р. Х. Лабораторный практикум по дисциплин. «Цифровая обработка сигналов и изображений» и «Методы и средства обработки изображений» для студ. спец. I-40 02 01 «Вычислительные машины, системы и сети» и I-40 01 01 «Программное обеспечение информационных технологий» всех форм обуч. В 2 ч. Ч. 1 / Р. Х. Садыхов, М. М. Лукашевич. – Мн.: БГУИР, 2006. – 48 с.
- Availability and description of the File Checksum Integrity Verifier utility [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://support.microsoft.com/en-us/help/841290/availability-and-description-of-the-file-checksum-integrity-verifier-u> – Дата доступа: 25.05.2019
- Catch2 [Электронный ресурс]. – Электронные данные. – Режим доступа: <https://github.com/catchorg/Catch2> – Дата доступа: 11.04.2019
- Ватолин, Д. Методы сжатия данных. Устройство архиваторов, сжатие изображений и видео / Д. Ватолин, А. Ратушняк, М. Смирнов, В. Юкин. – М.: ДИАЛОГ-МИФИ, 2003. – 384 с.
- Глецевич, И. И. Вычислительные машины, системы и сети: дипломное проектирование: методическое пособие / И. И. Глецевич, В. А. Прытков, А. С. Сидорович. – Мн.: БГУИР, 2019. – 99 с.
- Гонсалес, Р. Цифровая обработка изображений. Издание 3-е, исправленное и дополненное / Р. Гонсалес, Р. Вудс. – М.: Техносфера, 2012. – 1104 с.
- Горовой, В. Г. Экономическое обоснование проекта по разработке программного обеспечения / В. Г. Горовой, А. В. Грицай, В. А. Пархименко. – Мн.: БГУИР, 2018. – 12 с.
- Прата, С. Язык программирования С. Лекции и упражнения, 5-е издание / С. Прата. – М.: Издательский дом «Вильямс», 2013. – 960 с.
- Сэломон, Д. Сжатие данных, изображений и звука / Д. Сэломон. – М.: Техносфера, 2004. – 368 с.

**ПРИЛОЖЕНИЕ А**  
*(обязательное)*

Текст программы

## Файл fl\_compressor.h

```
#ifndef FL_COMPRESSOR_H
#define FL_COMPRESSOR_H

#include <inttypes.h>

#define PB_LENGTH 50

struct ImageMetadata {
    unsigned userData      : 8;
    unsigned xSize         : 16;
    unsigned ySize         : 16;
    unsigned zSize         : 16;
    unsigned sampleType    : 1;
    unsigned reserved_0    : 2;
    unsigned dynamicRange  : 4;
    unsigned sampleEncodingOrder : 1;
    unsigned subFrameInterleavingDepth : 16;
    unsigned reserved_1    : 2;
    unsigned outputWordSize : 3;
    unsigned entropyCoderType : 1;
    unsigned reserved_2    : 10;
};

struct PredictorMetadata {
    unsigned reserved_0      : 2;
    unsigned predictionBands : 4;
    unsigned predictionMode  : 1;
    unsigned reserved_1     : 1;
    unsigned localSumType    : 1;
    unsigned reserved_2     : 1;
    unsigned registerSize    : 6;
    unsigned weightComponentResolution : 4;
    unsigned wuScalingExpChangeInterval : 4;
    unsigned wuScalingExpInitialParameter : 4;
    unsigned wuScalingExpFinalParameter : 4;
    unsigned reserved_3     : 1;
    unsigned weightInitMethod : 1;
    unsigned weightInitTableFlag : 1;
    unsigned weightInitResolution : 5;
};

struct EncoderMetadata {
    unsigned unaryLengthLimit : 5;
    unsigned rescalingCounterSize : 3;
    unsigned initialCountExponent : 3;
    unsigned accumInitConstant : 4;
    unsigned accumInitTableFlag : 1;
};

enum {
    PREDICTOR_MAP,
    PREDICTOR_RESTORE
};

int getLocalSum(uint16_t * currentBand, int sizeY, int sizeX, int y, int x);
int clip(int val, int val_min, int val_max);
int sgn_plus(int val);
int mod_R(int x, int R);
int d(uint16_t *currentBand, int sizeY, int sizeX, int y, int x);
int dN(uint16_t *currentBand, int sizeY, int sizeX, int y, int x);
int dW(uint16_t *currentBand, int sizeY, int sizeX, int y, int x);
int dNW(uint16_t *currentBand, int sizeY, int sizeX, int y, int x);
void getU(int * U, int P, uint16_t *image, int sizeY, int sizeX, int z, int y, int x);
void weightInitDefault(int * W, int om, int P);
int getPredictedD(int * U, int * W, int size);
int getScalingExp(int D, int Om, int v_min, int v_max, int t, int t_inc, int Nx);
uint16_t getMappedPredictionResidual(int s, int scale_s_pred, int s_min, int s_max);
uint16_t getRestoredValue(int mappedResidual, int scale_s_pred, int s_min, int s_max, int s_mid);
void updateW(int * W, int * U, int size, int e, int ro, int w_min, int w_max);
```

```

void runPredictor(uint16_t *in, uint16_t *out, ImageMetadata * imageMeta, PredictorMetadata *
predMeta, int opType);
size_t getAccum(size_t prevAccum, size_t prevCounter, uint32_t prevResidual, uint32_t gamma);
size_t getCounter(size_t prevCounter, unsigned gamma);
unsigned getCodeWordSize(size_t counter, size_t accum);
void encodeGolomb(uint16_t * in, uint32_t * out, size_t * outSize, ImageMetadata * imageMeta,
EncoderMetadata * encoderMeta);
void decodeGolomb(uint32_t * in, uint16_t * out, ImageMetadata * imageMeta, EncoderMetadata *
encoderMeta);
int loadFromPGM(char *fileName, uint16_t *data[], unsigned * sizeX, unsigned * sizeY, unsigned *
maxValue);
int saveToPGM(char *fileName, uint16_t data[], unsigned sizeX, unsigned sizeY, unsigned
maxValue);
void swapBytes(uint16_t * p, size_t size);
int loadCompressedImage(char * fileName, void ** data, size_t * dataSize,
ImageMetadata * imageMeta, PredictorMetadata * predMeta, EncoderMetadata
* encoderMeta);
int saveCompressedImage(char * fileName, void * data, size_t dataSize,
ImageMetadata * imageMeta, PredictorMetadata * predMeta, EncoderMetadata
* encoderMeta);
void printUsage();

#endif // FL_COMPRESSOR_H

```

## Файл fl\_compressor.c

```

#include "fl_compressor.h"
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <string.h>

int getLocalSum(uint16_t * band, int sizeY, int sizeX, int y, int x) {
    int sNW, sN, sNE, sW;
    if (sizeX <= 0 || sizeY <= 0)
        return -1;
    if (y > 0 && y < sizeY) {
        if (x > 0 && x < sizeX - 1) {
            sNW = band[sizeX * (y - 1) + (x - 1)];
            sN = band[sizeX * (y - 1) + x];
            sNE = band[sizeX * (y - 1) + (x + 1)];
            sW = band[sizeX * y + (x - 1)];
            return sNW + sN + sNE + sW;
        } else if (x == 0) {
            sN = band[sizeX * (y - 1) + x];
            sNE = band[sizeX * (y - 1) + (x + 1)];
            return (sN + sNE) << 1;
        } else if (x == sizeX - 1) {
            sNW = band[sizeX * (y - 1) + (x - 1)];
            sN = band[sizeX * (y - 1) + x];
            sW = band[sizeX * y + (x - 1)];
            return sW + sNW + (sN << 1);
        }
    } else if (y == 0 && x > 0 && x < sizeX) {
        return band[x - 1] << 2;
    }
    return -1;
}

int clip(int val, int val_min, int val_max) {
    int t = val_min > val ? val_min : val;
    return val_max < t ? val_max : t;
}

int sgn_plus(int val) {
    return val < 0 ? -1 : 1;
}

int mod_R(int x, int R) {
    return ((x + (111 << (R - 111))) % (111 << R)) - (111 << (R - 1));
}

