В.С. ЧЕРНЕГА

С Ж А Т И Е
И Н Ф О Р М А Ц И И
В К О М П Ь Ю Т Е Р Н Ы Х
С Е Т Я Х

MPEG4

LZW JPEG RLE Px64

LZSS

LZSS MHC

ZIP LZ77

ZIP MPEG READ

PCX

HUFF JPEG LZArc Px64

LZ78 PKZIP JPEG

Министерство образования и науки Украины
Институт содержания и методов обучения
Севастопольский государственный технический университет

В.С. ЧЕРНЕГА

СЖАТИЕ ИНФОРМАЦИИ
В
КОМПЬЮТЕРНЫХ
СЕТЯХ

Под редакцией д-ра техн. наук, проф. В.К. Маригодова

Рекомендовано Министерством образования Украины в качестве учебного пособия для студентов вузов, обучающихся по направлениям
“Компьютерные науки” и “Компьютерная инженерия”

“ СевГТУ ”
Севастополь 1997

ББК 32.97
Ч46
УДК 621.391.251: 681.3.053

Рецензенты:
д-р техн.наук , проф. Ю.П.Жураковский, д-р техн.наук, проф. Н.Е. Сапожников

Чернега В.С.

Ч46 Сжатие информации в компьютерных сетях: Учебное пособие для вузов: Под ред. д.т.н., проф. В.К. Маригодова. — Севастополь: СевГТУ,—1997. — 212 с.: ил.

ISBN 5-7763-8853-8

В книге изложены новые методы сжатия текстовых сообщений на основе динамического кодирования неравномерными кодами Хаффмена, кодирования строк символов на основе алгоритмов Лемпеля-Зива (LZ) и методы арифметического кодирования. Значительное внимание уделяется вопросам программной реализации приведенных алгоритмов сжатия.

Рассмотрены одномерные и двухмерные методы сжатия черно-белых факсимильных изображений, а также методы сжатия многоградационных черно-белых и цветных неподвижных и подвижных изображений с частичной потерей информации на базе кодирования с дискретно-косинусным преобразованием, в том числе стандартные процедуры кодирования изображений JPEG и MPEG.

Книга предназначена в качестве учебного пособия для студентов вузов, обучающихся по направлениям “Компьютерные науки”, “Компьютерная инженерия”, “Компьютеризированные системы автоматики и управления” и др., а также для широкого круга лиц, интересующихся вопросами построения и создания архиваторов данных, компрессоров информации в телекоммуникационных системах и системах мультимедиа. Она может быть полезна для опытных программистов в качестве справочного пособия.

Чернега Виктор Степанович
Сжатие информации в компьютерных сетях

V.S.Chernega
Information Compression on Computer Networks

ISBN 5-7763-8853-8

Оригинал-макет книги подготовлен автором

Формат 60x90/16. Бум. тип №2. Офс. печ. Усл. п.л. 13,5. Тираж 500 экз. Заказ № 41
Издательство “СевГТУ”, Украина, 335053 г.Севастополь 53, Студгородок, НМЦ

ПРЕДИСЛОВИЕ

Повышение эффективности функционирования народного хозяйства непосредственно связано с широким внедрением средств вычислительной техники во все отрасли человеческой деятельности, объединение персональных ЭВМ в компьютерные сети. Производительность существующих компьютерных сетей может быть заметно увеличена за счет повышения скорости передачи данных по каналам связи, которое обеспечивается использованием методов сжатия информации. Вопросы сжатия информации (эффективного кодирования) играют важную роль в подготовке специалистов по информатике и вычислительной технике и входят в программы спецкурсов подготовки инженеров ряда специальностей, в частности:

7.080401 — Компьютеризированные системы обработки информации и управления.

7.091401 — Автоматизированное управление в технических и организационных системах.

Существующие учебные пособия и научные монографии не отражают современное состояние теории и практики компрессии данных. В них описываются только классические двухпроходные алгоритмы Шеннона-Фано и Хаффмена, разработанные в начале 50-х годов.

В последнее десятилетие наблюдается бурное развитие теории и практических методов сжатия текстовых и графических сообщений. За это время за рубежом появилось большое количество научных публикаций в периодической литературе и ряд монографий, в которых предложены новые методы сжатия информации, использующие адаптивные алгоритмы динамического сжатия Хаффмена, кодирования строк переменной длины методом Лемпеля-Зива, арифметическое кодирование и др. Сведения, изложенные в этих источниках, являются практически недоступными для студентов вузов. Это связано, во-первых, с тем, что излагаемые в периодической печати материалы написаны очень лаконично и предназначены для подготовленных в этой области читателей. Во-вторых, практически все публикации изданы на английском или немецком языках и, кроме того, многие источники отсутствуют в библиотеках вузов Украины.

Во время научно-педагогической стажировки автора в Федеральной высшей технической школе (ETH) г.Цюриха (Швейцария) в 1994 - 96 гг. им было проработано большое количество первоисточников по проблемам сжатия информации. На основе собранного материала подготовлен курс

лекций для студентов СевГТУ и учебное пособие для обучающихся по направлениям “Компьютерные науки” и “Компьютерная инженерия”.

Цель учебного пособия “Сжатие информации в компьютерных сетях” — обобщить опыт разработки и использования способов сжатия информации в ведущих странах мира и изложить основы методов проектирования эффективных систем компрессии текстовых и графических сообщений. Книга будет полезной не только для студентов указанных специальностей, но и для учащихся экономических вузов, факультетов повышения квалификации по вычислительной технике и информатике. Она может быть также использована специалистами, занимающихся вопросами создания архиваторов данных, компрессоров информации в телекоммуникационных системах и системах мультимедиа.

Автор глубоко признателен научному редактору, д.т.н., проф. В.К.Маригодову, за его большой труд по редактированию рукописи, а также рецензентам — профессору кафедры автоматики и управления в технических системах Национального технического университета Украины (Киевский политехнический институт) д.т.н., проф. Ю.П.Жураковскому и заместителю начальника по научной работе Севастопольского военно-морского института д.т.н., проф. Н. Е. Сапожникову за доброжелательную критику и полезные замечания, которые способствовали улучшению книги.

Пожелания и замечания по книге просьба направлять по адресу: Украина, 335053 г.Севастополь, Студгородок, Севастопольский государственный технический университет, НМЦ, Издательство СевГТУ.

Автор

ВВЕДЕНИЕ

Стремительный рост объемов информации, производимой человеком в процессе его жизнедеятельности, массовое подключение пользователей персональных компьютеров к всемирным информационным сетям, внедрение мультимедийных информационных систем привело к тому, что существующие каналы связи не в состоянии обеспечить необходимую пропускную способность, гарантирующую доставку сообщений потребителям в требуемые сроки. Внедрение широкополосных каналов, оптических линий связи позволит в значительной степени решить проблемы своевременной доставки сообщений. Однако процесс внедрения таких каналов является весьма дорогостоящим и требует значительного времени.

Вторая проблема, вызванная информационным взрывом в обществе - хранение информации. Несмотря на поразительные достижения в последние годы в области создания высокоэффективных запоминающих устройств на базе магнитных и оптических методов записи информации, эти устройства не могут без специальных мер обеспечить хранение необходимых объемов информации, в частности подвижных изображений. Так только для одного кадра цветного изображения с разрешением 620×560 пикселов требуется примерно 1 Мбайт памяти, а для записи одночасового фильма с движущимся изображением при частоте 30 кадров в секунду потребовалось бы около 108 Гбайт памяти.

Повышение эффективности существующих систем передачи и хранения информации сравнительно просто и с минимальными затратами достигается путем использования методов сжатия (компрессии) сообщений. В процессе сжатия информации производится уменьшение объема исходного сообщения за счет устранения естественной избыточности, присущей практически всем источникам информации. В связи с тем, что компрессия в настоящее время осуществляется преимущественно программным способом, при ее внедрении не требуется дополнительно изменять действующие каналы передачи, системы записи и носители информации. Это обстоятельство играет особую роль, так как в пользовании находится большое число технических средств передачи и хранения информации срок службы которых далеко не исчерпан, а их замена требует немалых средств.

Впервые метод сжатия текстовых сообщений был разработан основоположниками классической теории информации К.Шенноном и Р. Фано в 1948 г. Ими было предложено кодировать символы сообщений неравномерным кодом, причем длина кодовой комбинации зависит от

вероятности появления символа в сообщении. Этот код получил название кода Шеннона-Фано. Второй важнейшей вехой в области сжатия информации явилась работа Д.Хаффмена, опубликованная в 1952 г. В ней предложен метод построения оптимального неравномерного кода, обеспечивающего минимальную длину закодированного сообщения. Коды Шеннона-Фано и Хаффмена относятся к классическим методам сжатия сообщений, на основе которых возникло множество других алгоритмов компрессии. Классические методы сжатия предполагают, что в процессе передачи или записи информации на носитель таблица кодирования остается неизменной, в связи с чем их можно отнести к статическим методам сжатия сообщений.

Неравномерные коды с момента их появления длительное время не находили широкого применения на практике в связи с трудностями технической реализации. Взамен их были разработаны упрощенные методы сжатия, использующие наличие в текстовых и графических сообщениях повторяющихся символов, их групп и целых строк. Эти методы не обеспечивали оптимального сжатия, но были достаточно эффективны и просты в реализации.

Качественный скачок в области сжатия информации произошел в конце 70-х годов, когда для компрессии текстовых и графических сообщений были предложены арифметическое кодирование и кодирование строк символов переменной длины, основанное на базе алгоритмов А.Лемпеля и Я.Зива. Широкому внедрению классических и новых методов сжатия в практику способствовало появление быстродействующих микропроцессоров и полупроводниковых запоминающих устройств большой емкости. Появились методы, которые позволяли осуществлять компрессию данных в темпе их поступления на вход кодирующего устройства, без предварительного анализа статистических свойств сообщения. При этом происходила адаптация компрессора к статистическим характеристикам сжимаемого сообщения. Такие методы получили название динамических методов сжатия данных. Примерно в это же время были разработаны алгоритмы динамического сжатия с использованием кодов Хаффмена, в которых нет жесткого закрепления за символами сообщения определенных кодовых комбинаций. Таблица кодирования непрерывно изменяется в соответствии со статистическими характеристиками сжимаемого сообщения. Хотя эффективность динамических методов сжатия немного ниже оптимального кода Хаффмена, основным их преимуществом является высокое быстродействие по сравнению со статическими и возможность работать в реальном времени.

В алгоритмах сжатия черно-белых неподвижных и подвижных изображений, наряду со специально разработанными методами, широко

используется арифметическое и хаффменовское кодирование. Методы сжатия многоградационных черно-белых и цветных изображений существенно отличаются от методов компрессии текстовых сообщений. Однако во многих случаях на последующих ступенях процедуры сжатия изображений используется код Хаффмена или арифметическое кодирование.

Книга состоит из введения, восьми глав, заключения и приложений. В первой главе вводятся основные понятия и определения техники сжатия данных. Описан ряд простейших способов компрессии текстовых и графических сообщений, которые применяются и в настоящее время, либо являются составной частью более сложных методов. Во второй главе приведены краткие сведения и основные аналитические соотношения теории кодирования источников, которые используются в последующих главах.

Третья глава посвящена большей частью классическим двухпроходным методам сжатия, в частности методам Шеннона-Фано и Хаффмена. Эти методы рассматриваются с позиций их программной реализации на базе современной вычислительной техники. В этой же главе описан новый метод компрессии данных, основанный на арифметическом кодировании и рассмотрены особенности программной реализации такого метода в современных микропроцессорных системах.

В четвертой и пятой главах детально рассмотрены новые методы динамического кодирования текстовых сообщений. Детально рассмотрены процедуры построения кодовых деревьев оптимального кода Хаффмена. Проанализированы недостатки методов кодирования строк переменной длины и показаны пути их устранения. Рассмотрены пути повышения быстродействия при поиске строк символов в кодовой таблице.

Шестая и седьмая главы посвящены методам сжатия неподвижных черно-белых и полутоновых изображений. Подробно рассмотрены алгоритмы одномерных и двухмерных методов сжатия черно-белых и цветных изображений. Приведены расчетные формулы предсказания значения яркости последующих пикселов многоградационных изображений, описана стандартная процедура сжатия неподвижных изображений JPEG.

Восьмая глава полностью посвящена проблемам сжатия цветных подвижных изображений. В ней рассмотрены методы внутрикадрового и межкадрового кодирования, оценки параметров движения объектов, а также изложены особенности построения стандарта MPEG.

Глава 1

ОСНОВНЫЕ ПОНЯТИЯ, ОПРЕДЕЛЕНИЯ И ПРОСТЕЙШИЕ СПОСОБЫ СЖАТИЯ ДАННЫХ

1.1. Основные определения техники сжатия данных

Сжатием (*компрессией*) данных называется процедура снижения избыточности сообщения с целью уменьшения его объема. В предлагаемой книге рассматриваются методы уменьшения избыточности, которое проводится на основе формальных критериев без учета смыслового содержания (семантики) сообщения. В процессе компрессии данных устраняется естественная избыточность источника информации, а при декомпрессии, перед выдачей данных потребителю, происходит восстановление исходного сообщения. Устранение избыточности источников информации осуществляется путем применения эффективного кодирования. Обобщенная схема сжатия данных изображена на рис.1.1.

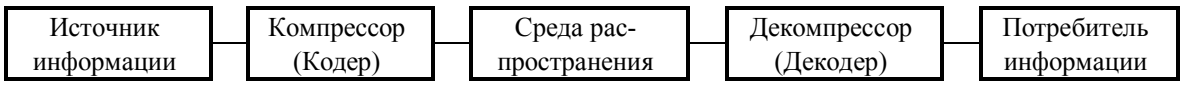


Рис. 1.1 Обобщенная структурная схема сжатия данных

Для оценки эффективности процедуры сжатия сообщений используется несколько показателей степени компрессии данных. При оценке эффективности сжатия текстовых сообщений наиболее широко используется коэффициент сжатия $K_{СЖ}$, который характеризует объем сообщения $V_{СЖ}$ (в битах или байтах) на выходе компрессора после сжатия по отношению к исходному объему $V_И$

$$K_{сж} = V_{сж} / V_и . \qquad (1.1)$$

Часто коэффициент сжатия выражают в процентах. Для этого значение, вычисленное по (1.1), умножается на 100. Очевидно, что при отсутствии сжатия $K_{сж}$ = 100 % и уменьшается с повышением эффективности процедуры компрессии.