```

```

int d(uint16_t * currentBand, int sizeY, int sizeX, int y, int x) {
    return (currentBand[sizeX * y + x] << 2) - getLocalSum(currentBand, sizeY, sizeX, y, x);
}

int dN(uint16_t * currentBand, int sizeY, int sizeX, int y, int x) {
    if(y == 0)
        return 0;
    return (currentBand[sizeX * (y - 1) + x] << 2) - getLocalSum(currentBand, sizeY, sizeX, y,
x);
}

int dW(uint16_t * currentBand, int sizeY, int sizeX, int y, int x) {
    if(y == 0)
        return 0;
    if(x == 0)
        return (currentBand[sizeX * (y - 1) + x] << 2) - getLocalSum(currentBand, sizeY, sizeX,
y, x);
    return (currentBand[sizeX * y + (x - 1)] << 2) - getLocalSum(currentBand, sizeY, sizeX, y,
x);
}

int dNW(uint16_t * currentBand, int sizeY, int sizeX, int y, int x) {
    if(y == 0)
        return 0;
    if(x == 0)
        return (currentBand[sizeX * (y - 1) + x] << 2) - getLocalSum(currentBand, sizeY, sizeX,
y, x);
    return (currentBand[sizeX * (y - 1) + (x - 1)] << 2) - getLocalSum(currentBand, sizeY, sizeX,
y, x);
}

void getU(int * U, int P, uint16_t * image, int sizeY, int sizeX, int z, int y, int x) {
    int bandSize = sizeY * sizeX;
    uint16_t * band = image + bandSize * z;

    U[0] = dN(band, sizeY, sizeX, y, x);
    U[1] = dW(band, sizeY, sizeX, y, x);
    U[2] = dNW(band, sizeY, sizeX, y, x);

    if(z == 0) {
        int val = 0;
        for(int i = 1; i <= P; i++) {
            U[i + 2] = val;
        }
        return;
    }

    for(int i = 1; i <= P; i++) {
        int stepsDown = z - i > 0 ? i : z;
        U[i + 2] = d(band - bandSize * stepsDown, sizeY, sizeX, y, x);
    }
}

void weightInitDefault(int * W, int om, int P) {
    W[0] = W[1] = W[2] = 0;
    if(P <= 0)
        return;
    W[3] = 7 * (1 << om) / 8;
    for (int i = 1; i < P; i++) {
        W[3 + i] = W[3 + i - 1] / 8;
    }
}

int getPredictedD(int * U, int * W, int size) {
    int sum = 0;
    for (int i = 0; i < size; i++) {
        sum += U[i] * W[i];
    }
    return sum;
}

int getScalingExp(int D, int Om, int v_min, int v_max, int t, int t_inc, int Nx) {

```



```

    return clip(v_min + (t - Nx)/t_inc, v_min, v_max) + D - Om;
}

uint16_t getMappedPredictionResidual(int s, int scale_s_pred, int s_min, int s_max) {
    int s_pred = scale_s_pred >> 1;
    int residual = s - s_pred;
    int abs_residual = residual < 0 ? -residual : residual;
    int teta_a = s_pred - s_min;
    int teta_b = s_max - s_pred;
    int teta = teta_a < teta_b ? teta_a : teta_b;
    int k = (scale_s_pred & 1) == 0 ? residual : -residual;

    if(abs_residual > teta)
        return (uint16_t) (abs_residual + teta);
    if (0 <= k && k <= teta)
        return (uint16_t) (abs_residual << 1);
    return (uint16_t) ((abs_residual << 1) - 1);
}

uint16_t getRestoredValue(int mapped_residual, int scale_s_pred, int s_min, int s_max, int s_mid)
{
    int s_pred = scale_s_pred >> 1;
    int residual;
    int teta_a = s_pred - s_min;
    int teta_b = s_max - s_pred;
    int teta = teta_a < teta_b ? teta_a : teta_b;
    int k = ((scale_s_pred + mapped_residual) & 1) == 0 ? 1 : -1;

    if(mapped_residual > teta * 2) {
        residual = (teta - mapped_residual) * sgn_plus(s_pred - s_mid);
    } else {
        residual = ((mapped_residual + 1) >> 1) * k;
    }

    return (uint16_t) (residual + s_pred);
}

void updateW(int * W, int * U, int size, int e, int ro, int w_min, int w_max) {
    int sign = sgn_plus(e);
    int f = ro < 0 ? (1 << -ro) : (1 << ro);
    if(ro < 0) {
        for(int i = 0; i < size; i++) {
            W[i] = clip(W[i] + ((sign * (U[i] * f) + 1) / 2), w_min, w_max);
        }
    } else {
        for(int i = 0; i < size; i++) {
            W[i] = clip(W[i] + ((sign * (U[i] / f) + 1) / 2), w_min, w_max);
        }
    }
}

void runPredictor(uint16_t * in, uint16_t * out, ImageMetadata * imageMeta, PredictorMetadata *
predMeta, int opType) {
    int sizeX = imageMeta->xSize;
    int sizeY = imageMeta->ySize;
    int sizeZ = imageMeta->zSize;
    int D = imageMeta->dynamicRange;
    D = D == 0 ? 16 : D;

    int P = predMeta->predictionBands;
    int R = predMeta->registerSize;
    int Om = predMeta->weightComponentResolution + 4;
    int v_min = predMeta->wuScalingExpInitialParameter - 6;
    int v_max = predMeta->wuScalingExpFinalParameter - 6;
    int t_inc = 1 << (predMeta->wuScalingExpChangeInterval + 4);
    int w_min = -(1 << (Om + 2));
    int w_max = (1 << (Om + 2)) - 1;
    uint16_t s_min = 0;
    uint16_t s_max = (uint16_t)((1L << D) - 1);
    uint16_t s_mid = (uint16_t)((1L << (D - 1)) - 1);

```

```

int bandSize = sizeY * sizeX;

uint16_t * curBandIn = in;
uint16_t * curBandOut = out;
uint16_t * predBase = (opType == PREDICTOR_MAP) ? in : out;
uint16_t * curPredBase = predBase;

int uwSize = 3 + P;

int * msU = (int *)malloc(sizeX * sizeY * uwSize * sizeof(int));
int * msW = (int *)malloc(sizeX * sizeY * uwSize * sizeof(int));

char progressBar[PB_LENGTH + 1] = {0};
int progress = 0;
char pbFormat[100];
sprintf(pbFormat, "%s%d%s", "\r%3ld%% [%-", PB_LENGTH, "s]");
memset(progressBar, '-', PB_LENGTH);

if(opType == PREDICTOR_MAP)
    printf("Prediction...\n");
else
    printf("Restoration...\n");

weightInitDefault(msW, Om, P);

for(int i = 1; i < bandSize; i++)
    memcpy(msW + uwSize * i, msW, (size_t)uwSize * sizeof(int));

for(int z = 0; z < sizeZ; z++) {
    int * curU = msU + uwSize;
    int * curW = msW + uwSize;
    int scale_s_pred;

    if(P > 0 && z > 0)
        scale_s_pred = *(curPredBase - bandSize) * 2;
    else
        scale_s_pred = s_mid * 2;

    if(opType == PREDICTOR_MAP)
        curBandOut[0] = getMappedPredictionResidual(curBandIn[0], scale_s_pred, s_min,
s_max);
    else
        curBandOut[0] = getRestoredValue(curBandIn[0], scale_s_pred, s_min, s_max, s_mid);

    int t = 1;
    for(int y = 0; y < sizeY; y++) {
        for(int x = (y == 0 ? 1 : 0); x < sizeX; x++) {
            int local_sum = getLocalSum(curPredBase, sizeY, sizeX, y, x);
            getU(curU, P, predBase, sizeY, sizeX, z, y, x);
            int d_pred = getPredictedD(curU, curW, uwSize);
            int a = d_pred + ((local_sum - ((int)s_mid << 2)) << Om);
            int m = mod_R(a, R);
            int val = m >> (Om + 1);
            scale_s_pred = clip(val + 2 * s_mid + 1, 2 * s_min, 2 * s_max + 1);
            if(opType == PREDICTOR_MAP)
                curBandOut[t] = getMappedPredictionResidual(curBandIn[t], scale_s_pred,
s_min, s_max);
            else
                curBandOut[t] = getRestoredValue(curBandIn[t], scale_s_pred, s_min, s_max,
s_mid);

            int e = 2 * curPredBase[t] - scale_s_pred;
            int ro = getScalingExp(D, Om, v_min, v_max, t, t_inc, sizeX);
            updateW(curW, curU, uwSize, e, ro, w_min, w_max);
            curU += uwSize;
            curW += uwSize;
            t++;
        }
    }
    curBandIn += bandSize;
    curBandOut += bandSize;
    curPredBase += bandSize;
}

```

```

        progress = (z + 1) * (PB_LENGTH - 1) / sizeZ;
        memset(progressBar, '#', (size_t)progress + 1);
        printf(pbFormat, (z + 1) * 100 / sizeZ, progressBar);
    }

    if(msU)
        free(msU);
    if(msW)
        free(msW);

    memset(progressBar, '#', PB_LENGTH);
    printf(pbFormat, 100, progressBar);

    if(opType == PREDICTOR_MAP)
        printf("\nPrediction done.\n");
    else
        printf("\nRestoration done.\n");
}

size_t getAccum(size_t prevAccum, size_t prevCounter, uint32_t prevResidual, uint32_t gamma) {

    if(prevCounter < (1 << gamma) - 1)
        return prevAccum + (size_t)prevResidual;

    return (prevAccum + (size_t)prevResidual + 1) / 2;
}

size_t getCounter(size_t prevCounter, unsigned gamma) {
    if(prevCounter < (1 << gamma) - 1)
        return prevCounter + 1;
    return (prevCounter + 1) / 2;
}

unsigned getCodeWordSize(size_t counter, size_t accum) {
    size_t right = accum + ((49 * counter) >> 7);
    size_t left = counter;
    unsigned k = 0;

    do {
        left <= 1;
        k++;
    } while(left <= right);

    return k - 1;
}

void encodeGolomb(uint16_t * in, uint32_t * out, size_t * outSize, ImageMetadata * imageMeta,
EncoderMetadata * encoderMeta) {

    uint32_t sizeX = imageMeta->xSize;
    uint32_t sizeY = imageMeta->ySize;
    uint32_t sizeZ = imageMeta->zSize;
    int D = imageMeta->dynamicRange;
    D = D == 0 ? 16 : D;

    uint32_t bandSize = sizeY * sizeX;

    uint16_t * curBandIn = in;
    uint32_t * curBandOut = out;

    unsigned U_max = encoderMeta->unaryLengthLimit;
    unsigned gamma_0 = encoderMeta->accumInitConstant;
    unsigned gamma = encoderMeta->rescalingCounterSize + 4;
    unsigned k = encoderMeta->accumInitConstant;
    size_t counter = 1 << gamma_0;
    size_t accum = (3 * (1 << (k + 6)) - 49) * counter / (1 << 7);

    char progressBar[PB_LENGTH + 1] = {0};
    int progress = 0;

```

```

char pbFormat[100];
sprintf(pbFormat, "%s%d%s", "\r%3ld%% [%-", PB_LENGTH, "s]");
memset(progressBar, '-', PB_LENGTH);
printf("Encoding...\n");

uint64_t buffer = 0;
uint32_t bitCounter = 0;

for (uint32_t z = 0; z < sizeZ; z++) {
    buffer <<= D;
    buffer += (uint16_t)curBandIn[0];
    bitCounter += D;
    for(uint32_t i = 1; i < bandSize; i++) {
        accum = getAccum(accum, counter, curBandIn[i - 1], gamma);
        counter = getCounter(counter, gamma);
        k = getCodeWordSize(counter, accum);
        div_t d = div((int32_t)curBandIn[i], (1 << k));
        uint32_t u = (uint32_t)d.quot;
        u = (u < U_max) ? u : U_max;
        buffer <<= u;
        bitCounter += u;
        if(u < U_max) {
            buffer <<= 1;
            buffer += 1;
            bitCounter +=1;
            buffer <<= k;
            bitCounter += k;
            buffer += (uint64_t)d.rem;
        } else {
            buffer <<= D;
            bitCounter += D;
            buffer += (uint16_t)curBandIn[i];
        }
        if(bitCounter >= 32) {
            bitCounter -= 32;
            uint32_t val = (uint32_t)(buffer >> bitCounter);
            *curBandOut = val;
            curBandOut++;
        }
    }
    curBandIn += bandSize;
    progress = (z + 1) * (PB_LENGTH - 1) / sizeZ;
    memset(progressBar, '#', (size_t)progress + 1);
    printf(pbFormat, (z + 1) * 100 / sizeZ, progressBar);
}
if(bitCounter > 0) {
    uint32_t val = (uint32_t)(buffer << (32 - bitCounter));
    *curBandOut = val;
    curBandOut++;
}

*outSize = (curBandOut - out) * sizeof (uint32_t);

memset(progressBar, '#', PB_LENGTH);
printf(pbFormat, 100, progressBar);
printf("\nEncoding done.\n");
}

```

```

void decodeGolomb(uint32_t * in, uint16_t * out, ImageMetadata * imageMeta, EncoderMetadata *
encoderMeta) {

```

```

    uint32_t sizeX = imageMeta->xSize;
    uint32_t sizeY = imageMeta->ySize;
    uint32_t sizeZ = imageMeta->zSize;
    int D = imageMeta->dynamicRange;
    D = D == 0 ? 16 : D;

```

```

    uint32_t bandSize = sizeY * sizeX;

```

```

    uint32_t * curBandIn = in;
    uint16_t * curBandOut = out;

```

```

    unsigned U_max = encoderMeta->unaryLengthLimit;

```

```

unsigned gamma_0 = encoderMeta->accumInitConstant;
unsigned gamma = encoderMeta->rescalingCounterSize + 4;
unsigned k = encoderMeta->accumInitConstant;
size_t counter = 1 << gamma_0;
size_t accum = (3 * (1 << (k + 6)) - 49) * counter / (1 << 7);

char progressBar[PB_LENGTH + 1] = {0};
int progress = 0;
char pbFormat[100];
sprintf(pbFormat, "%s%d%s", "\r%3ld%% [%-", PB_LENGTH, "s]");
memset(progressBar, '-', PB_LENGTH);
printf("Decoding...\n");

uint64_t buffer = 0;
uint64_t lastBit = 1ull << 63;
uint32_t bitCounter = 0;
uint32_t sampleCounter = 0;
uint32_t bandsCounter = 0;
uint16_t val = 0;

buffer += *curBandIn;
buffer <<= 32;
curBandIn++;
buffer += *curBandIn;
curBandIn++;

while(bandsCounter < sizeZ) {
    val = (uint16_t)(buffer >> (64 - D));
    buffer <<= D;
    bitCounter += D;

    curBandOut[0] = val;
    sampleCounter = 1;

    while(sampleCounter < bandSize) {

        accum = getAccum(accum, counter, val, gamma);
        counter = getCounter(counter, gamma);
        k = getCodeWordSize(counter, accum);

        uint16_t u = 0;
        while((buffer & lastBit) == 0ull) {
            u++;
            buffer <<= 1;
            bitCounter++;
            if(u == U_max)
                break;
        }
        if(u == U_max) {
            val = (uint16_t)(buffer >> (64 - D));
            buffer <<= D;
            bitCounter += D;
        } else {
            val = (uint16_t)(u << k);
            buffer <<= 1;
            bitCounter++;
            if(k > 0) {
                val += (uint16_t)(buffer >> (64 - k));
                buffer <<= k;
                bitCounter += k;
            }
        }
        if(bitCounter >= 32) {
            bitCounter -= 32;
            buffer >>= bitCounter;
            buffer += *curBandIn;
            buffer <<= bitCounter;
            curBandIn++;
        }

        curBandOut[sampleCounter] = val;
        sampleCounter++;
    }
}