Оценка уменьшения объема изображений в процессе их сжатия в основном производится с помощью коэффициента компрессии $K_С$, который определяется по формуле

$$K_с = V_И / V_{сж}. \qquad (1.2)$$

Он показывает во сколько раз уменьшился объем исходного сообщения после компрессии. Не трудно заметить, что $K_{сж}$ и $K_с$ являются обратными величинами, т. е. $K_{сж} = 1 / K_с$.

Некоторые авторы для определения степени сжатия используют коэффициент сжатия данных $K_{СД}$, определяемый соотношением

$$K_{СД} = (1 - V_{сж} / V_И)100\ \% , \qquad (1.3)$$

который характеризует объем данных, исключенных из сообщения в процессе его сжатия. При отсутствии эффекта сжатия $K_{СД}$ = 0 % , а в случае максимального сжатия коэффициент $K_{СД}$ приближается к 100%.

В теоретических работах по компрессии данных часто эффективность процедуры сжатия оценивается затратами количества битов на один байт сообщения. Каждый из рассмотренных коэффициентов по разному отображает связь между одними и теми же величинами: объемами исходного и сжатого сообщения и трудно отдать предпочтение одному из них. В данной книге при рассмотрении конкретных примеров, иллюстрирующих один из методов сжатия, будет использоваться $K_{сж}$ (1.1) или $K_С$ (1.2), в зависимости от того, какой из коэффициентов чаще применяется в данной области техники сжатия сообщений.

Процесс исторического развития методов сжатия сообщений характеризуется реализацией вначале простейших методов, которые позволяли сокращать объем сообщения за счет кодирования групп одинаковых символов, либо выделения повторяющихся групп символов и фраз и заменой их определенными кодовыми комбинациями.

На основе фундаментальных исследований в теории информации, проведенных К.Шенноном, Р.Фано, Р.Е.Кричевским и др. авторами [10,16,21], были разработаны методы кодирования источников информации, которые стали классическими. В процессе реализации этих методов вначале создается модель сообщения, а затем осуществляется кодирование безизбыточными кодами. При этом модель до завершения

компрессии остается неизменной. Такие методы можно отнести к группе статических методов сжатия информации.

В конце 70-х годов были разработаны динамические методы сжатия данных [14,28,32,37], в которых процесс построения модели и процедура кодирования производятся одновременно. При изменении статистических свойств сообщения осуществляется модификация модели, то есть происходит адаптация ее к сжимаемому сообщению.

Все методы компрессии сообщений делятся на две большие группы: *методы сжатия без потерь* и *методы сжатия с частичной потерей информации*. В случае сжатия без потерь восстановленное после декомпрессии сообщение полностью соответствует исходному. Эти методы используются в файловых архиваторах, при сжатии текстовых сообщений, в аппаратуре передачи данных. Они также применяются в большинстве систем передачи факсимильных сообщений.

Системы сжатия с частичной потерей информации широко применяются в системах обработки полутоновых и цветных неподвижных и подвижных изображений, предназначенных для восприятия человеком. В них учитывается особенность зрительного анализатора человека не замечать незначительных искажений изображения. Поэтому в таких системах при компрессии сообщения часть несущественной информации отбрасывается, а восстановленное после декомпрессии сообщение отличается от исходного. За счет устранения при компрессии части информации удается существенно увеличить степень сжатия сообщения без заметного ухудшения качества воспроизводимого изображения.

Методы сжатия текстовых сообщений без потерь в свою очередь могут быть разделены на две подгруппы: *статические* и *динамические* методы. В статических методах компрессор на основе анализа статистических свойств сообщения составляет кодовую таблицу и передает ее декомпрессору. В процессе сжатия сообщения эта таблица не изменяется. При динамических методах компрессия данных осуществляется в темпе поступления символов сообщения и одновременно строится таблица кодирования и декодирования. В процессе сжатия сообщения осуществляется синхронная модификация кодовых таблиц на кодирующей и декодирующей сторонах.

Динамическое сжатие может осуществляться путем кодирования символов фиксированной длины неравномерными кодовыми комбинациями (*variable - length source codes*), либо кодирования строк символов переменной длины равномерными кодовыми комбинациями (*variable - to - fixed length coding*).

В процессе преобразования неподвижных изображений (*факсимиле*) они разделяются на ряд строк, элементами которых являются элементарные площадки белого и черного цветов, отображаемых

соответственно сигналами логических 0 и 1. Процедуре сжатия подвергаются последовательности серий нулей и единиц одной или нескольких строк изображения. Применяемые при этом методы компрессии получили соответственно название одномерного и многомерного (чаще двумерного) кодирования.

При сжатии полутоновых и цветных изображений соответствующий им цифровой сигнал подвергается предварительной обработке и преобразованию, в ходе которых исключается часть малосущественной для зрителя информации. После этого полученный массив данных кодируется одним из методов, применяемых для сжатия сообщений без потерь.

1.2. Кодирование длины повторяющихся символов

В текстовых файлах часто встречаются относительно длинные последовательности одинаковых символов. В первую очередь это символы пробела, дефиса и некоторых специальных знаков. Избыточность таких сообщений может быть существенно сокращена за счет замены группы одинаковых символов последовательностью, состоящей из трех байтов. Первый является специальным признаком компрессии *S*к, индицирующим начало сжатой строки; второй *Ch* - собственно повторяющийся символ и третий *Cn* - счетчик количества одинаковых символов в сжимаемой последовательности. Этот метод получил название "*Run-Length encoding*" *RLE* - кодирование [31]. Например, при поступлении от источника последовательности символов ABCCCCCDEAAAA7C строка сжатых данных приобретает вид ABSкC5DESкA47C, т. е. вместо 15 она занимает объем 12 байтов. Очевидно, что сжимать исходную последовательность целесообразно при длине одинаковых символов в строке не менее 4-х. Максимальное число одинаковых символов ограничивается разрядностью счетчика *Cn* и обычно не превышает 255.

В качестве признака компрессии можно выбрать любую неиспользуемую комбинацию кода КОИ-7 или КОИ-8. Для случаев, когда используются все наборы заданного кода обработки информации, можно применять двойные символы, которые не могут встречаться в тексте (например ЬЬ). В этом случае минимальное количество повторяющихся символов в блоке, которое целесообразно сжимать, равно пяти.

Способ *RLE* достаточно широко использовался в системах архивации до середины 80-х годов, в частности при сжатии псевдографических изображений. В настоящее время он также находит применение при сжатии неподвижных изображений (факсимильные сообщения, файлы

PCX-формата), а также является составной частью ряда комбинированных способов сжатия.

1.3. Применение бит-индикаторов

Во многих текстовых сообщениях имеет место ситуация наличия большого числа пробелов или каких-либо других знаков, которые располагаются по одному или небольшими группами между символами, что исключает применение рассмотренного выше способа кодирования групп одинаковых символов. В этом случае определенная степень сжатия может быть достигнута при использовании способа побитного отображения, известного в зарубежной литературе под названием *"Bit mapping"*. Такое название способ получил от того, что при сжатии данных символ, наиболее часто встречающийся в сообщении, исключается, а в сообщение вводится вспомогательная двоичная последовательность, отображающая структуру сжатой строки [31].

Для иллюстрации способа рассмотрим некоторую строку данных, состоящей из произвольных символов, разделенных одним или несколькими знаками пробелов (Рис.1.2,*а*). Формат строки с исключенными знаками пробела показан на рис.1.2,*б*.

В начале сжатой строки размещается управляющее слово - *бит-индикатор*, каждый разряд которого определяет один из символов сжимаемой строки данных. Причем, номер единичной позиции указывает на месторасположение символа данных, а номер нулевой позиции - на наличие и позицию символов пробела в несжатой строке. Направление счета номера позиции бит-индикатора не играет роли, однако должно быть заранее оговорено для кодера и декодера. На рис.1.2,*б* принят счет номеров битов слева направо, а нули на 2,4,5 и 6-й позициях показывают, что из исходной строки исключены знаки пробела, которые располагались на соответствующих символьных позициях.

Символ данных 1		Символ данных 2				Символ данных 3	Символ данных 4

а)

Управляющее слово	Символ данных 1	Символ данных 2	Символ данных 3	Символ данных 4

б)

Рис. 1.2. Формат исходной строки (а) и с исключенными символами пробела (б)

Коэффициент сжатия сообщения зависит от количества пробелов в нем. Худшая ситуация наблюдается при полном отсутствии пробелов в тексте. Если длина сжимаемой строки равна S символов, то общее количество символов после сжатия в случае отсутствия пробелов равно $S+S$ / 8. Здесь второе слагаемое учитывает, что на каждые 8 символов требуется один символ бит-указателя. Тогда коэффициент сжатия равен

$$K_{сж} = (S / 8 + S) / S = 1{,}125 . \qquad (1.4)$$

Таким образом, массив "сжатой" информации увеличивается на 12,5%. В лучшем случае, при наличии сплошных пробелов, $K_{сж}$= 0,125, и объем сжатых данных уменшается в 8 раз.

Если же вероятность появления пробелов равна $P_П$, то их общее число в строке длиной S будет $P_П S$, а коэффициент сжатия соответственно равен:

$$K_{сж} = \{S (1 - P_П) + S / 8\} / S = (1 - P_П) + 1/8 . \qquad (1.5)$$

1.4. Сжатие цифровых последовательностей

Если блоки данных преимущественно содержат числовую информацию, то сжатие сообщений может быть достигнуто путем уменьшения числа битов на знак с 7 до 4-х, то есть заменой (*упаковкой*) комбинаций кода КОИ-7 *(ASCII)* на четырехразрядные. Числа в коде КОИ-7 всегда имеют в трех старших разрядах комбинацию 011 и поэтому нет необходимости передавать эти биты. Кроме цифр в упакованной форме могут быть переданы знаки (+) и (-), а также десятичная точка (.). Это связано с тем, что младшие тетрады этих символов отличаются от младших тетрад десятичных чисел и при их распаковке могут быть легко преобразованы в соответствующий им эквивалент в *ASCII*-коде. Знак пробела, который часто встречается в цифровых последовательностях, целесообразно кодировать четырехразрядной комбинацией, состоящей из четырех единиц 1111, что соответствует младшей тетраде символа (/).

Для того, чтобы при распаковке (на приемной стороне) можно было определить начало упакованной последовательности, используются символ признака сжатия данных *Sк* и счетчик количества упакованных чисел *Сп*, которые размещаются непосредственно перед сжатой последовательностью. Пример фрагмента кадра с упакованной последовательностью десятичных чисел со знаком и десятичной точкой показан на рис.1.3.

Sk	*Ch*	‘ 2 ’	‘ 6 ’	‘ . ‘	‘ 3 ‘	‘ / ‘	. . .	‘ / ‘	‘ 5 ‘	‘ 7 ‘	‘ 4 ‘

Рис. 1.3. Фрагмент кадра с упакованной последовательностью десятичных чисел

Коэффициент сжатия упакованной десятичной последовательности зависит от её длины:

$$K_{\text{сж}} = V_{УП} / V_{И} = (n_{\text{Sk}} + n_{\text{ch}} + 4\, N_{\text{Б}}) / 8\, N_{\text{Б}} , \qquad (1.6)$$

где $V_{УП}$ и $V_{И}$ — объем соответственно упакованной и исходной последовательности в битах; $N_{\text{В}}$ — количество байтов исходной числовой последовательности; n_{Sk} , n_{cn} — количество битов для кодирования символа признака сжатия данных и счетчика упакованных чисел соответственно.

Если для представления n_{Sk} и n_{cn} выбирается по одному байту, то коэффициент сжатия рассчитывается по формуле:

$$K_{сж} = 2 / N_{\text{В}} + 0{,}5 . \qquad (1.7)$$

Отсюда видно, что для того, чтобы $K_{\text{сж}}$ был меньше 1, сжимать следует цифровые последовательности данных не менее пяти байтов. Минимально достижимый коэффициент сжатия равен

$$K_{\text{сж min}} = \lim_{N_B \to \infty} \left(\frac{n_{Sk} + n_{cn}}{8 N_B} + 0{,}5 \right) = 0{,}5 . \qquad (1.8)$$

Таким образом, исходная цифровая последовательность может быть сжата до 50% от своего первоначального вида, что эквивалентно повышению эффективной скорости передачи в 2 раза (или соответственно двукратному снижению объема занимаемой памяти).

1.5. Способы замены строк и шаблонов

Диатомическое кодирование. В словарных конструкциях различных языков всегда существуют устойчивые буквосочетания (*строки*), которые встречаются чаще других символов, причем процентное содержание комбинаций, состоящих из двух или трех букв, намного выше более длинных сочетаний. Это свойство используется при диатомическом кодировании, при котором сжатие происходит за счет замены пары символов специальными кодовыми словами.

Максимальное количество заменяемых пар определяется разрядностью выходных слов. Следует заметить, что общее количество наиболее часто встречающихся пар символов относительно небольшое. Как показали исследования ряда авторов [31,40] в английском тексте 25 наиболее часто встречающихся пар составляют около 34 % от общего числа символов в тексте. Значения таких комбинаций и их общее количество на тысячу знаков приведено в Приложении П4 . В программах, написанных на языках высокого уровня (БЕЙСИК, КОБОЛ, ФОРТРАН) 138 пар символов составляют около 20% от общего числа используемых в тексте знаков.

Для повышения эффективности процедуры компрессии необходимо знать средние частоты появления пар символов в сжимаемых массивах. Если в качестве выходных кодовых слов использовать 8-разрядные комбинации, то можно уменьшить объем совокупности пар символов на 50%.