```

```

        bandsCounter++;
        curBandOut += bandSize;

        progress = bandsCounter * (PB_LENGTH - 1) / sizeZ;
        memset(progressBar, '#', (size_t)progress + 1);
        printf(pbFormat, bandsCounter * 100 / sizeZ, progressBar);
    }

    memset(progressBar, '#', PB_LENGTH);
    printf(pbFormat, 100, progressBar);
    printf("\nDecoding done.\n");
}

void swopBytes(uint16_t * p, size_t size) {
    for(size_t i = 0; i < size; i++) {
        uint16_t tmp = p[i] >> 8;
        p[i] = (uint16_t)((p[i] << 8) + tmp);
    }
}

int loadFromPGM(char *fileName, uint16_t *data[], unsigned * sizeX, unsigned * sizeY, unsigned *
maxValue) {
    FILE * fp;
    char buffer[1024];

    fp = fopen(fileName, "rb");
    if(!fp)
        return -1;

    fscanf(fp, "%s", buffer);
    if(strcmp(buffer, "P5")) {
        printf("File %s has an invalid format. Must be PGM.\n", fileName);
        return -1;
    }
    if(fscanf(fp, "%u%u%u", sizeX, sizeY, maxValue) != 3)
        return -1;

    size_t size = *sizeX * *sizeY;
    int bitsPerChannel = int(log(*maxValue + 1.0) / log(2.0));

    if(!(8 < bitsPerChannel && bitsPerChannel <= 16))
        return -1;

    int ch = fgetc(fp);
    while(ch != '\n')
        ch = fgetc(fp);

    *data = (uint16_t *)malloc(size * sizeof (uint16_t));
    if(!(*data))
        return -1;

    if(fread(*data, sizeof(uint16_t), size, fp) != size) {
        free(*data);
        return -1;
    }
    fclose(fp);

    swopBytes(*data, size);

    return 0;
}

int saveToPGM(char *fileName, uint16_t data[], unsigned sizeX, unsigned sizeY, unsigned maxValue)
{
    FILE * fp;
    fp = fopen(fileName, "wb");
    if(!fp)
        return -1;
    fprintf(fp, "P5\n%u %u\n%u\n", sizeX, sizeY, maxValue);
}

```

```

size_t size = sizeX * sizeY;

uint16_t * p = (uint16_t *)malloc(size * sizeof(uint16_t));
if(!p) {
    fclose(fp);
    return -1;
}
memcpy(p, data, size * sizeof(uint16_t));
swopBytes(p, size);

if(fwrite(p, sizeof(uint16_t), size, fp) != size) {
    if(p)
        free(p);
    fclose(fp);
    return -1;
}

if(p)
    free(p);
fclose(fp);

return 0;
}

int saveCompressedImage(char * fileName, void * data, size_t dataSize,
                        ImageMetadata * imageMeta, PredictorMetadata * predMeta, EncoderMetadata
* encoderMeta) {

    FILE * fp;
    fp = fopen(fileName, "wb");
    if(!fp)
        return -1;

    size_t i_size = sizeof (ImageMetadata);
    size_t p_size = sizeof (PredictorMetadata);
    size_t e_size = sizeof (EncoderMetadata);

    size_t total = 0;

    total += fwrite(imageMeta , i_size , 1, fp);
    total += fwrite(predMeta , p_size , 1, fp);
    total += fwrite(encoderMeta, e_size , 1, fp);
    total += fwrite(data      , dataSize, 1, fp);

    fclose(fp);

    if(total != 4)
        return -1;

    return 0;
}

int loadCompressedImage(char * fileName, void ** data, size_t * dataSize,
                        ImageMetadata * imageMeta, PredictorMetadata * predMeta, EncoderMetadata
* encoderMeta) {

    FILE * fp;
    fp = fopen(fileName, "rb");
    if(!fp)
        return -1;

    size_t i_size = sizeof (ImageMetadata);
    size_t p_size = sizeof (PredictorMetadata);
    size_t e_size = sizeof (EncoderMetadata);

    size_t headerSize = i_size + p_size + e_size;

    size_t total = 0;

```

```

total += fread(imageMeta , i_size, 1, fp);
total += fread(predMeta , p_size, 1, fp);
total += fread(encoderMeta, e_size, 1, fp);

if(total != 3) {
    fclose(fp);
    return -1;
}

fseek(fp, 0L, SEEK_END);
*dataSize = (size_t)ftell(fp) - headerSize;
fseek(fp, (long)headerSize, SEEK_SET);

*data = malloc(*dataSize);
if(!(*data)) {
    fclose(fp);
    return -1;
}

total += fread(*data, *dataSize, 1, fp);
fclose(fp);

if(total != 4)
{
    free(*data);
    return -1;
}

return 0;
}

void printUsage()
{
    printf("fl-compressor.exe <--compress | --decompress> <file1 file2 ...> --output <file_name> [options]\n");
}

```

## Файл main.c

```

#include "fl_compressor.h"
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <time.h>
#define MAX_FILES 500
#define MAX_FILE_NAME LENGHT 128

int main(int argc, char* argv[])
{
    ImageMetadata imageMeta;
    memset(&imageMeta, 0, sizeof (ImageMetadata));
    imageMeta.userData = 0; // : 8;
    imageMeta.xSize = 0; // : 16;
    imageMeta.ySize = 0; // : 16;
    imageMeta.zSize = 0; // : 16;
    imageMeta.sampleType = 0; // : 1;
    imageMeta.reserved_0 = 0; // : 2;
    imageMeta.dynamicRange = 0; // : 4;
    imageMeta.sampleEncodingOrder = 0; // : 1;
    imageMeta.subFrameInterleavingDepth = 0; // : 16;
    imageMeta.reserved_1 = 0; // : 2;
    imageMeta.outputWordSize = 0; // : 3;
    imageMeta.entropyCoderType = 0; // : 1;
    imageMeta.reserved_2 = 0; // : 10;

    PredictorMetadata predMeta;
    memset(&predMeta, 0, sizeof (PredictorMetadata));
    predMeta.reserved_0 = 0; // : 2;
    predMeta.predictionBands = 2; // : 4;
    predMeta.predictionMode = 0; // : 1;
}

```



```

predMeta.reserved_1      = 0;  // : 1;
predMeta.localSumType    = 0;  // : 1;
predMeta.reserved_2      = 0;  // : 1;
predMeta.registerSize    = 32; // : 6;
predMeta.weightComponentResolution = 0; // : 4;
predMeta.wuScalingExpChangeInterval = 0; // : 4;
predMeta.wuScalingExpInitialParameter = 0; // : 4;
predMeta.wuScalingExpFinalParameter = 15; // : 4;
predMeta.reserved_3      = 0;  // : 1;
predMeta.weightInitMethod = 0;  // : 1;
predMeta.weightInitTableFlag = 0; // : 1;
predMeta.weightInitResolution = 0; // : 5;

EncoderMetadata encoderMeta;
memset(&encoderMeta, 0, sizeof (EncoderMetadata));
encoderMeta.unaryLengthLimit = 8;  // : 5;
encoderMeta.rescalingCounterSize = 1; // : 3;
encoderMeta.initialCountExponent = 1; // : 3;
encoderMeta.accumInitConstant = 2;  // : 4;
encoderMeta.accumInitTableFlag = 0; // : 1;
int compress = 1;
char outFileName[MAX_FILE_NAME_LENGTH] = "output";
char * imageNames[MAX_FILES] = {NULL};
unsigned imageCounter = 0;
printf( "//\n"
        "// Fast Lossless Compressor version 0.1\n"
        "//\n");
if (argc == 1) {
    printUsage();
    return -1;
}
for (int i = 1; i < argc; ++i) {
    if (!strcmp(argv[i], "--help") || !strcmp(argv[i], "/?")) {
        printUsage();
        return -1;
    }
    else if (!strcmp(argv[i], "--compress")) {
        compress = 1;
    }
    else if (!strcmp(argv[i], "--decompress")) {
        compress = 0;
    }
    else if (!strcmp(argv[i], "--output")) {
        strncpy(outFileName, argv[i + 1], MAX_FILE_NAME_LENGTH - 1);
        i++;
    }
    else if (!strcmp(argv[i], "-U")) {
        int U = atoi(argv[i + 1]);
        if(U <= 0 || U > 20) {
            printf("Incorrect value of -U key (unary limit).\n"
                   "Correct: 0 < U <= 20\n");
            printUsage();
            return -1;
        }
        encoderMeta.unaryLengthLimit = (unsigned)U;
        i++;
    }
    else if (!strcmp(argv[i], "-K")) {
        int K = atoi(argv[i + 1]);
        if(K < 0 || K > 15) {
            printf("Incorrect value of -K key (accumulator init constant).\n"
                   "Correct: 0 <= K <= 15.\n");
            printUsage();
            return -1;
        }
        encoderMeta.accumInitConstant = (unsigned)K;
        i++;
    }
    else if (!strcmp(argv[i], "-G")) {
        int G = atoi(argv[i + 1]);
        if(G < 1 || G > 16) {
            printf("Incorrect value of -G key (init count exponent).\n"
                   "Correct: 1 <= G <= 16.\n");
            printUsage();
            return -1;
        }
        encoderMeta.initialCountExponent = (unsigned)(G == 16 ? 0 : G);
        i++;
    }
}

```

```

    }
    else if (!strcmp(argv[i], "-P")) {
        int P = atoi(argv[i + 1]);
        if(P < 0 || P > 15) {
            printf("Incorrect value of -P key \n"
                "(count of previous band for prediction)).\n"
                "Correct: 0 <= P <= 15.\n");
            printUsage();
            return -1;
        }
        predMeta.predictionBands = (unsigned)P;
        i++;
    }
    else if (!strcmp(argv[i], "-t_inc")) {
        int t_inc = atoi(argv[i + 1]);
        if(t_inc < 0 || t_inc > 15) {
            printf("Incorrect value of -t_inc key\n"
                "(weight update scaling exponent change interval).\n"
                "Correct: 0 <= t_inc <= 15.\n");
            printUsage();
            return -1;
        }
        predMeta.wuScalingExpChangeInterval = (unsigned)t_inc;
        i++;
    }
    else if (!strcmp(argv[i], "-nu_min")) {
        int nu_min = atoi(argv[i + 1]);
        if(nu_min < 0 || nu_min > 15) {
            printf("Incorrect value of -nu_min key\n"
                "(weight update scaling exponent initial parameter).\n"
                "Correct: 0 <= nu_min <= 15.\n");
            printUsage();
            return -1;
        }
        predMeta.wuScalingExpInitialParameter = (unsigned)nu_min;
        i++;
    }
    else if (!strcmp(argv[i], "-nu_max")) {
        int nu_max = atoi(argv[i + 1]);
        if(nu_max < 0 || nu_max > 15) {
            printf("Incorrect value of -nu_max key\n"
                "(weight update scaling exponent final parameter).\n"
                "Correct: 0 <= nu_max <= 15.\n");
            printUsage();
            return -1;
        }
        predMeta.wuScalingExpFinalParameter = (unsigned)nu_max;
        i++;
    }
    else if (!strcmp(argv[i], "-Om")) {
        int Om = atoi(argv[i + 1]);
        if(Om < 0 || Om > 15) {
            printf("Incorrect value of -Om key (weight component resolution)).\n"
                "Correct: 0 <= Om <= 15.\n");
            printUsage();
            return -1;
        }
        predMeta.weightComponentResolution = (unsigned)Om;
        i++;
    }
    else {
        imageNames[imageCounter] = argv[i];
        imageCounter++;
    }
}
if(imageCounter == 0) {
    printUsage();
    return -1;
}
uint16_t * images[MAX_FILES] = {NULL};
uint16_t * data = NULL;
uint16_t * mappedResiduals = NULL;
uint32_t * encodedData = NULL;
unsigned loadCounter = 0;
unsigned sizeX = 0;
unsigned sizeY = 0;
unsigned sizeZ = 0;
unsigned maxValue = 0;

```

```

struct timespec start, stop;
if(compress) {
    clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &start);
    printf("Opening files...\n");
    for (unsigned i = 0; i < imageCounter; i++) {
        int res = loadFromPGM(imageNames[i], &images[loadCounter],
                             &sizeX, &sizeY, &maxValue);

        if(res != 0) {
            printf("Cannot load %s file.\n", imageNames[i]);
            continue;
        }
        loadCounter++;
    }
    if(loadCounter == 0) {
        printf("No files has been uploaded.\nCompression failed.");
        return -1;
    }
    size_t bandSize = sizeX * sizeY * sizeof (uint16_t);
    size_t size = loadCounter * bandSize;
    data = (uint16_t *)malloc(size);
    if(!data) {
        printf("Cannot allocate %lld MB from memory.\n", size >> 20);
        return -1;
    }
    for (unsigned i = 0; i < loadCounter; i++) {
        memcpy((uint8_t *)data + bandSize * i, images[i], bandSize);
        free(images[i]);
    }
    printf("Start compression %d files totaling %lld bytes (%.2lf MB)...\n",
          loadCounter, size, (double) size / (1 << 20));
    imageMeta.xSize = sizeX;
    imageMeta.ySize = sizeY;
    imageMeta.zSize = loadCounter;
    unsigned d = (unsigned)log2(maxValue + 1);
    imageMeta.dynamicRange = d == 16 ? 0 : d;
    mappedResiduals = (uint16_t *)malloc(size);
    if(!mappedResiduals) {
        printf("Cannot allocate %lld MB from memory.\n", size >> 20);
        return -1;
    }
    runPredictor(data, mappedResiduals, &imageMeta, &predMeta, PREDICTOR_MAP);
    free(data);
    encodedData = (uint32_t *)malloc(size);
    if(!encodedData) {
        printf("Cannot allocate %lld MB from memory.\n", size >> 20);
        return -1;
    }
    size_t outSize = 0;
    encodeGolomb(mappedResiduals, encodedData, &outSize, &imageMeta, &encoderMeta);
    free(mappedResiduals);
    saveCompressedImage(outFileName, encodedData, outSize, &imageMeta, &predMeta,
&encoderMeta);
    free(encodedData);
    clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &stop);
    struct timespec elapsed_time;
    elapsed_time.tv_sec = stop.tv_sec - start.tv_sec;
    elapsed_time.tv_nsec = stop.tv_nsec - start.tv_nsec;
    printf("Compression done.\n");
    printf("Elapsed time is %d.%d seconds.\n", elapsed_time.tv_sec, elapsed_time.tv_nsec /
1000);
    outSize += sizeof (ImageMetadata) + sizeof (PredictorMetadata) + sizeof
(EncoderMetadata);
    printf("Compressed file size is %lld bytes (%.2lf MB).\n",
          outSize, (double)outSize / (1 << 20));
    printf("Compression ratio is %.2lf%%.\n", (double)outSize * 100 / size);
} else {
    for (unsigned i = 0; i < imageCounter; i++) {
        clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &start);
        size_t dataSize = 0;
        printf("Starting decompression %s...\n", imageNames[i]);
        int res = loadCompressedImage(imageNames[i], (void *)&encodedData, &dataSize,
&imageMeta, &predMeta, &encoderMeta);

        if(res != 0) {
            printf("Cannot load %s file.\n", imageNames[i]);
            continue;
        }
        sizeX = imageMeta.xSize;
        sizeY = imageMeta.ySize;