Замена языковых шаблонов. Этот способ является более высоким уровнем диатомического кодирования при котором кодируются не отдельные буквосочетания, а ключевые слова, являющиеся типовыми образами *(шаблонами)* определенного языка. Способ обладает достаточно высокой эффективностью при архивации и передаче текстов программ и других файлов данных, содержащих повторяющиеся фрагменты слов или целых фраз. Так, например, в языках программирования высокого уровня часто используются слова-шаблоны *READ, WRITE, PRINT, IF, FOR, BEGIN, END* и другие. В общих текстах английского языка наиболее часто встречаются слова "and", "the", "that", "this" и пр. Суть способа состоит в том, что вместо шаблонов языка осуществляется генерация специальных кодовых слов, не используемых для отображения отдельных символов алфавита. Чтобы осуществлять такую замену составляется таблица соответствия между шаблонами и специальными символами компрессии. Причем, в качестве шаблонов могут использоваться не только отдельные слова языка, но и строки, состоящие из 3-х, 4-х или более пробелов. Так как число ключевых слов в языке может быть достаточно большим, то в

качестве специальных символов компрессии применяются последовательности, состоящие из двух байт, первый символ которой является признаком замены, а второй - отображает конкретный шаблон. Для первого символа следует выбирать редко или вообще неиспользуемый символ. Примером фрагмента таблицы замены, применяемой при сжатии программ, написанных на языке Турбо-Паскаль (TP), является табл.1.1. Здесь знак денежной единицы $ является индикатором замены, а знак " _ " — пробелом.

Таблица 1.1

Шаблон	BEGIN	ELSE_	END	IF_	READ	THEN	WRITE
Кодовое слово	$1	$2	$3	$4	$5	$6	$7

Однако большинство языков программирования используют практически все символы кода *ASCII*, в том числе и знак $. Чтобы исключить ошибочную интерпретацию признака сжатия $ при декомпрессии данных, применяют дублирование этого символа, которое свидетельствует, что это не признак компрессии, а несжатый символ $. Ниже приведена иллюстрация процедуры сжатия фрагмента программы, написанной на языке Турбо-паскаль. Обратите внимание, что некоторые операторы языка, которые требуют разделительного пробела, заменяются при сжатии вместе с этим пробелом. Как видно из примера компрессии, исходный фрагмент программы содержит 106 символов, а сжатый - 85, т.е. коэффициент сжатия равен 0,8.

Исходная программа:	Сжатая программа:
BEGIN	$1
WRITE (‘T1=‘);	$7(`T1=`);
READ (T1);	$5(T1);
WRITE(‘ T2=‘);	$7(`T2=`);
READ (T2);	$5(T2);
IF T1 <= t < T2	$4T1 <= t < T2
THEN $R:= E*Z1	$6$$R:= E*Z1
ELSE $R:=E*Z2	$2$$R:= E*Z2;
WRITE(‘$R =’,$R)	$7(`$R=`,$$R)
END;	$3;

Изложенные в этой главе простейшие способы сжатия данных широко применялись в качестве самостоятельных процедур в конце 70-х - середине 80-х годов для архивации данных и в коммуникационных системах. Однако они не позволяют полностью использовать избыточность источника сообщений, что ограничивает их самостоятельное применение. Существенный эффект сжатия данных наблюдается при комбинировании этих способов с оптимальным неравномерным кодированием, принципы которого рассмотрены в последующих разделах.

Контрольные вопросы к главе 1

1. Какие показатели используются для оценки степени сжатия сообщений?
2. В каких случаях можно осуществлять сжатие сообщений с частичной потерей информации?
3. За счет чего осуществляется сжатие при использовании метода *RLE* ?
4. Как зависит коэффициент сжатия в системах, использующих бит-индикаторы, от статистических параметров сообщения?
5. Каким образом уменьшают объем сообщений, состоящих из последовательности цифр?
6. В каких сообщениях целесообразно применять способы сжатия путем замены шаблонов?
7. В результате процедуры компрессии объем исходного сообщения уменьшился с 62-х до 12,4 Кбайт. Определить показатели эффективности сжатия данных $K_{сж}$, $K_{с}$ и $K_{сд}$.

Глава 2

ЭЛЕМЕНТЫ ТЕОРИИ КОДИРОВАНИЯ ИСТОЧНИКОВ

2.1. Характеристики источника дискретных сообщений

Классические методы компрессии данных используют естественную избыточность источников дискретных сообщений. Применяя специальные методы кодирования символов, вырабатываемых источником, устраняют избыточность сообщения, существенно снижая тем самым объем исходных данных. При теоретических исследованиях процессов сжатия информации используют модель источника, называемого *дискретным стационарным эргодическим источником*. Такой источник генерирует дискретную последовательность символов из ограниченного числа элементов, удовлетворяющую условиям стационарности и эргодичности. *Стационарность* означает, что вероятности отдельных символов не зависят от времени, т.е. от расположения их по длине сообщения. Последовательность является *эргодичной*, если статистические характеристики некоторого достаточно длинного отрезка сообщения с вероятностью, близкой к единице, справедливы для всех сообщений, создаваемых источником. В дальнейшем, для краткости, если не будет особых оговорок, будем называть его просто источник либо дискретный источник.

Под дискретным источником на практике понимают устройство, выбирающее дискретные последовательности a_i (i = 0, 1, 2, ... , m-1) из ограниченного числа m элементов, принадлежащих некоторому алфавиту A. Величина m называется объемом алфавита источника. Каждый элемент сообщения появляется на выходе источника с вероятностью $\mathrm{P}(a_i)$, причем $\sum \mathrm{P}(a_i) = 1$. Количество информации $\boldsymbol{I}(a_i)$, содержащееся в символе a_i, определяется выражением

$$I(a_i) = -\log_2 P(a_i). \qquad (2.1)$$

Основание логарифма может быть произвольным. Оно определяет лишь систему единиц измерения количества информации. Наиболее часто используется двоичная система. Количество информации при этом измеряется в битах.

Среднее количество информации *H(A),* приходящееся на один символ, генерируемый источником независимых дискретных сообщений, определяется как математическое ожидание дискретной случайной величины $I(a_i)$ и характеризует количество информации, содержащееся в одном случайно выбранном символе

$$H(A) = \overline{I(a_i)} = -\sum_{i=1}^{m} P(a_i) \log_2 P(a_i) . \qquad (2.2)$$

Величина *H(A)* называется энтропией источника независимых сообщений. Энтропия *H(A)* характеризует первичный алфавит ***A*** источника и является независимой информационной характеристикой. Источник, у которого символы на выходе появляются независимо друг от друга, получил название источника без памяти. Энтропия такого источника при равновероятном появлении символов, когда $P(a_i) = P = 1/m$ для всех $i = 0, \ldots, m-1$, достигает максимума

$$H_{\max}(A) = -mP \log_2 (1/m) = \log_2 m.$$

В дальнейшем, для упрощения записи, основание логарифма 2 будет опускаться.

Для учета имеющей место на практике статистической взаимозависимости появления символов на выходе источника, характеризуемой корреляционной связью между символами сообщения, вводят понятие условной энтропии

$$H(A / A') = -\sum_{i=1}^{m} P(a_i) \sum_{j=1}^{m} P(a_j / a_i) \log P(a_i / a_j) , \qquad (2.3)$$

где $P(a_j / a_i)$ - вероятность появления символа a_j при условии, что перед ним на выходе источника был символ a_i. Протяженность статистической связи между символами сообщений характеризует глубину памяти источника. Источник считается стационарным, если условное распределение вероятностей символов a_i при заданных предшествующих ему k символов не зависит от i, т. е.

$$P(a_i / a_{i-1}, a_{i-2}, ..., a_{i-k}) = P(a_j / a_{j-1}, a_{j-2}, ..., a_{j-k}). \quad (2.4)$$

Условная энтропия — это среднее количество информации, которое содержит один символ сообщения, при условии, что о другом символе получена вся информация и существуют корреляционные связи между соседними символами в сообщении. Из-за корреляционных связей между символами и неравновероятностью их появления в реальных сообщениях уменьшается количество информации, которое переносит один символ. Численно эти потери информации характеризуются коэффициентом избыточности:

$$r = [H_{\max}(A) - H(A)] / H_{\max}(A) = 1 - H(A) / \log m. \quad (2.5)$$

Обратите внимание, что если энтропия определяет информационную нагрузку на символ сообщения, то избыточность - недогруженность символов.

Другим важным информационным параметром источника дискретных сообщений является его производительность $H'(A)$, определяемая средним количеством информации, выдаваемом источником в единицу времени T.

$$H'(A) = H(A) / T \ . \quad (2.6)$$

Избыточность источника нельзя рассматривать как признак его несовершенства. Обычно она является следствием физических свойств источника. Устранение избыточности сообщения заметно уменьшает объем носителя, требуемого для хранения информации, либо позволяет увеличить скорость передачи и приводит к существенному экономическому выигрышу.

2.2. Марковские источники

Во многих практических случаях встречаются источники с памятью, т. е. у которых вероятность выбора символа в данный момент времени зависит от того, какие знаки были выбраны источником в предыдущий момент времени. Поскольку такая статистическая связь распространяется на ограниченное число предыдущих символов, то для математического описания источников используются дискретные цепи Маркова [4,16,19].

Цепь Маркова порядка n характеризует последовательность событий, вероятности которых зависят от того, какие n событий предшествовали данному. Эти n конкретных событий определяют состояние источника, в котором он находится при выдаче очередного знака. Состояние марковского источника изменяется после выдачи им очередного символа. Новое состояние S_{i+1} однозначно определяется старым S_i состоянием и очередным символом. Обычно состояние источника дискретных сообщений изменяется в фиксированные моменты времени t_k (такты). Такой источник называют синхронным. Таким образом, если марковский источник генерирует символы a_j (j = 1, 2, ... , m), где m - размер алфавита, и источник имеет Z состояний S_i (i = 1, 2, ... , Z), то для описания его работы необходимо задать матрицу - строку $P(S_i)$ начального распределения вероятностей состояний источника, а также матрицу условных вероятностей $P(a_j / S_i)$ генерирования источником символов a_j при нахождении его в состоянии S_i . Эти условные вероятности называют переходными вероятностями источника из состояния S_i в состояние S_j. Обозначим для краткости $P(a_j / a_i) = P_{ij}$. Тогда матрица переходных вероятностей будет иметь вид:

$$\begin{vmatrix} P_{11} & P_{12}\dots & P_{1m} \\ P_{21} & P_{22}\dots & P_{2m} \\ \dots\dots & & \dots \\ P_{m1} & P_{m2} & P_{mm} \end{vmatrix}. \qquad (2.7)$$

Находясь в любом состоянии источник генерирует один из символов используемого алфавита. Поэтому сумма элементов строк матрицы (2.7) всегда равна 1.

Энтропия марковского источника определяется выражением:

$$H(A) = -\sum_{j=1}^{z} P(S_j) \sum_{i=1}^{m} P(a_i / S_j) \log_2 P(a_i / S_j). \qquad (2.8)$$

Если символы на выходе источника появляются независимо, т. е. статистические связи отсутствуют, то после выбора источником символа a_i его состояние не изменится (Z=1). Следовательно, $P(S_1) = 1$ и выражение (2.8) превращается в (2.2). Когда корреляционные связи наблюдаются только между двумя символами (цепь Маркова первого порядка),

максимальное число различных состояний источника равно объему алфавита m. Следовательно, $P(a_j/S_q) = P(a_j/a_q)$, где q=1,..., m. При этом формула (2.8) принимает вид (2.3). Средняя энтропия на символ, так называемая марковская энтропия первого порядка, определяется как среднее значение энтропии совместного появления символов a_i и a_j, т. е.

$$H_{M1} = \frac{1}{2} H(a_i, a_j) = \frac{1}{2}\left[H(a_i) + H(a_j / a_i)\right]. \qquad (2.9)$$

Марковская энтропия более высокого (n-го) порядка определяется соответственно как средняя энтропия на генерируемый символ

$$H_{M_2} = \frac{1}{n+1} H(a_1, a_2, \ldots, a_{n+1}) \le \frac{1}{n+1} \sum_{i+1}^{n+1} H(a_i), \qquad (2.10)$$

причем знак равенства в этом уравнении соответствует для независимых символов.

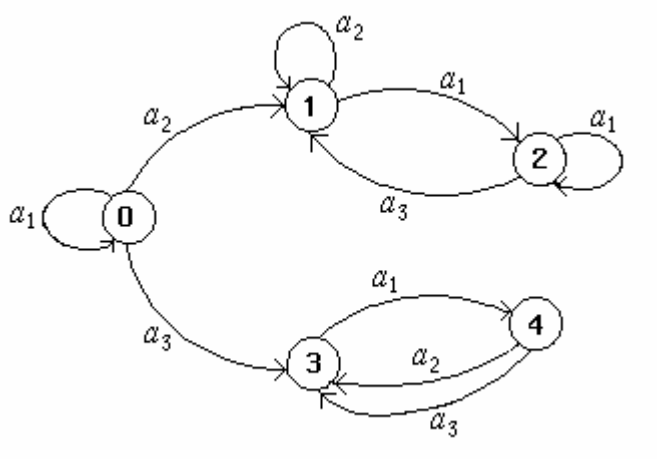


Рис.2.1. Граф марковского источника

Функционирование источника можно представить в виде направленного графа. На рис.2.1 показан граф марковского источника, имеющего четыре состояния, алфавит состоит из трех символов. Каждый узел графа отображает собой состояние (S_1, S_2, S_3, S_4), направленная ветвь соответствует символам источника и указывает изменение состояния (в направлении стрелки), произошедшее в результате генерирования одного из символов алфавита (a_1, a_2, a_3). Исходное состояние, из которого начинается функционирование источника, обозначено S_o. Если для

каждого S_i -го состояния марковского источника заданы условные вероятности появления символов $P(a_k / S_i)$ для k = =1, ... , m, то любая вероятность $P(S_j / S_i)$, приводящая к переходу источника из состояния S_i в S_j , может быть вычислена суммированием условных вероятностей символов, приводящих к этому переходу. Например, для источника, граф которого изображен на рис.2.1,

$$P(S_2 / S_1) = P(a_1 / S_1); \quad (2.11)$$

$$P(S_3 / S_4) = P(a_2 / S_4) + P(a_3 / S_4). \quad (2.12)$$

Очевидно, что $P(S_3 / S_4) = 1$, так как переход из S_4 в S_3 произойдет обязательно при генерации источником любого символа. Матрица условных вероятностей $P(a_k / S_i)$ полностью описывает источник и ансамбль сообщений, образованных им последовательностями символов. Если вероятность возможного возвращения в состояние S_i из любого другого за один или несколько переходов равна 1, то такое состояние называется "возвратным". В противном случае состояние называется невозвратным. На рис.2.1 все состояния, кроме S_o, возвратны. Множество состояний источника называют "замкнутым", если переходы за один шаг из любого состояния этого множества могут вести только в состояние этого же множества. Марковская цепь называется *"неразложимой",* если никакое подмножество ее состояний не замкнуто. В противном случае цепь называется *"разложимой".* Так для источника (Рис.2.1) состояние 1 и 2 образуют одно неразложимое множество, а состояния 3 и 4 - второе. Конечные цепи Маркова, используемые для описания дискретного источника с памятью, обладают рядом свойств, важнейшими из которых являются следующие:

Состояния любой конечной однородной цепи Маркова могут быть однозначно разбиты на одно или большее число неразложимых множеств состояний и множество (может быть пустым) невозвратных состояний. С вероятностью 1 цепь в конце концов оказывается в одном из неразложимых множеств и остается в нем. Если некоторое подмножество состояний образует неразложимую подцепь, то эта подцепь может изучаться независимо от всех других состояний.