```

```

sizeZ = imageMeta.zSize;
maxValue = (1 << (imageMeta.dynamicRange == 0 ? 16 : imageMeta.dynamicRange)) - 1;
size_t bandTotal = sizeX * sizeY;
size_t size = sizeZ * bandTotal * sizeof (uint16_t);
mappedResiduals = (uint16_t *)malloc(size);
if(!mappedResiduals) {
    printf("Cannot allocate %lld MB from memory.\n", size >> 20);
    return -1;
}
decodeGolomb(encodedData, mappedResiduals, &imageMeta, &encoderMeta);
free(encodedData);
data = (uint16_t *)malloc(size);
if(!data) {
    printf("Cannot allocate %lld MB from memory.\n", size >> 20);
    return -1;
}
runPredictor(mappedResiduals, data, &imageMeta, &predMeta, PREDICTOR_RESTORE);
free(mappedResiduals);
char buffer[1024] = {0};
for(unsigned j = 0; j < sizeZ; j++) {
    sprintf(buffer, "%s%04d.pgm", outFileNames, j);
    saveToPGM(buffer, data + bandTotal * j, sizeX, sizeY, maxValue);
}
free(data);

clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &stop);
struct timespec elapsed_time;
elapsed_time.tv_sec = stop.tv_sec - start.tv_sec;
elapsed_time.tv_nsec = stop.tv_nsec - start.tv_nsec;

printf("Decompression done.\n");
printf("Decompressed %d files totaling %.2lf MB\n", sizeZ, (double)size / (1 << 20));
printf("Elapsed time is %d.%d seconds.\n", elapsed_time.tv_sec,
elapsed_time.tv_nsec / 1000);
}
}
return 0;
}

```

## Файл main.cpp

```

#define CATCH_CONFIG_MAIN // This tells Catch to provide a main() - only do this in one cpp file
#include "catch.hpp"

```

## Файл unit\_tests.cpp

```

#include "catch.hpp"
#include "fl_compressor.h"
TEST_CASE( "getLocalSum", "[local_sum]" ) {
    uint16_t ms[] = {
        7,    2,    6,    7,    5,    1,    8,
        5,    4,    9,    9,    9,    6,    3,
        5,    2,    6,    5,    6,    1,    7,
        2,    8,    8,    8,    7,    6,    5,
        8,    1,    9,    7,    3,    9,    9
    };
    int sizeX = 7;
    int sizeY = 5;
    REQUIRE ( getLocalSum(ms, sizeY, sizeX, 0, 0) == -1 );
    REQUIRE ( getLocalSum(ms, sizeY, sizeX, 8, 1) == -1 );
    REQUIRE ( getLocalSum(ms, sizeY, sizeX, 3, 9) == -1 );
    REQUIRE ( getLocalSum(ms, sizeY, sizeX, 0, 1) == 28 );
    REQUIRE ( getLocalSum(ms, sizeY, sizeX, 1, 0) == 18 );
    REQUIRE ( getLocalSum(ms, sizeY, sizeX, 1, 1) == 20 );
    REQUIRE ( getLocalSum(ms, sizeY, sizeX, 0, 6) == 4 );
    REQUIRE ( getLocalSum(ms, sizeY, sizeX, 2, 3) == 33 );
    REQUIRE ( getLocalSum(ms, sizeY, sizeX, 4, 6) == 25 );
    REQUIRE ( getLocalSum(ms, sizeY, sizeX, 4, 0) == 20 );
    REQUIRE ( getLocalSum(ms, sizeY, sizeX, 2, 6) == 13 );
}
TEST_CASE("clip", "[clip]" ) {
    REQUIRE ( clip(9, 1, 8) == 8 );
    REQUIRE ( clip(5, 1, 5) == 5 );
    REQUIRE ( clip(4, 1, 5) == 4 );
    REQUIRE ( clip(1, 1, 5) == 1 );
    REQUIRE ( clip(0, 1, 5) == 1 );
}

```

```

TEST_CASE("sgn_plus", "[sgn_plus]") {
    REQUIRE( sgn_plus( 1) ==  1);
    REQUIRE( sgn_plus( 0) ==  1);
    REQUIRE( sgn_plus(-2) == -1);
    REQUIRE( sgn_plus( 3) ==  1);
    REQUIRE( sgn_plus(-1) == -1);
}

TEST_CASE("mod_R", "[mod_R]") {
    REQUIRE( mod_R(5, 32) ==  5);
    REQUIRE( mod_R(10, 5) == 10);
    REQUIRE( mod_R(20, 5) == -12);
    REQUIRE( mod_R(16, 5) == -16);
    REQUIRE( mod_R(0 , 5) ==  0);
}

TEST_CASE("d", "[d]") {
    uint16_t ms[] = {
        7, 2, 6, 7, 5, 1, 8,
        5, 4, 9, 9, 9, 6, 3,
        5, 2, 6, 5, 6, 1, 7,
        2, 8, 8, 8, 7, 6, 5,
        8, 1, 9, 7, 3, 9, 9
    };
    int sizeX = 7;
    int sizeY = 5;
    REQUIRE (d(ms, sizeY, sizeX, 1, 1) == -4);
    REQUIRE (d(ms, sizeY, sizeX, 0, 1) == -20);
    REQUIRE (d(ms, sizeY, sizeX, 1, 0) ==  2);
    REQUIRE (d(ms, sizeY, sizeX, 4, 6) == 11);
}

TEST_CASE("dN", "[dN]") {
    uint16_t ms[] = {
        7, 2, 6, 7, 5, 1, 8,
        5, 4, 9, 9, 9, 6, 3,
        5, 2, 6, 5, 6, 1, 7,
        2, 8, 8, 8, 7, 6, 5,
        8, 1, 9, 7, 3, 9, 9
    };
    int sizeX = 7;
    int sizeY = 5;
    REQUIRE (dN(ms, sizeY, sizeX, 1, 1) == -12);
    REQUIRE (dN(ms, sizeY, sizeX, 0, 1) ==  0);
    REQUIRE (dN(ms, sizeY, sizeX, 1, 0) == 10);
    REQUIRE (dN(ms, sizeY, sizeX, 4, 6) == -5);
}

TEST_CASE("dW", "[dW]") {
    uint16_t ms[] = {
        7, 2, 6, 7, 5, 1, 8,
        5, 4, 9, 9, 9, 6, 3,
        5, 2, 6, 5, 6, 1, 7,
        2, 8, 8, 8, 7, 6, 5,
        8, 1, 9, 7, 3, 9, 9
    };
    int sizeX = 7;
    int sizeY = 5;
    REQUIRE (dW(ms, sizeY, sizeX, 1, 1) ==  0);
    REQUIRE (dW(ms, sizeY, sizeX, 0, 1) ==  0);
    REQUIRE (dW(ms, sizeY, sizeX, 1, 0) == 10);
    REQUIRE (dW(ms, sizeY, sizeX, 4, 6) == 11);
}

TEST_CASE("dNW", "[dNW]") {
    uint16_t ms[] = {
        7, 2, 6, 7, 5, 1, 8,
        5, 4, 9, 9, 9, 6, 3,
        5, 2, 6, 5, 6, 1, 7,
        2, 8, 8, 8, 7, 6, 5,
        8, 1, 9, 7, 3, 9, 9
    };
    int sizeX = 7;
    int sizeY = 5;
    REQUIRE (dNW(ms, sizeY, sizeX, 1, 1) ==  8);
    REQUIRE (dNW(ms, sizeY, sizeX, 0, 1) ==  0);
    REQUIRE (dNW(ms, sizeY, sizeX, 1, 0) == 10);
    REQUIRE (dNW(ms, sizeY, sizeX, 4, 6) == -1);
}

TEST_CASE("getU", "[getU]") {
    uint16_t ms[4][15] = {
        {
            2, 8, 7, 3, 9,

```

```

        3, 4, 3, 7, 5,
        5, 6, 7, 3, 3
    }, {
        2, 7, 2, 1, 9,
        2, 6, 9, 5, 9,
        7, 5, 2, 5, 7
    }, {
        7, 7, 3, 5, 2,
        6, 2, 1, 5, 3,
        2, 9, 3, 2, 8
    }, {
        6, 3, 4, 3, 3,
        6, 1, 6, 4, 7,
        1, 5, 3, 3, 5
    }
};
int sizeX = 5;
int sizeY = 3;
int z = 3;
int y = 1;
int x = 1;
int P = 3;
int U[10];
getU(U, P, ms[0], sizeY, sizeX, z, y, x);
int U_1[6] = {-7, 5, 5, -15, 11, -4};
for(int i = 0; i < 6; i++) {
    INFO("Index: " << i);
    CHECK(U[i] == U_1[i]);
}
z = 1;
y = 1;
x = 1;
P = 3;
getU(U, P, ms[0], sizeY, sizeX, z, y, x);
int U_2[6] = {15, -5, -5, -4, -4, -4};
for(int i = 0; i < 6; i++) {
    INFO("Index: " << i);
    CHECK(U[i] == U_2[i]);
}
z = 0;
y = 0;
x = 1;
P = 2;
getU(U, P, ms[0], sizeY, sizeX, z, y, x);
int U_3[5] = {0, 0, 0, 0, 0};
for(int i = 0; i < 5; i++) {
    INFO("Index: " << i);
    CHECK(U[i] == U_3[i]);
}
}
TEST_CASE("weightInitDefault", "[weightInitDefault]") {
    int W[10];
    int W_1[6] = {0, 0, 0, 14, 1, 0};
    weightInitDefault(W, 4, 3);
    for(int i = 0; i < 6; i++) {
        INFO("Index: " << i);
        CHECK(W[i] == W_1[i]);
    }
    int W_2[6] = {0, 0, 0, 224, 28, 3};
    weightInitDefault(W, 8, 3);
    for(int i = 0; i < 6; i++) {
        INFO("Index: " << i);
        CHECK(W[i] == W_2[i]);
    }
}
TEST_CASE("getPredictedD") {
    int U[] = {
        7, 2, 6, 7, 5, 1, 8,
        5, 4, 9, 9, 9, 6, 3
    };
    int W[] = {
        5, 2, 6, 5, 6, 1, 7,
        2, 8, 8, 8, 7, 6, 5
    };
    REQUIRE(getPredictedD(U, W, 1) == 35);
    REQUIRE(getPredictedD(U, W, 2) == 39);
    REQUIRE(getPredictedD(U, W, 3) == 75);
}

```

```

TEST_CASE("getMappedPredictionResidual") {
    CHECK(getMappedPredictionResidual( 9,  8, 0, 63) == 9);
    CHECK(getMappedPredictionResidual( 9, 17, 0, 63) == 1);
    CHECK(getMappedPredictionResidual( 9, 22, 0, 63) == 3);
    CHECK(getMappedPredictionResidual( 8,  3, 0, 63) == 8);
    CHECK(getMappedPredictionResidual( 4,  0, 0, 63) == 4);
    CHECK(getMappedPredictionResidual(13,  5, 0, 63) == 13);
    CHECK(getMappedPredictionResidual( 8, 31, 0, 63) == 14);
    CHECK(getMappedPredictionResidual( 7, 10, 0, 63) == 4);
    CHECK(getMappedPredictionResidual(11, 40, 0, 63) == 17);
}
TEST_CASE("getRestoredValue") {
    CHECK(getRestoredValue( 9,  8, 0, 63, 32) == 9);
    CHECK(getRestoredValue( 1, 17, 0, 63, 32) == 9);
    CHECK(getRestoredValue( 3, 22, 0, 63, 32) == 9);
    CHECK(getRestoredValue( 4,  0, 0, 63, 32) == 4);
    CHECK(getRestoredValue( 5,  5, 0, 63, 32) == 5);
    CHECK(getRestoredValue( 4, 10, 0, 63, 32) == 7);
    CHECK(getRestoredValue(17, 40, 0, 63, 32) == 11);
    CHECK(getRestoredValue(13,  5, 0, 63, 32) == 13);
    CHECK(getRestoredValue(14, 31, 0, 63, 32) == 8);
}
TEST_CASE("updateW") {
    int U_1[3] = {5, 10, 15};
    int W_1[3] = {3, 1, 2};
    int size = 3;
    int e = 1;
    int ro = 1;
    int w_min = 0;
    int w_max = 4;
    int W_1_std[3] = {4, 4, 4};
    updateW(W_1, U_1, size, e, ro, w_min, w_max);
    for(int i = 0; i < size; i++) {
        INFO("Index: " << i);
        CHECK(W_1[i] == W_1_std[i]);
    }
    int U_2[3] = {5, 10, 15};
    int W_2[3] = {3, 1, 2};
    e = 3;
    ro = 2;
    w_min = -8;
    w_max = 8;
    int W_2_std[3] = {4, 2, 4};
    updateW(W_2, U_2, size, e, ro, w_min, w_max);
    for(int i = 0; i < size; i++) {
        INFO("Index: " << i);
        CHECK(W_2[i] == W_2_std[i]);
    }
    int U_3[3] = {-2, 1, -3};
    int W_3[3] = {-1, 0,  3};
    e = -3;
    ro = -1;
    w_min = -8;
    w_max = 8;
    int W_3_std[3] = {1, 0, 6};
    updateW(W_3, U_3, size, e, ro, w_min, w_max);
    for(int i = 0; i < size; i++) {
        INFO("Index: " << i);
        CHECK(W_3[i] == W_3_std[i]);
    }
}
TEST_CASE("runPredictor") {
    uint16_t input[12 * 4 * 5] = {
        100,  101,  100,  100,  101,
        100,  102,  101,  101,  101,
        102,  100,  100,  100,  103,
        102,  100,  101,  100,  102,

        100,  104,  100,  102,  100,
        103,  103,  100,  101,  103,
        102,  102,  100,  100,  100,
        102,  101,  101,  100,  102,

        102,  100,  101,  101,  101,
        102,  101,  101,  100,  101,
        102,  100,  105,  100,  100,
        101,  100,  100,  100,  100,
    };
}

```

```

100,    101,    101,    100,    101,
100,    102,    101,    101,    101,
102,    100,    100,    100,    103,
102,    100,    101,    100,    102,

100,    104,    100,    102,    100,
103,    103,    100,    101,    103,
102,    102,    100,    100,    100,
102,    101,    101,    100,    102,

102,    100,    101,    101,    101,
102,    101,    101,    100,    101,
102,    100,    105,    100,    100,
101,    100,    100,    100,    100,

100,    101,    101,    100,    101,
100,    102,    101,    101,    101,
102,    100,    100,    100,    103,
102,    100,    101,    100,    102,

100,    104,    100,    102,    100,
103,    103,    100,    101,    103,
102,    102,    100,    100,    100,
102,    101,    101,    100,    102,

102,    100,    101,    101,    101,
102,    101,    101,    100,    101,
102,    100,    105,    100,    100,
101,    100,    100,    100,    100,

100,    101,    101,    100,    101,
100,    102,    101,    101,    101,
102,    100,    100,    100,    103,
102,    100,    101,    100,    102,

100,    104,    100,    102,    100,
103,    103,    100,    101,    103,
102,    102,    100,    100,    100,
102,    101,    101,    100,    102,

102,    100,    101,    101,    101,
102,    101,    101,    100,    101,
102,    100,    105,    100,    100,
101,    100,    100,    100,    100
};
uint16_t mappedResiduals[12 * 4 * 5] = {0};
uint16_t restoredData[12 * 4 * 5] = {0};
ImageMetadata imageMeta;
imageMeta.dynamicRange = 0;
imageMeta.xSize = 5;
imageMeta.ySize = 4;
imageMeta.zSize = 12;
PredictorMetadata predMeta;
predMeta.predictionBands = 2;
predMeta.predictionMode = 0;
predMeta.localSumType = 0;
predMeta.registerSize = 32;
predMeta.weightComponentResolution = 0;
predMeta.wuScalingExpChangeInterval = 0;
predMeta.wuScalingExpInitialParameter = 0;
predMeta.wuScalingExpFinalParameter = 15;
predMeta.weightInitMethod = 0;
predMeta.weightInitTableFlag = 0;
predMeta.weightInitResolution = 0;
runPredictor(input, mappedResiduals, &imageMeta, &predMeta, PREDICTOR_MAP);
runPredictor(mappedResiduals, restoredData, &imageMeta, &predMeta, PREDICTOR_RESTORE);
int size = 12 * 4 * 5;
for(int i = 0; i < size; i++) {
    INFO("Index: " << i);
    CHECK(input[i] == restoredData[i]);
}
}
TEST_CASE("getCodeWordSize") {
    REQUIRE(getCodeWordSize(5, 6) == 0);
    REQUIRE(getCodeWordSize(16, 35) == 1);
    REQUIRE(getCodeWordSize(80, 1120) == 3);
    REQUIRE(getCodeWordSize(15, 10) == 0);
    REQUIRE(getCodeWordSize(12, 856) == 6);
}