Число переходов, начиная с некоторого состояния S_i неразложимого множества, требующегося для первого возвращения в S_i, является случайной величиной, которая называется временем возвращения в S_i. Периодом неразложимого множества состояний называется наибольшее целое число M такое, что все возможные времена возвращений для состояний этого множества являются кратными M. Например, период

множества состояний 1 и 2 на рис.2.1 равен 1, так как время возвращения для любого состояния может быть любым положительным целым числом. Период состояний 3 и 4 равен 2, в связи с тем, что время возвращения равно 2 для каждого состояния. Если неразложимое множество имеет период $M \geq 2$, то оно называется *периодическим*, а если $M = 1$, то множество называется *эргодическим*. Марковский неразложимый дискретный источник с конечным числом состояний Z в любой момент времени находится в одном из возможных состояний. Поэтому можно записать, что

$$\sum_{j=1}^{Z} P(S_j) = 1 . \qquad (2.13)$$

При этом в последующий момент времени источник может перейти в одно из других состояний либо сохранить прежнее состояние. В связи с этим для него справедливо следующее равенство:

$$\sum_{j=1}^{Z} P(S_j / S_i) = 1 \quad . \qquad (2.14)$$

Пример 2.1. Дискретный Марковский источник с памятью первого порядка генерирует последовательность символов, принадлежащих алфавиту, состоящему из трех символов $A\{ a, b, c \}$. В каждый тактовый интервал на выходе источника появляется очередной символ. Пусть для текущего состояния источника имеется распределение вероятностей генерации символов $P(a)$, $P(b)$ и $P(c)$, причем $P(a)+P(b)+P(c)=1$. Распределение вероятностей появления последующих символов после выдачи источником предыдущего задано в виде матрицы переходных вероятностей

	a	*b*	*c*
a	1/3	1/3	1/3
b	1/4	1/2	1/4
c	1/4	1/4	1/2

. (2.15)

На этой матрице текущее состояние (символ) обозначается буквой слева, а следующее состояние - буквой над матрицей.

Необходимо построить граф переходов марковского источника, вычислить распределение вероятностей выдачи источником букв на втором и третьем тактах, а также определить распределение переходных вероятностей (предельную матрицу) для некоторого бесконечного такта.

Построим граф переходов заданного марковского источника (рис.2.2).

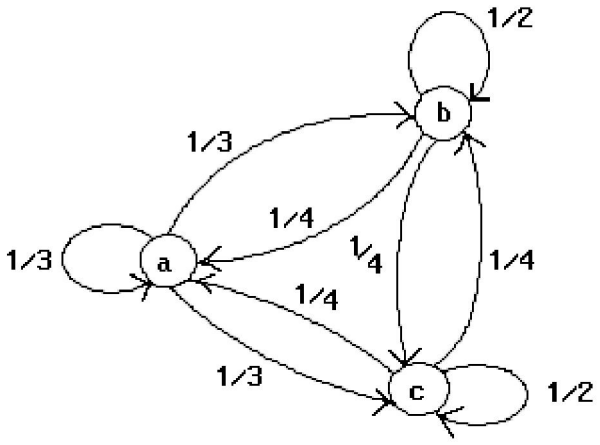


Рис.2.2. Граф переходов марковского источника

Для определения вероятностей символов на выходе источника во втором такте, при условии, что вероятности появления их в первом такте задаются матрицей строкой *[P(a), P(b), P(c)],* необходимо умножить этот вектор-строку справа на матрицу переходных вероятностей (2.15).

$$[P(a),\ P(b),\ P(c)] \times \begin{vmatrix} 1/3 & 1/3 & 1/3 \\ 1/4 & 1/2 & 1/4 \\ 1/4 & 1/4 & 1/2 \end{vmatrix} = [P(a)/3 + P(b)/4 +$$

$$+ P(c)/4,\ P(a)/3 + P(b)/2 + P(c)/4,\ P(a)/3 + P(b)/4 + P(c)/2] .$$

Чтобы вычислить распределение вероятностей на третьем такте, нужно еще раз умножить полученный вектор на матрицу переходных вероятностей (2.15). Поскольку произведение матриц ассоциативно, то сначала можно умножить матрицу на себя, получить квадрат матрицы, а затем умножить распределение вероятностей справа на этот квадрат. Квадрат матрицы переходных вероятностей равен:

$$\begin{vmatrix} 1/3 & 1/3 & 1/3 \\ 1/4 & 1/2 & 1/4 \\ 1/4 & 1/4 & 1/2 \end{vmatrix} \times \begin{vmatrix} 1/3 & 1/3 & 1/3 \\ 1/4 & 1/2 & 1/4 \\ 1/4 & 1/4 & 1/2 \end{vmatrix} = \begin{vmatrix} 10/36 & 13/36 & 13/36 \\ 13/48 & 19/48 & 16/48 \\ 13/48 & 16/48 & 19/48 \end{vmatrix} .$$

Выделим общий множитель элементов матрицы и представим ее в виде:

$$(1/144) \times \begin{vmatrix} 40 & 52 & 52 \\ 39 & 57 & 48 \\ 39 & 48 & 57 \end{vmatrix}. \qquad (2.16)$$

Распределение вероятностей на третьем такте представляется матрицей-строкой: (1/144)[*P(a)*/40 +*P(b)*/39+*P(c)*/39, *P(a)*/52+ *P(b)*/57 + *P(c)*/48, *P(a)*/52 + *P(b)*/48 + *P(c)*/57]. Анализируя матрицу (2.14), можно заметить, что элементы квадрата матрицы переходных вероятностей изменяются не так сильно, как элементы исходной матрицы. Если для нахождения распределения вероятностей на следующем (четвертом) такте эту матрицу умножить еще раз на переходную матрицу, то разница между элементами результирующей матрицы будет еще менее заметна, то есть происходит стабилизация вероятностей.

Элементы предельной матрицы не должны изменяться от такта к такту работы источника. Для частного случая матрицы (2.15) должно выполняться следующее равенство:

$$[P(a), P(b), P(c)] \times \begin{vmatrix} 1/3 & 1/3 & 1/3 \\ 1/4 & 1/2 & 1/4 \\ 1/4 & 1/4 & 1/2 \end{vmatrix} = [P(a), P(b), P(c)].$$

Умножим строку на матрицу и результат представим в виде трех уравнений:

$$\begin{aligned} P(a)/3 + P(b)/4 + P(c)/4 &= P(a) \\ P(a)/3 + P(b)/2 + P(c)/4 &= P(b) \\ P(a)/3 + P(b)/4 + P(c)/2 &= P(c)\ . \end{aligned} \qquad (2.17)$$

В (2.17) только два уравнения являются независимыми. Поэтому, выделив два первых из них и дополнив его третим уравнением, вытекающим из условия задачи $P(a) + P(b) + P(c) = 1$, решая эту систему, находим предельное (*стационарное*) распределение вероятностей.

$$P(a) = 3/11;\ \ P(b) = P(c) = 4/11.$$

Таким образом, независимо от первоначального распределения вероятностей, для матрицы переходных вероятностей (2.15) получаем одно и то же распределение вероятностей [3/11, 4/11, 4/11].

Марковский процесс, при котором из произвольного состояния можно перейти в любое другое состояние и который с течением времени переходит к предельному распределению, не зависящему от начального состояния, является эргодическим. Источник, описываемый таким

марковским процессом, называется эргодическим стационарным источником. Рассмотренный выше источник является примером эргодического источника.

2.3. Кодовые деревья

Для наглядности и удобства построения алгоритмов кодирования и декодирования изображение множества кодовых комбинаций (слов) осуществляется в виде графа, состоящего из узлов и ветвей, соединяющих узлы, расположенные на разных уровнях. Такой граф носит название "кодовое дерево". Началом кодового дерева является корень. Из корня исходит m ветвей, образующих первый уровень дерева. Каждая ветвь заканчивается узлом из которого, в свою очередь, могут выходить m ветвей второго уровня и т.д. Величина m называется *степенью* дерева и численно равна значности отображаемого кода. Узлы, в которых происходит разделение ветвей, называют *узлами ветвления*, а оконечные узлы, из которых не исходит ни одной ветви - концевыми, *терминальными* или *листьями* дерева. Терминальные узлы являются внешними, а узлы ветвления - внутренними узлами дерева. Внутренний узел дерева часто называют *родителем*, а исходящие из него узлы - потомками или *дочерними* узлами.

Любой уровень дерева в общем случае содержит m^k узлов, где k-номер уровня. Уровень узла k определяется количеством ветвей дерева, насчитываемых при движении от корня к заданному узлу. Корень имеет нулевой уровень, а уровень любого другого узла дерева на 1 больше уровня своего родителя. Ветви, исходящие из узлов, нумеруют различными m-ичными буквами 0, 1, 2, ..., m-1. Обычно принято обозначать крайнюю левую ветвь нулем, хотя это условие не является обязательным. Обозначение ветвей соответствует выбору между символами алфавита на определенном уровне. Ветви первого уровня определяют первый символ кодового слова (*старший разряд*), второго уровня - второй и т.д.

Если степень дерева m=2, то оно называется двоичным (*бинарным*). В этом случае из корня исходит две ветви, которые образуют левое и правое поддерево. Бинарные деревья используются для анализа и построения двоичных кодов. Деревья степени больше двух называют сильноветвящимися деревьями (*multiway trees*). Так как на практике в основном применяются двоичные коды, то наиболее широкое распространение получили бинарные деревья, хотя во многих работах по теории информации и кодирования результаты анализа зачастую обобщаются на m-ичные коды с применением m-ичных кодовых деревьев.

Одним из основных параметров кодового дерева является его *высота* $h=k_{max}$, определяемая максимальным уровнем концевого узла. Дерево высотой h является полным, если количество его листьев равно m^h. На рис.2.3 изображено полное двоичное дерево, а на рис.2.4 - троичное. Здесь кружками обозначены узлы ветвления, а прямоугольниками - листья. Возле каждого узла дерева проставлены соответствующие ему кодовые комбинации. Как видно из рисунков, для отображения кодовых слов кода используются листья дерева. Причем вид кодовой комбинации определяется значениями ветвей дерева.

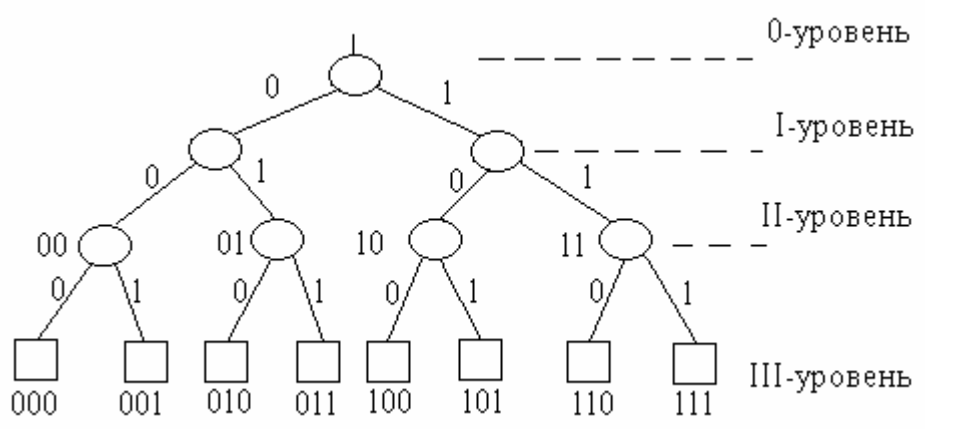


Рис.2.3. Полное двоичное кодовое дерево

Как уже отмечалось выше, ветвь первого уровня отображает старший разряд кодового слова.