```



```

}
TEST_CASE("Golomb") {
    uint16_t input[3 * 4 * 5] = {
        1,    1,    0,    0,    1,
        0,    2,    1,    1,    1,
        2,    0,    0,    0,    3,
        2,    0,    1,    0,    2,

        0,    4,    0,    2,    0,
        3,    3,    0,    1,    3,
        2,    2,    0,    0,    0,
        2,    1,    1,    0,    2,

        2,    0,    1,    1,    1,
        2,    1,    1,    0,    1,
        2,    0,    5,    0,    0,
        1,    0,    0,    0,    0
    };
    uint32_t encodedOut[3 * 4 * 5] = {0};
    uint16_t decodedOut[3 * 4 * 5] = {0};
    size_t outSize = 0;
    ImageMetadata imageMeta;
    imageMeta.dynamicRange = 0;
    imageMeta.xSize = 5;
    imageMeta.ySize = 4;
    imageMeta.zSize = 3;
    EncoderMetadata encoderMeta;
    encoderMeta.unaryLengthLimit = 8;
    encoderMeta.accumInitConstant = 2;
    encoderMeta.initialCountExponent = 1;
    encoderMeta.rescalingCounterSize = 1;
    encodeGolomb(input, encodedOut, &outSize, &imageMeta, &encoderMeta);
    decodeGolomb(encodedOut, decodedOut, &imageMeta, &encoderMeta);
    int size = 3 * 4 * 5;
    for(int i = 0; i < size; i++) {
        INFO("Index: " << i);
        CHECK(input[i] == decodedOut[i]);
    }
}

TEST_CASE("save/load PGM") {
    uint16_t data[6 * 7] = {
        215,    2447,    3842,    1083,    119,    672,    1782,
        2406,    3397,    2431,    2823,    1154,    1462,    3492,
        898,    2892,    1144,    2849,    2422,    3584,    1153,
        1287,    1389,    1479,    3597,    2460,    2089,    2111,
        589,    2685,    140,    3056,    1198,    2115,    656,
        2311,    2867,    1166,    386,    1462,    2163,    3300
    };
    uint16_t * loadedData = NULL;
    char fileName[] = "test.pgm";
    unsigned sizeX = 7;
    unsigned sizeY = 6;
    unsigned maxValue = 4095;
    unsigned loadedSizeX;
    unsigned loadedSizeY;
    unsigned loadedMaxValue;
    CHECK(saveToPGM(fileName, data, sizeX, sizeY, maxValue) == 0);
    CHECK(loadFromPGM(fileName, &loadedData, &loadedSizeX, &loadedSizeY, &loadedMaxValue) == 0);
    if(loadedData) {
        int size = sizeX * sizeY;
        for(int i = 0; i < size; i++) {
            INFO("Index: " << i);
            CHECK(data[i] == loadedData[i]);
        }
        free(loadedData);
    }
}

TEST_CASE("save/load compressed image") {
    char fileName[] = "test_compressed.fl";
    uint32_t data[6 * 5] = {
        581612148,    840150402,    170625740,    1768229585,    4127652133,
        2166189274,    921143472,    2829598208,    4086598280,    1337180266,
        2599861949,    1672172116,    855853699,    2135216968,    3721482342,
        2363560580,    1555567274,    3700090308,    974886365,    3518892602,
        2865892643,    3278841227,    1314848842,    2703874718,    229157794,
        4130334434,    119450082,    794094992,    756900839,    635552470
    };
    uint32_t * loadedData = NULL;

```

```

ImageMetadata l_imageMeta;
PredictorMetadata l_predMeta;
EncoderMetadata l_encoderMeta;
size_t size = 6 * 5;
size_t dataSize = size * sizeof (uint32_t);
size_t loadDadaSize = 0;
ImageMetadata imageMeta;
imageMeta.userData = 127; // : 8;
imageMeta.xSize = 500; // : 16;
imageMeta.ySize = 800; // : 16;
imageMeta.zSize = 200; // : 16;
imageMeta.sampleType = 0; // : 1;
imageMeta.reserved_0 = 3; // : 2;
imageMeta.dynamicRange = 8; // : 4;
imageMeta.sampleEncodingOrder = 0; // : 1;
imageMeta.subFrameInterleavingDepth = 12000; // : 16;
imageMeta.reserved_1 = 2; // : 2;
imageMeta.outputWordSize = 5; // : 3;
imageMeta.entropyCoderType = 0; // : 1;
imageMeta.reserved_2 = 0; // : 10;
PredictorMetadata predMeta;
predMeta.reserved_0 = 3; // : 2;
predMeta.predictionBands = 5; // : 4;
predMeta.predictionMode = 0; // : 1;
predMeta.reserved_1 = 1; // : 1;
predMeta.localSumType = 0; // : 1;
predMeta.reserved_2 = 0; // : 1;
predMeta.registerSize = 3; // : 6;
predMeta.weightComponentResolution = 10; // : 4;
predMeta.wuScalingExpChangeInterval = 6; // : 4;
predMeta.wuScalingExpInitialParameter = 0; // : 4;
predMeta.wuScalingExpFinalParameter = 12; // : 4;
predMeta.reserved_3 = 0; // : 1;
predMeta.weightInitMethod = 0; // : 1;
predMeta.weightInitTableFlag = 0; // : 1;
predMeta.weightInitResolution = 20; // : 5;
EncoderMetadata encoderMeta;
encoderMeta.unaryLengthLimit = 7; // : 5;
encoderMeta.rescalingCounterSize = 2; // : 3;
encoderMeta.initialCountExponent = 6; // : 3;
encoderMeta.accumInitConstant = 11; // : 4;
encoderMeta.accumInitTableFlag = 0; // : 1;
CHECK(saveCompressedImage(fileName, data, dataSize, &imageMeta, &predMeta, &encoderMeta) ==
0);

memset(&l_imageMeta, 0, sizeof (ImageMetadata));
memset(&l_predMeta, 0, sizeof (PredictorMetadata));
memset(&l_encoderMeta, 0, sizeof (EncoderMetadata));

CHECK(loadCompressedImage(fileName, (void **)&loadedData, &loadDadaSize,
&l_imageMeta, &l_predMeta, &l_encoderMeta) == 0);
CHECK(memcmp(&imageMeta, &l_imageMeta, sizeof (ImageMetadata)) == 0);
CHECK(memcmp(&predMeta, &l_predMeta, sizeof (PredictorMetadata)) == 0);
CHECK(memcmp(&encoderMeta, &l_encoderMeta, sizeof (EncoderMetadata)) == 0);
if(loadedData) {
    for(int i = 0; i < size; i++) {
        INFO("Index: " << i);
        CHECK(data[i] == loadedData[i]);
    }
    free(loadedData);
}
}

```

**ПРИЛОЖЕНИЕ Б**  
*(обязательное)*

Спецификация

**ПРИЛОЖЕНИЕ В**  
*(обязательное)*

Ведомость документов