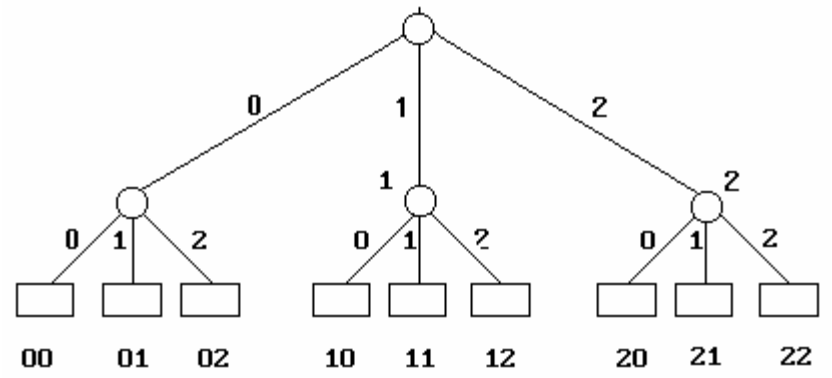


Рис.2.6. Полное троичное кодовое дерево

Другим важным параметром кодового дерева является его объем D, численно равный максимальному количеству кодовых комбинаций, которые могут быть отображены с помощью данного дерева, т.е. $D=m^h$. Очевидно, что двоичное кодовое дерево, используемое для кодирования источника, генерирующего m_1 символов, содержит m_1 терминальных

(внешних) и (m_1 - 1) внутренних (промежуточных) узлов. Общее количество узлов дерева равно $2m_1$ - 1. Число ветвей l_i , которое нужно пройти от корня дерева к листу Z_i называется длиной пути к Z_i. Корень имеет путь 0, его прямые потомки - путь 1 и т.д. Вообще терминальный узел на уровне k имеет длину пути $l_k=k$. Длина пути всего дерева определяется суммой путей для всех его узлов. Её также называют длиной внутреннего пути. Так длина внутреннего пути для дерева, изображенного на рис.2.3, равна 34. Средняя длина пути равна

$$l_{cp} = (\sum_k n_k k) / N_t \ , \qquad (2.18)$$

где n_k - число узлов на k-том уровне; $N_t = 2m_1 - 1$ - общее количество узлов дерева. Длина внешнего пути определяется как сумма длин путей внешних узлов. Для дерева (рис.2.3) она равна 24. Если число внешних вершин на уровне k равно n_{Tk}, а общее количество терминальных узлов составляет N_T, то средняя длина внешнего пути равна

$$l_{Tcp} = \sum_k n_{Tk} k) / N_T \ . \qquad (2.19)$$

Кодовые деревья могут использоваться для отображения кодов с одинаковым числом элементов в кодовых словах (*равномерных кодов*), так и для отображения неравномерных кодов, использующих кодовые слова с различным количеством символов в слове. При этом все терминальные узлы дерева равномерного кода расположены на одном уровне, а неравномерного - на разных. Примеры кодовых деревьев неравномерного кода изображены на рис.2.5 . Если для каждого узла дерева высота его двух поддеревьев различается не более чем на 1, то такое дерево называется *сбалансированным* или *АВЛ*-деревом, названным так по инициалам математиков, которые ввели это определение (Адельсон, Вельский, Ландис).

Двоичное дерево называют *почти полным бинарным* при выполнении следующих условий:

1) уровни листьев в дереве различаются не более чем на 1;
2) если любой узел дерева имеет правого потомка уровня k, тогда все его левые потомки , являющиеся листьями, имеют уровень не менее k.

Деревья, представленные на рис.2.5а,б не являются почти полными, так как в дереве на рис.2.5,а не выполняется условие 1), а в дереве на рис.2.5,б не

выполняется условие 2). Бинарное дерево (рис.2.5,в) является почти полным.

Узлы почти полного бинарного дерева могут быть занумерованы так, что корню назначается номер 1, левому дочернему узлу - удвоенный номер его родителя, правому - удвоенный номер его родителя плюс 1 (см. рис.2.3 и 2.5,в.). При таком порядке нумерации каждому узлу присвоен уникальный номер, который определяет позицию узла внутри дерева.

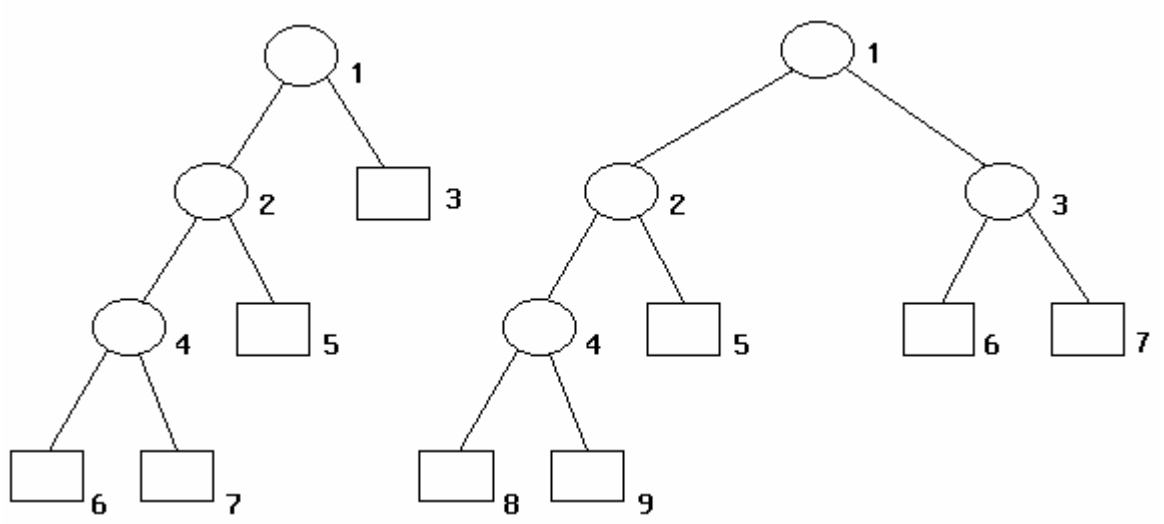


Рис.2.5. Примеры деревьев неравномерного кода

При построении и анализе различных кодов используют *вероятностные кодовые деревья*. Под этим понятием подразумевается

кодовое дерево с неотрицательными числами (вероятностями), присвоенными каждому узлу так, что выполняются следующие условия:

1) корневому узлу присваивается вероятность, равная 1;

2) вероятность каждого родительского узла (включая корень) равна сумме вероятностей дочерних узлов;

3) средняя длина внешнего пути вероятностного кодового дерева равна сумме произведений вероятности *i*-го терминального 0 узла P_i на длину пути l_{Ti} к этому узлу

$$l_{Tcp} = \sum_{i=1}^{n_T} P_{Ti} l_{Ti} \quad . \qquad (2.20)$$

При этом дерево не обязательно должно быть *m*-ичным, т. е. иметь одинаковое число ветвей, исходящих из всех промежуточных узлов. На рис.2.6 изображено вероятностное дерево, которое не является ни двоичным, ни троичным. Обратите внимание, что в вероятностном кодовом дереве сумма вероятностей терминальных узлов должна быть равна единице.

Для вероятностного дерева справедлива *лемма 1*, утверждающая, что средняя длина внешнего пути равна сумме вероятностей промежуточных узлов, включая корень,

$$l_{Tcp} = \sum_{i=1}^{N_B} q_i \quad . \qquad (2.21)$$

Здесь N_B - число внутренних узлов вероятностного дерева; q_i - вероятность *i*-го внутреннего узла.

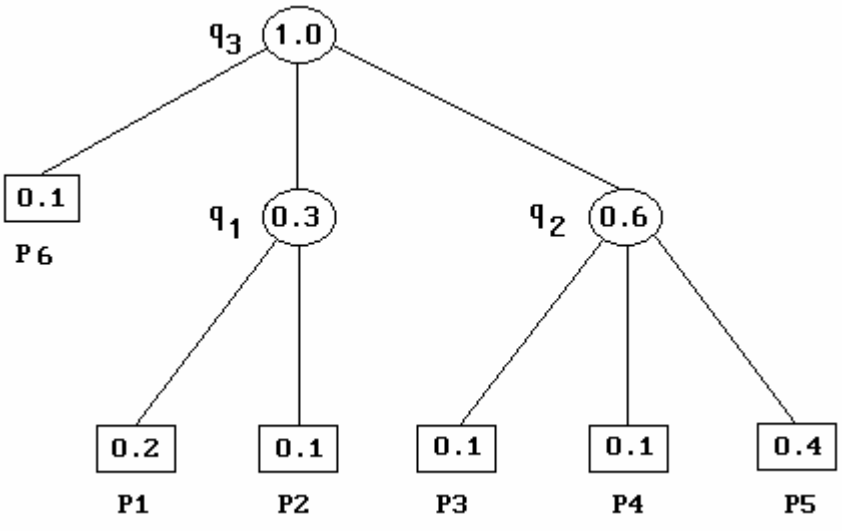


Рис.2.6. Вероятностное кодовое дерево

С учетом того, что вероятность корня равна сумме вероятностей терминальных узлов, выражение (2.21) может быть записано в виде суммы вероятностей всех узлов, за исключением корня

$$l_{Tcp} = \sum_{i=1}^{N_B=1} q_i + \sum_{j=1}^{N_T} P_j = \sum_{k=1}^{2N_T-2} Q_k \quad , \qquad (2.\ 22)$$

где Q_k -вероятность k-го узла дерева (внутреннего или терминального).

Доказательство. Вероятность каждого внутреннего узла равна сумме вероятностей терминальных узлов в поддереве, исходящих из этого узла. Но терминальный узел, расположенный на расстоянии d от корня дерева, входит в качестве составляющей вероятности каждого из внутренних узлов (включая корень) на пути от этого листа до корня. Поэтому сумма вероятностей всех внутренних узлов дерева равняется сумме произведений вероятности каждого листа и его высоты. А эта сумма и является средней длиной внешнего пути дерева.

В вероятностном кодовом дереве (рис.2.6) сумма вероятностей внутренних узлов равна 0,3 + 0,6 + 1 = 1,9 , что согласно лемме 1 представляет среднюю длину внешнего пути. Для проверки произведем вычисление l_{Tcp} = 0,1×1 + 0,2 × 2 + 0,1×2+ 0,1×2 + 0,1×2 + 0,4×2 = 1,9 .

Вероятность любого узла дерева можно интерпретировать как вероятность достижения этого узла при случайном перемещении по дереву от корня к некоторому листу. Если эта вероятность относится к узлу ветвления, то условную вероятность выбора каждой выходящей ветви в качестве следующего направления движения можно определить путем деления вероятности узла в конце ветви на вероятность узла в начале (т.е. узла ветвления). Так, для примера (рис.2.6) вероятности продвижения к терминальным узлам с вероятностями 0,2 и 0,1 от узла ветвления с вероятностью 0,3 будут равны 2/3 и 1/ 3 соответственно.

Для вероятностного кодового дерева, имеющего N_T терминальных узлов с вероятностями P_1, P_2 , ... , P_{N_T} можно определить его энтропию (*терминальную неопределенность*) по формуле (2.2)

$$H_T = -\sum_{i=1}^{N_T} P_i \log P_i \ . \qquad (2.23)$$

Рассмотрим вероятностное кодовое дерево с N_B внутренними узлами, вероятности которых равны q_1, q_2,..., q_{N_B}. Найдем энтропию ветвей в каждом узле ветвления, которая равна энтропии случайной

переменной, определяющей ветвь дерева, исходящей из этого узла , при условии, что узел был достигнут при случайном прохождении по дереву. Предположим, что P_{i1}, $P_{i\,2}$, ... , $P_{i\,M}$ - вероятности узлов (одни из них могут быть промежуточными, а другие - терминальными), расположенных на концах M ветвей, исходящих из i-го промежуточного узла, имеющего вероятность q_i . Т.к. условная вероятность выбора j-й из этих ветвей в качестве следующего направления движения, при условии, что находимся в узле с вероятностью q_i равна $P_{ij}/q_{\,i}$, то энтропия ветвей H_i определится по формуле:

$$H_i = -\sum_{j=1}^{M} (P_{ij} / q_i) \log(P_{ij} / q_i) \quad . \qquad (2.24)$$

Используя (2.24) с учетом того, что в соответствии со свойством вероятностного кодового дерева сумма P_{ij} по j равна q_i, а $\log(P_{ij}/q_{\,i}) = \log P_{i\,j} - \log q_i$, определим произведение

$$q_i H_i = -\sum_{j=1}^{M} P_{ij} \log P_{ij} + q_i \log q_i \quad . \qquad (2.25)$$

Просуммируем произведение $q_i H_i$ по i = 1, 2, ... , $N_{\text{В}}$ по всем внутренним узлам дерева. Из (2.25) видно, что при $i = k$ внутренний k-й узел, если он не является корневым, вносит в сумму составляющую + $q_k \log q_k$, а также член $q_k \log q_k$ со знаком минус для такого i, при котором $P_{ij}=q_k$ (т. е., составляющую для i, для которого узел k находится на конце ветви, выходящей из внутреннего узла i). Следовательно, общий вклад всех внутренних узлов в сумму $\sum q_i H_i$ равен нулю, включая и корень, так как $q_1 = 1$. Таким образом, можно сделать вывод, что k-й терминальный узел вносит в сумму произведения $q_i H_i$ составляющую - $q_k \log q_k$, так как он влияет только на составляющую, для которой i такое, что $P_{i\,j} = q_k$, а сумма произведения

$$\sum_{i=1}^{N_B} q_i H_i = -\sum_{k=1}^{N_T} P_k \log P_k = H_T \quad . \qquad (2.26)$$

Т. е. терминальная энтропия вероятностного дерева равняется сумме энтропии ветвей всех внутренних узлов дерева, включая корень, взвешенной с вероятностью соответствующих узлов:

$$H_T = \sum_{i=1}^{N_B} q_i H_i \ . \qquad (2.27)$$

Пример 2.2. Вычислим терминальную энтропию и энтропию ветвей для дерева, изображенного на рис.2.6 . Здесь количество терминальных узлов $N_т = 6$, а число внутренних - $N_B = 3$.

Пронумеруем узлы в порядке слева направо, снизу вверх, то есть $P_1 = 0{,}2$; $P_2 = 0{,}1$; $P_3 = 0{,}1$; $P_4 = 0{,}1$; $P_5 = 0{,}4$; $P_6 = 0{,}1$; $q_1 = 0{,}3$; $q_2 = 0{,}6$; $q_3 = 1$. Тогда энтропия дерева

$$H_т = - \sum_i P_i \log P_i = - (0{,}2 \times \log 0{,}2 + 0{,}1 \times \log 0{,}1 + 0{,}1 \times \log 0{,}1 + 0{,}4 \times \log 0{,}4 + 0{,}1 \times \log 0{,}1) = 2{,}322 \ .$$

Как видно из рис.2.6, из первого внутреннего узла исходит $m_1 = 2$ ветви с вероятностями $P_{11} = 2 / 3$ и $P_{12} = 1 / 3$. Тогда энтропия ветвей этого узла равна

$$H_1 = - (2/3) \log (2/3) - (1/3) \log (1/3) = 0{,}9183 \ .$$

Из второго внутреннего узла исходит $m_2 = 3$ ветви с вероятностями $P_2 = 1/ 6$; $P_{22} = 1/ 6$; $P_{23} = 4/ 6$. Следовательно,

$$H_2 = - (1/6) \log (1/6) - (1/6) \log (1/ 6) - (4/ 6) \log (4/ 6) = 1{,}2516 \ .$$

Для третьего, корневого, узла $P_{31} = 0{,}1$; $P_{32} = 0{,}3$ и $P_{33} = 0{,}6$, а его энтропия $H_3 = - 0{,}1 \log 0{,}1 - 0{,}3 \log 0{,}3 - 0{,}6 \log 0{,}6 = 1{,}2955$.

Вычислим терминальную энтропию вероятностного дерева по формуле (2.27). $H_т = 0{,}3 \times 0{,}9183 + 0{,}6 \times 1{,}2516 + 1{,}2955 = 2{,}322$, что соответствует величине, определенной при прямом вычислении H_T .

В процессе кодирования и декодирования информации кодер и декодер осуществляют построение кодового дерева, в ходе которого дерево растет или сокращается, меняет свою структуру. Основными операциями с деревьями при построении и анализе кодов являются поиск с включением или исключением узлов, изменение местоположения узлов, а также

упорядочивание структуры по определенным правилам. Особенности алгоритмической и программной реализации операций с деревьями подробно рассмотрено в специальной литературе [2,11].

2.4. Кодирование источников неравномерными кодами

Под кодированием источника понимается процедура представления символов дискретного множества сообщений, вырабатываемых источником информации, принадлежащих некоторому алфавиту $A(a_1, a_2,...,a_{m1})$ объемом m_1, кодовыми словами, представляющими собой комбинации элементов (символов), принадлежащих алфавиту $Z(z_1, z_2, ...,z_{m2})$, объемом m_2. Алфавит символов источника A называют *первичным*, а алфавит элементов кодовых слов Z - *вторичным*. Необходимым условием декодирования является взаимное однозначное соответствие кодовых слов во вторичном алфавите кодируемым символам первичного алфавита источника. Коды, в которых сообщения представлены комбинациями с одинаковым числом символов, называются *равномерными*, а коды, содержащие комбинации с различным числом символов в кодовых словах - *неравномерными*.

Основной задачей при кодировании источников является построение такого кода, у которого число элементов вторичного алфавита, затрачиваемое на кодирование одного символа источника, будет *минимальным*. Процедура такого кодирования называется *оптимальным* или *эффективным* кодированием [4,21]. При оптимальном кодировании используются статистические свойства источника, при котором более вероятным символам источника соответствуют самые короткие кодовые комбинации, а менее вероятным - более длинные, т. е., эффект связан с различием в числе элементов (символов) кодовых комбинаций. Но при этом код становится неравномерным, что существенно затрудняет декодирование кодовых слов, так как очень сложно определить границы, отделяющие одну кодовую комбинацию от другой.

Для решения этой проблемы эффективный код необходимо строить так, чтобы ни одна кодовая комбинация не совпадала с началом другой, более длинной комбинацией. Коды, удовлетворяющие такому условию, называют *префиксными* кодами, т. е. код, обладающий свойством префикса - это код, в котором никакое кодовое слово не является начальной частью (*префиксом*) другого. В табл.2.1 приведено два типа неравномерных кодов. Первый из них является префиксным, а второй нет, так как комбинация , отображающая символ a_2, является префиксом

кодового слова, отображающего символ a_3. При поступлении последовательности вида 1000001101101101 00 она однозначно декодируется, в результате чего формируется строка символов a_4 a_1 a_2 a_3 a_3 a_3 a_1. А при декодировании последовательности вида 00010101010101, которая была получена в результате кодирования сообщения кодом второго типа, может быть получена строка символов $a_1\ a_2\ a_2\ a_2\ a_2\ a_2\ a_2$ либо $a_1\ a_4\ a_3\ a_4\ a_3$.

Одним из основных параметров, характеризующих неравномерный код, является *средняя длина* кодовых слов во вторичном алфавите $l_{ср}$. Если дискретный источник без памяти имеет алфавит из m_1 символов $a_1, a_2, \ldots, a_{m1}$ с вероятностями $P(a_1), P(a_2), \ldots, P(a_{m1})$, а вторичный алфавит содержит m_2 различных символов, то

$$l_{cp} = \sum_{i=1}^{m_1} P(a_i) l(a_i), \qquad (2.28)$$

где $P(a_i)$-вероятность появления i-го символа алфавита источника; $l(a_i)$ - длина i -го кодового слова.

Таблица 2.1

Символы	Код I	Код II
a_1	00	00
a_2	01	01
a_3	101	101
a_4	100	010

В работах по теории информации и кодирования [16,19,21] доказан ряд теорем, определяющих связь между энтропией дискретного источника и средней длиной кодового слова.

Теорема 1. При заданных конечном первичном алфавите источника A с энтропией $H(A)$ и вторичном алфавите, состоящем из m_2 символов, можно так приписать кодовые слова символам источника, что будет выполняться свойство префикса и средняя длина кодового слова $l_{ср}$ будет удовлетворять условию:

$$H(A)/\log m_2 \le l_{ср} < [H(A)/\log m_2] + 1. \qquad (2.29)$$

Здесь основание логарифма не играет существенной роли. При оценке количества информации в битах основание принимается равным двум. Выражение (2.29) устанавливает нижнюю и верхнюю границы для $l_{\text{ср}}$.

Еще больше можно приблизить $l_{\text{ср}}$ к энтропии источника, если кодовые слова приписывать не отдельным символам источника, а *строкам*, состоящим из L символов.

Теорема 2. Для заданных дискретного источника без памяти с энтропией $H(A)$ и вторичного алфавита из m_2 символов возможно так закодировать последовательность L символов источника, что будет выполняться свойство префикса и средняя длина кодовых слов на один символ источника $l_{\text{ср}}$ будет удовлетворять условию:

$$H(A) / \log m_2 \le l_{ср} < H(A) / \log m_2 + 1/L . \qquad (2.30)$$

Для двоичных кодов ($m_2 = 2$) выражение (2.29) принимает вид $H(A) \le l_{\text{ср}} < H(A) + 1$. К такому же виду стремится (2.30) при увеличении длины кодируемой строки L.

Возможность построения неравномерного префиксного кода с заданными длинами кодовых слов определяется *неравенством Крафта* [19]: m-ичный префиксный код существует только тогда, если длины кодовых слов являются положительными целыми числами $W_1, W_2, \ldots, W_k$ и выполняется неравенство

$$\sum_{i=1}^{K} m^{-W_i} \le 1 . \qquad (2.31)$$

При доказательстве используют тот факт, что в полном m-ичном дереве высотой h из каждого узла на уровне $W < h$ исходит m^{k-W} терминальных узлов. Предположим, что существует m - ичный префиксный код с длинами кодовых комбинаций $W_1, W_2, \ldots, W_h$. Построим полное m-ичное дерево с высотой $h = \max W_i$.

Рассмотрим процесс создания дерева для искомого кода путем усечения полного m - ичного дерева высотой h. Находим узел Z_i , расположенный на дереве на расстоянии $W_i < h$ от корня, и обрежем ветвь дерева на этом узле с тем, чтобы чтобы образовался терминальный узел. После такой операции из полного дерева удаляется $m^{h - W_i}$ концевых узлов, которые могли бы быть использованы для других кодовых слов. Так как существует только m терминальных узлов, которые можно удалить, то после обрезки всех ветвей будет удалено

$$m^{h-W_1} + m^{h-W_2} + \ldots + m^{h-W_k} \le m^h \qquad (2.32)$$

ветвей. Разделив (2.32) на m^h , получим неравенство Крафта (2.31).

Пример 2.3. Построить двоичный префиксный код с длинами $W_1 = W_2 = W_3 = 2$; $W_4 = 3$ и $W_5 = 3$.

Поскольку $\sum 2^{-i} = 1/4 + 1/4 + 1/4 + 1/8 + 1/16 = 15/16 < 1$, то такой код существует. Для построения кода используем полное двоичное дерево с высотой $h = W_5 = 4$. Отсекая три ветви дерева на втором уровне и одну на третьем, получаем дерево, изобра-женное на рис.2.7.

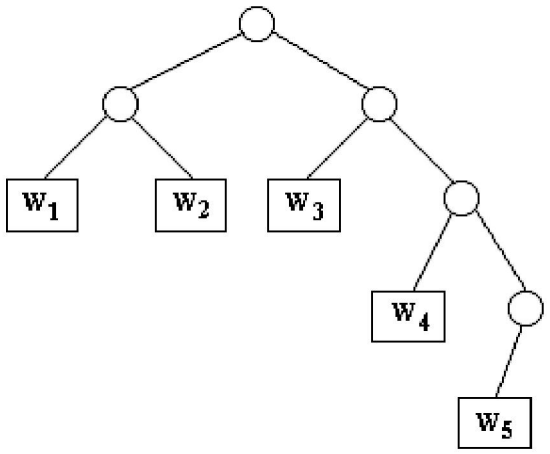


Рис.2.7. Дерево двоичного префиксного кода

Пример 2.4 . Построить двоичный префиксный код с длинами слов $W_1 = 1$; $W_2 = W_3 = 2$; $W_4 = 3$; $W_5 = 4$.

Так как $\sum 2^{-W_i} = 1/2 + 1/4 + 1/4 + 1/8 + 1/16 = 19/16 > 1$, то такого кода не существует. В этом можно убедиться, попытавшись построить кодовое дерево для заданных условий.

Основные выводы, вытекающие из вышеизложенного, могут быть сформулированы следующим образом.

1. Энтропия первичного алфавита *H(A)* и средняя длина кодовых слов во вторичном алфавите $l_{ср}(m_2)$ - величины взаимосвязанные и соизмеримые. Энтропия первичного алфавита может характеризовать возможный предел сокращения кодового слова во вторичном алфавите.

2. Длины кодовых слов префиксного кода должны удовлетворять неравенству Крафта.

3. Средняя длина кодовых слов $l_{ср}$ (m_2) во вторичном алфавите ни при каких условиях не может быть меньше энтропии кодируемого алфавита *H(A).*

4. Качество кода с точки зрения минимальной средней длины кодовой комбинации может быть определено путем сравнения *H(A)* и $l_{ср}(m_2)$. При правильном построении кода эти величины будут отличаться друг от друга не более чем на единицу.

5. При любом количестве символов алфавита источника более целесообразным является кодирование строк. Чем больше символов в строке, тем меньше разность между верхней и нижней границами, определяющими среднее число кодовых элементов на символ сообщения.

6. Разность $l_{ср}$ - $H(A) / \log m_2$ уменьшается с ростом энтропии, которая достигает максимума при равновероятных и взаимонезависимых символах.

Поэтому при построении оптимального кода, у которого средняя длина кодовой комбинации будет минимальной, необходимо добиваться наименьшей избыточности каждого из кодовых слов, которые в свою очередь должны строиться из равновероятных и взаимонезависимых символов.

2.5. Кодирование с предсказанием

Во многих случаях статистическая структура источника дискретных сообщений известна заранее и имеется его модель, например, марковская цепь *k*-го порядка. Используя модель источника, можно предсказать с некоторой вероятностью появление очередного конкретного символа. Метод предсказания может быть фиксированным или адаптивным, при котором алгоритм вычисления изменяется в зависимости от качества предыдущих предсказаний. Конкретные методы предсказаний будут рассмотрены в разделе 7, здесь же лишь предполагается, что имеется некоторый предсказатель *P*, который достаточно правильно предсказывает следующий символ источника.

Сущность кодирования с предсказанием заключается в том, что на вход компрессора подается *сигнал ошибки предсказания*, равный разности между выходами источника и предсказателя. В случае правильно предсказанных символов разностный сигнал представляет собой *поток нулей*, который можно достаточно эффективно сжать с помощью процедуры кодирования длины повторяющихся символов. Схемы компрессии и декомпрессии с предсказанием данных, поступающих с двоичного источника, изображены соответственно на рис. 2.8,*а* и *б*.

Источник *S* выдает в некоторый момент времени *t* очередной *i*-й символ a_i (0 или 1), который поступает на вход вычитателя В. Предсказатель Р на основе группы символов $a_{i-1}, a_{i-2},\ldots,$ сгенерированных источником в предшествующие моменты времени, вычисляет вероятности появления 0 и 1 и выдает на второй вход вычитателя символ a_i с максимальной вероятностью появления. Для двоичных символов роль вычитателя выполняет сумматор по модулю два.

Ошибка предсказания e_n на n-м такте равна 0, если предсказанное значение символа a_i совпало с символом источника a_i, и равна 1, если предсказание было неверным. Предсказатель P на основе e_n исправляет

свои предсказания и к следующему такту точно знает всю предыдущую последовательность символов источника.

Если предсказатель достаточно точный, то последовательность $e_{\text{п}}$ состоит в основном из нулей с редкими единицами между ними. Даже при плохом предсказании число нулей будет существенно больше количества единиц. Случайное предсказание давало бы примерно равное число нулей и единиц [19].

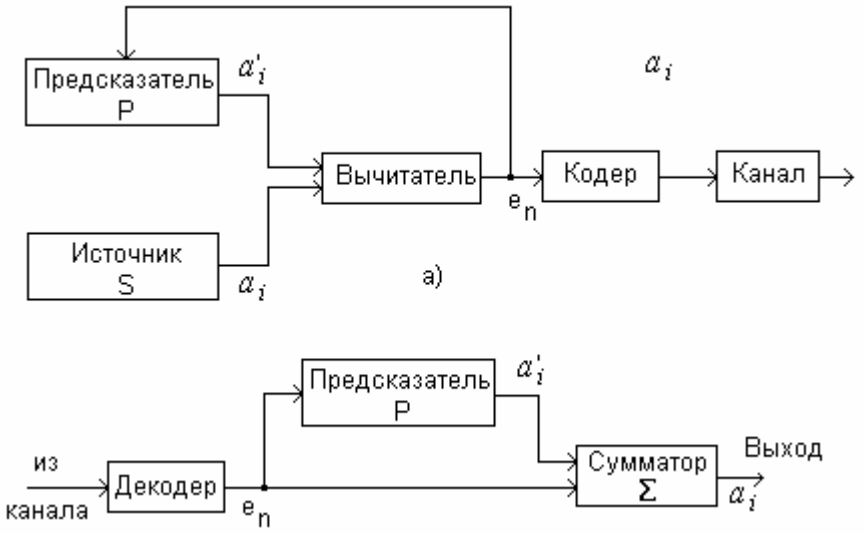


Рис. 2.8. Схема компрессии и декомпрессии данных с предсказанием

В кодере сжатие последовательности нулей производится за счет преобразования их в коды длин серий одинаковых символов либо путем применения неравномерного кодирования. Предсказатель декомпрессора должен в точности совпадать с соответствующим устройством компрессора и работать с ним синхронно. Тогда начиная с одного и того же состояния оба устройства делают одинаковые предсказания. В декодере декомпрессора происходит восстановление сжатой последовательности символов ошибки $e_{\text{п}}$. Когда на выходе декодера появляется единица, она суммируется по модулю 2 в устройстве Σ с предсказанным значением и потребителю выдается правильное решение. В блоке P ошибка прибавляется к предсказанному символу и формируется правильное значение предсказания, которое используется на последующих этапах предсказания.

Для расчета средней длины серий нулей сигнала $e_{\text{п}}$ предположим, что вероятность $p_{\text{п}}$ того, что устройство P правильно осуществило предсказание - величина постоянная, а каждая ошибка предсказания

независима от других ошибок. Тогда вероятность ошибки предсказания $q_{\text{п}} = 1 - p_{\text{п}}$. Естественно считать, что $p_{\text{п}} \geq 1/2$, поскольку иначе можно просто поменять местами $p_{\text{п}}$ и $q_{\text{п}}$, заменяя предсказанные нули на единицы и наоборот. Большое $p_{\text{п}}$ соответствует хорошему кодеру, а $p_{\text{п}}$, близкое к 1/2, означает, что предсказание практически совпадает с простым угадыванием.

Вероятность $p(n)$ серий из n нулей, которая всегда заканчивается единицей, равна $p_{\text{п}}^{n} q_{\text{п}}$, где (n=0,1,...). Величина n может достигать довольно больших значений. Для кодирования длин серий будем использовать двоичное число, представляющее собой кодовую комбинацию, состоящую из k битов. Из всего диапазона значений чисел от 0 до $2^k - 1$ выделим два числа: 0 — для кодирования нулевой длины серии, и $2^k - 1$ — если серия состоит из более чем 2^k - 1 нулей. В этом случае, если длина серии находится между 0 и 2^k - 2, то декодер генерирует указанное число нулей и затем одну 1. Если принято число 0, то декодер выводит лишь одну единицу, а при приеме числа $2^k - 1$ выдает $2^k - 1$ нулей, за которыми не следует 1. Если же после этого принят 0, то, естественно, генерируется 1. Для любого другого числа генерируется указанное количество нулей и затем единица. Таким образом, передав нужное число k-разрядных двоичных чисел, можно определить любую длину серии, состоящей из нулей.

Чтобы найти среднюю длину кодовых комбинаций, используемых для кодирования длин серий, необходимо сложить произведения вероятностей появления серии нулей на количество битов, передаваемых в каждом случае. Таким образом, имеем:

Длина серии	Число передаваемых битов	
$0 \leq n \leq (2^k - 1) - 1$	k	
$2^k - 1 \leq n \leq 2(2^k - 1) - 1$	$2k$	
$2(2^k - 1) \leq n \leq 3(2^k - 1) - 1$	$3k$	и т. д.

Поэтому средняя длина кодовой комбинации равна:

$$k_{\text{ср}} = [p(0) + p(1) + \ldots + p(m-1]\, k + [p(m) + p(m+1) + \ldots$$

$$+p(2m-1)]\, 2k + [p(2m) + p(2m+1) + \ldots + p(3m-1)]3k + \ldots \ ,$$

где $p(n)$ - вероятность появления серии нулей длины n; $m = 2^k - 1$.

Поскольку $p(n) = p_{\text{п}}^{n} q_{\text{п}}$, получаем

$$k_{ср}=q_п[1+p_п+\ldots+p_п^{m-1}]k+q_п p_п^{m}[1+p_п+\ldots+p_п^{m-1}]2k+$$
$$+q_п p_п^{2m}[1+p_п+\ldots+p_п^{m-1}]\,3k+\ldots\ .$$

Вынесем за скобки общий множитель и заменяя ряд его суммой, приводим это выражение к виду:

$$k_{ср}=q_п[(1-p_п^{m})/q_п]\,[1+2p_п^{m}+3p_п^{2m}+\ldots]k\ .$$

Эта сумма в свою очередь равна:

$$k_{\ ср}=q_п[(1-p_п^{m})/q_п](1-p_п^{m})^{-2}\,k=k/(1-p_п^{m})\ . \qquad (2.33)$$

В связи с тем, что средняя длина кодовой комбинации $k_{ср}$ нелинейно зависит от k, построим небольшую таблицу (Табл.2.2) функции $k\ /\ (1-p_П^{2^{k}-1})$, которая позволяет для данного $p_п \geq 1/2$ выбрать наилучшее k.

Таблица 2.2

k	$p_п$							
	0.5	0.6	0.7	0.8	0.9	0.95	0.98	0.99
1	2.00	2.50	3.33	5.00	10.0	20.0	50.0	100
2	2.29	2.55	3.04	4.10	7.38	14.0	34.0	67.3
3			3.27	3.80	5.75	9.94	22.7	44.2
4				4.15	5.04	7.45	15.3	28.6
5					5.20	6.28	10.7	18.7
6						6.25	8.33	12.8
7						7.01	7.58	9.71
8							8.05	8.67
9								9.05
10								

Средняя длина серий одинаковых символов (нулей) равна сумме произведений длин на вероятность появления серии соответствующей длины:

$$L_{ср} = \sum_{k=1}^{\infty} k p_{\text{П}}^{k} q_{\text{П}} = q_{\text{П}} / (1 - p_{\text{П}})^2 = 1 / (1 - p_{\text{П}}) \,. \quad (2.24)$$

Коэффициент сжатия сообщения в этом случае определяется по формуле:

$K_{\text{сж}}$ [redacted] (2.35)

Численные значения коэффициента сжатия при различных вероятностях предсказания и оптимальных значениях длин кодовых комбинаций k, приведены в табл.2.3. Как видно из приведенного примера, эффективность сжатия полностью определяется качеством используемого предсказателя. Чем выше вероятность предсказания, тем длиннее образуемая последовательность нулей на входе компрессора и тем больше степень сжатия сообщения. Методы компрессии с использованием предсказания широко применяются при сжатии неподвижных и подвижных изображений.

Таблица 2.3

Вероятность предсказания $p_{\text{п}}$	0.5	0.6	0.7	0.8	0.8	0.95	0.98	0.99
Оптимальное значение k	1	1	2	3	4	6	7	8
Коэффициент сжатия $K_{\text{сж}}$	1	1	0.91	0.76	0.50	0.31	0.16	0.087

Контрольные вопросы к главе 2

1. Какими параметрами характеризуется дискретный стационарный эргодический источник информации ?

2. Как изменяется энтропия источника при наличии статистической зависимости между символами на его входе?

3. Охарактеризуйте марковский дискретный источник и способы его описания ?

4. Приведите примеры кодовых деревьев равномерного и неравномерного кодов и перечислите их параметры.

5. Каковы свойства вероятностных кодовых деревьев ?

6. Как осуществляется разделение кодовых комбинаций неравномерного кода ?

7. От чего зависит средняя длина кодового слова при кодировании источника неравномерным кодом ?

8. При каких условиях может быть построен неравномерный префиксный код?

9. В чем состоит сущность кодирования с предсказанием?

Глава 3
СТАТИЧЕСКИЕ МЕТОДЫ СЖАТИЯ ДАННЫХ

3.1. Код Шеннона-Фано

Для получения оптимального кода, имеющего минимальную длину кодовой комбинации, необходимо добиваться наименьшей избыточности каждого из кодовых слова, которые в свою очередь должны строиться из равновероятных и взаимонезависимых символов. При этом каждый кодовый элемент должен принимать значения 0 или 1 по возможности с равными вероятностями, а выбор очередного элемента быть независимым от предыдущего. Алгоритм построения такого кода впервые был предложен К.Шенноном в 1948 году и несколько позже модифицирован Р.Фано, в связи с чем он получил название кода Шеннона-Фано [16].

В соответствии с алгоритмом в начале процедуры кодирования все символы алфавита источника заносятся в таблицу в порядке убывания вероятностей. На первом этапе кодирования символы разбиваются на две группы так, чтобы суммы вероятностей символов в каждой из них были по возможности одинаковы. Всем символам верхней группы присваивается элемент кодовой комбинации 0, а всем нижним - 1. На втором этапе кодирования каждая из групп вновь разбивается на две равновероятные подгруппы. Второму элементу кодовых комбинаций для верхней подгруппы присваивается 0, а нижней - 1. Процесс кодирования продолжается до тех пор, пока в каждой подгруппе не останется по одному символу. Аналогично может быть построен альтернативный вариант оптимального префиксного кода Шеннона-Фано, в котором в процессе кодирования верхним подгруппам символов присваивается кодовый элемент 1, а нижним- 0. Этот код отличается от предыдущего тем, что его кодовые комбинации для соответствующих символов будут инверсными.

Рассмотрим алгоритм Шеннона-Фано на примере кодирования источника, алфавит которого состоит из 8 символов a_i (i = 1, 2, ... , 8), а вероятности появления символов в сообщении равны отрицательным степеням двойки, то есть $P(a_i) = (1/2)^i$. Чтобы ансамбль сообщений источника представлял полную группу событий, в примере принято $P(a_8)=P(a_7)=(1/2)^7$. Процедура разбиения символов на группы и подгруппы и образование кодовых слов показано в табл.3.1.

Таблица 3.1

Символы a_i	Вероятности $P(a_i)$	Этапы кодирования							Кодовые комбинации
		I	II	III	IV	V	VII	VII	
a_1	1/2	0							0
a_2	1/4	1	0						10
a_3	1/8	1	1	0					110
a_4	1/16	1	1	1	0				1110
a_5	1/32	1	1	1	1	0			11110
a_6	1/64	1	1	1	1	1	0		111110
a_7	1/128	1	1	1	1	1	1	0	1111110
a_8	1/128	1	1	1	1	1	1	1	1111111

Среднее число битов на символ при кодировании заданного источника кодом Шеннона-Фано равно:

$$L_{cp} = \sum_{i=1}^{8} P(a_i) l(a_i) = 1/2 + (1/4) \times 2 + (1/8) \times 3 + (1/16) \times 4 +$$

$$+ (1/32) \times 5 + (1/64) \times 6 + (1/128) \times 7 + (1/128) \times 8 = 1\frac{63}{64},$$

а энтропия источника:

$$H(A) = -\sum_{i=1}^{8} P(a_i) \log P(a_i) = (1/2)\log(1/2) + (1/4)\log(1/4) +$$

$$+ (1/8)\log(1/8) + (1/16)\log(1/16) + (1/32)\log(1/32) +$$

$$+ (1/64)\log(1/64) + (1/128)\log(1/128) + (1/128)\log(1/128) =$$

$$= 256 / 128 = 1\frac{63}{64}.$$

Таким образом код Шеннона-Фано для заданного распределения вероятностей символов является оптимальным, так как среднее число битов на символ в точности равно энтропии источника.

При обычном кодировании, не учитывающем статистических свойств источника, для представления каждой буквы требуется 3 бита. Следовательно, коэффициент сжатия $K_{сж}$ сообщения за счет

неравномерного кодирования Шеннона-Фано равен $K_{сж}$ = 2/3 = 0,666 = 67%.

Рассмотрим пример кодирования по алгоритму Шеннона-Фано для ансамбля символов с произвольным распределением вероятностей (табл.3.2).

Таблица 3.2

Символы a_i	Вероятности $P(a_i)$	Этапы кодирования				Кодовые комбинации
		I	II	III	IV	
a_1	0,22	0	0			00
a_2	0,20	0	1	0		010
a_3	0,16	0	1	1		011
a_4	0,16	1	0	0		100
a_5	0,10	1	0	1		101
a_6	0,10	1	1	0		110
a_7	0,04	1	1	1	0	1110
a_8	0,02	1	1	1	1	1111

Энтропия для ансамбля символов равна $H(A)$ = 2,76, а среднее число битов на символ $L_{ср}$=2,84. Это связано с тем, что некоторая избыточность в сжатом сообщении осталась. По сравнению с равномерным кодированием коэффициент сжатия для кода Шеннона-Фано составляет 0,95. Из теоремы Шеннона следует (2.30), что эту избыточность можно устранить, если перейти к кодированию достаточно большими блоками.

Пример 3.1. Закодировать по Шеннону-Фано последовательность, генерируемую двоичным марковским источником 1-го порядка, граф которого изображен на рис.3.1. Требуется составить таблицы кодов при объединении символов в блоки по 2 и 3 элемента; рассчитать энтропию источника, энтропию источника при группировании символов по два (энтропия 1-го порядка) и по три (энтропия 2-го порядка), определить избыточность полученных кодов, а также коэффициенты сжатия за счет применения неравномерного кодирования для обоих кодов.

Из рис.3.1 видно, что начальные вероятности состояний $P(S_1) = P(0)$ и $P(S_2) = P(1)$, при которых источник генерирует соответственно символы 0 и 1, одинаковы и равны 0,5, а вероятности переходов из одного состояние в другое равны следующим значениям: $P(0/0) = P(1/1)$ = 0,7; $P(1/0) = P(0/1)$ = 0,3. Очевидно, что при отсутствии группирования алфавит источника содержит два символа (m=2). Первый символ кодируется нулем, а второй — единицей. Энтропия источника (в битах на символ) равна

$$H_S = -0{,}7 \log 0{,}7 - 0{,}3 \log 0{,}3 = 0{,}881 \text{ бит/сим.}$$

Так как максимальная энтропия $H_{max} = log\ m = 1$ бит/сим, то избыточность кода

$$r = H_{max} - H_S / H_{max} = 1 - 0{,}881 / 1 = 0{,}119 = 11{,}9\ \% .$$

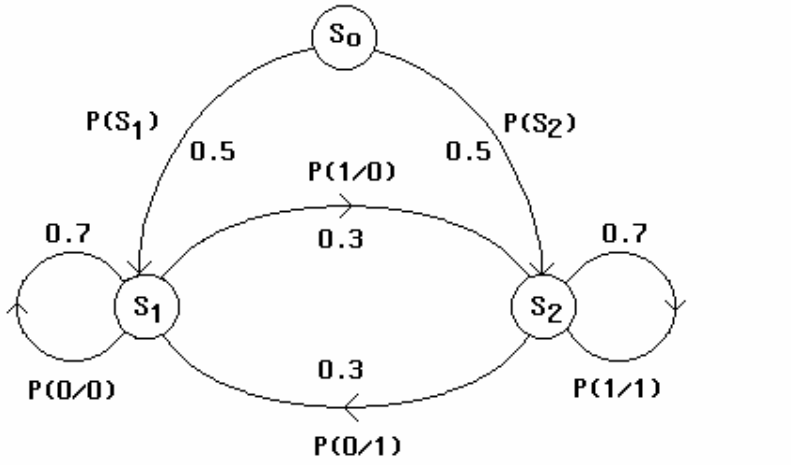


Рис.3.1. Граф марковского источника 1-го порядка

Естественно, что о кодировании методом Шеннона-Фано двух символов не может быть и речи.

Произведем укрупнение источника путем группирования символов по два (в дибиты) и рассчитаем вероятности появления дибит с учетом начальных вероятностей и вероятностей переходов:

$$P(0,0) = P(0)\ P(0/0) = 0{,}5 \times 0{,}7 = 0{,}35 ;$$
$$P(1,1) = P(1)\ P(1/0) = 0{,}5 \times 0{,}7 = 0{,}35 ;$$
$$P(0,1) = P(0)\ P(1/0) = 0{,}5 \times 0{,}3 = 0{,}15 ;$$
$$P(1,0) = P(1)\ P(0/1) = 0{,}5 \times 0{,}3 = 0{,}15 .$$

Построим таблицу кода Шеннона-Фано (табл.3.3). Энтропия 1-го порядка вычисляется по формуле (2.9) и равна

$$H_S = 0{,}5[2 \times 0{,}35 \log(1/ 0{,}35) + 2 \times 0{,}15 \log(1/0{,}15)] = 0{,}941 \text{ бит/сим.}$$

Таблица 3.3

Группы символов	Вероятности групп P_{ij}	Этапы кодирования			Кодовые комбинации
		I	II	III	
00	0.35	0			0
11	0.35	1	0		10
01	0.15	1	1	0	110
10	0.15	1	1	1	111

Обратите внимание, что множитель 0,5 учитывает то, что каждое кодовое слово содержит информацию о двух исходных символах источника. Для определения избыточности кода найдем среднюю длину кодовой комбинации на символ

$$l_{ср.сим} = l_{ср} / 2 = 0{,}5[1 \times 0{,}35 + 2 \times 0{,}35 + 3 \times 0{,}15) = 0{,}975 \text{ бит/сим.}$$

$$r = 1 - 0{,}941 / 0{,}975 = 0{,}034 = 3{,}4\ \% .$$

При вычислении коэффициента сжатия следует иметь в виду, что для передачи равномерным кодом понадобилось бы два бита, а при передаче неравномерным кодом $l_{ср} = 1{,}95$ битов. Следовательно, $K_{сж} = 1{,}95/2 = 0{,}975 = 97{,}5\ \%$.

Объединим символы источника в группы по три ($n_с = 3$) и рассчитаем вероятности появления этих групп:

$$P(0,0,0) = P(0,0)\,P(0,0) = 0{,}35 \times 0{,}7 = 0{,}245 ;$$
$$P(0,0,1) = P(0,0)\,P(1/0) = 0{,}35 \times 0{,}3 = 0{,}105 ;$$
$$P(0,1,0) = P(0,1)\,P(0/1) = 0{,}15 \times 0{,}3 = 0{,}045 ;$$
$$P(0,1,1) = P(0,1)\,P(1/1) = 0{,}15 \times 0{,}7 = 0{,}105;$$
$$P(1,0,0) = P(1,0)\,P(0/0) = 0{,}15 \times 0{,}7 = 0{,}105 ;$$
$$P(1,0,1) = P(1,0)\,P(1/0) = 0{,}15 \times 0{,}3 = 0{,}045 ;$$
$$P(1,1,0) = P(1,1)\,P(0/1) = 0{,}35 \times 0{,}3 = 0{,}105 ;$$
$$P(1,1,1) = P(1,1)\,P(1/1) = 0{,}35 \times 0{,}7 = 0{,}245 .$$

Комбинации кода Шеннона-Фано приведены в табл.3.4.

Таблица 3.4

Группы символов	Вероятности групп	Этапы кодирования					Кодовые комбинации
		I	II	III	IV	V	
000	0.245	0	0				00
111	0.245	0	1				01
001	0.105	1	0	0			100
011	0.105	1	0	1			101
100	0.105	1	1	0			110
110	0.105	1	1	1	0		1110
010	0.045	1	1	1	1	0	11110
101	0045	1	1	1	1	1	11111

Энтропия источника 2-го порядка, средняя длина кода на символ и его избыточность соответственно равны:

$$H_S = (1/3)[2 \times 0{,}245 \log(1/0{,}245) + 4 \times 0{,}105 \log(1/0{,}105) + 2 \times 0{,}045 \log(1/0{,}0045)] = 0{,}921 \text{ бит/сим};$$

$$l_{\text{ср.сим}} = l_{ср} / n_c = (1/3)(2 \times 2 \times 0{,}245 + 3 \times 3 \times 0{,}105 + 4 \times 0{,}105 + 2 \times 5 \times 0{,}045) = 0{,}932 \text{ бит/сим};$$

$$r = 1 - 0{,}921/0{,}932 = 0{,}012 = 1{,}2\ \%.$$

Из примера видно, что избыточность кода при группировании по три символа существенно ниже по сравнению с кодированием одиночных символов и группированием их по два.

Алгоритм Шеннона-Фано можно легко обобщить на случай произвольного алфавита из D символов путем последовательного разбиения ансамбля символов не на два, а на D равновероятных групп и подгрупп. Алгоритм обеспечивает получение кода минимальной длины при условии, если вероятности символов равны отрицательным степеням D независимо от того, равно ли D двум или любому другому целому числу. В противном случае полученные при разбиении подгруппы можно считать равновероятными лишь приближенно, в связи с чем среднее число кодовых элементов не может быть сделано равным минимально возможной величине. Пример построения кода Шеннона-Фано для $D > 2$ приведен в [21].

3.2. Код Хаффмена

Алгоритм Шеннона-Фано не всегда приводит к однозначному построению кода. Это вызвано тем, что при разбиении *m* символов источника на подгруппы можно сделать большей по вероятности как верхнюю, так и нижнюю подгруппы. При различном разбиении на подгруппы среднее число битов на символ может быть разным. Более эффективным является алгоритм, предложенный Хаффменом в 1952 г., который позволяет построить оптимальный код с наименьшим для данного распределения вероятностей средним числом битов на символ, то есть

$$l_{cp} = min\ [\ \sum_{i=1}^{m} P(a_i) l(a_i)\ ]\ . \qquad (3.1)$$

Для двоичного кода алгоритм Хаффмена сводится к следующему [21]. Символы сообщения упорядочиваются по убыванию вероятностей и располагаются в основной столбец таблицы таким образом, что $P(a_i) \geq P(a_j)$ для всех $i<j$ (табл.3.5).

Таблица 3.5

Символы a_i	Вероятности $P(a_i)$	Дополнительные столбцы						
		1	2	3	4	5	6	7
a_1	0.22	0.22	0.22	0.26	0.32	0.42	0.58	1.0
a_2	0.20	0.20	0.20	0.22	0.26	0.32	0.42	
a_3	0.16	0.16	0.16	0.20	0.22	0.26		
a_4	0.16	0.16	0.16	0.16	0.20			
a_5	0.10	0.10	0.16	0.16				
a_6	0.10	0.10	0.10					
a_7	0.04	0.06						
a_8	0.02							

Два последних символа объединяются в один вспомогательный, вероятность которого равна суммарной вероятности составляющих его символов. Все оставшиеся символы, вместе с образованным вспомогательным символом, снова располагаются по убыванию

вероятностей в дополнительном столбце. Два последних элемента столбца объединяются во второй вспомогательный символ, и образуется следующий дополнительный столбец, в котором все элементы расположены в порядке убывания вероятностей. Процедура продолжается до тех пор, пока не получится единственный вспомогательный символ, имеющий вероятность, равную единице. Код, построенный по рассмотренному алгоритму, получил название *кода Хаффмена*.

Для формирования кодовых комбинаций, соответствующих символам данного сообщения, необходимо проследить путь перехода символов по строкам и столбцам таблицы. При построении кодов Хаффмена наиболее часто используются кодовые деревья. С одной стороны это позволяет более наглядно отобразить процедуры кодирования и декодирования, а с другой — облегчить программную реализацию этих процедур.

Построение дерева (Рис.3.2,*а*) начинается с корневого узла, вероятность которого равна 1. Из корня проводятся две ветви, причем ветви с большей вероятностью присваивается значение (*бит*) 1, а с меньшей вероятностью - 0. Вновь образованные узлы могут отображать одиночный или вспомогательный символы. В последнем случае узел является промежуточным и из каждого из них, в свою очередь, снова проводятся по две ветви. Такое последовательное ветвление продолжается до тех пор, пока не будет достигнут узел, соответствующий вероятности символа алфавита (*узел листа*). Двигаясь по кодовому дереву от корня сверху вниз, можно записать для каждого символа соответствующую ему кодовую комбинацию (рис.3.2,б). Среднее число битов на символ при таком построении кода составляет

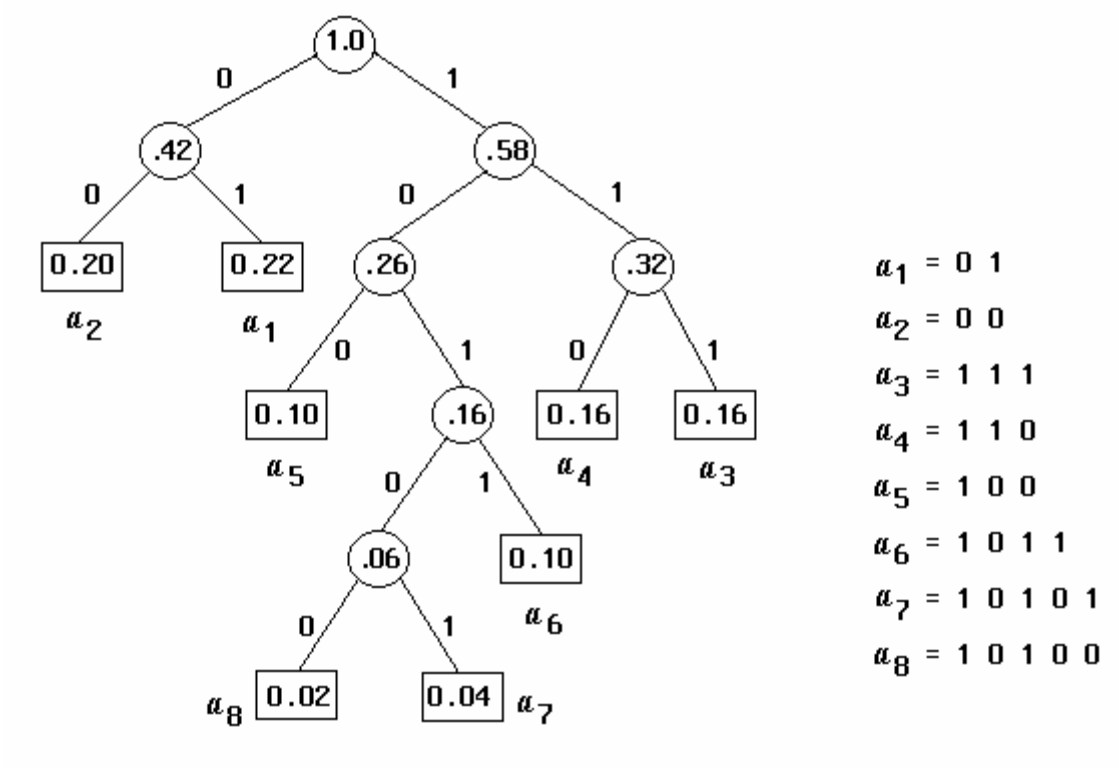
$$l_{ср} = \sum P(a_i)\, l(a_i) = 0{,}22 \times 2 + 0{,}2 \times 2 + 0{,}16 \times 3 + 0{,}16 \times 3 +$$
$$+ \ 0{,}1 \times 3 + 0{,}1 \ \times 4 + 0{,}04 \times 5 + 0{,}02 \times 5 = 2{,}8 \ \text{бит}.$$

Энтропия источника сообщения равна:

$$H(A) = -\sum P(a_i)\, \log P(a_i) = -\,(0{,}22 \log 0{,}22 + 0{,}2 \log 0{,}2 \ +$$
$$+0{,}16 \log 0{,}16 + 0{,}16 \log 0{,}16 + 0{,}1 \log 0{,}1 + 0{,}1 \log 0{,}1 +$$
$$+ \ 0{,}04 \log 0{,}04 + 0{,}02 \log 0{,}02 \ = \ 2{,}754 \ \text{бит}.$$

Здесь и во всех остальных примерах, если не будет специально оговорено, используется *логарифм по основанию 2*. Как видно из рассмотренного примера средняя длина кодовой комбинации и энтропия

источника практически совпадают, т. е. полученный код является оптимальным.



а) б)

Рис.3.2. Кодовое дерево (а) и таблица кода Хаффмена (б)

Алгоритм Хаффмена является двухпроходным, так как при его реализации требуется дважды просматривать кодируемое сообщение. При первом проходе вычисляются вероятности (*частоты*) появления символов в сжимаемом сообщении и строится хаффменовское дерево.

При втором проходе осуществляется кодирование символов, поступивших от источника. В этом случае определяется значение ветвей дерева при движении от *листа*, соответствующему кодируемому символу, к *корню*. Очевидно, что для ускорения процедуры декодирования биты кодовой комбинации на выход кодера должны выдаваться, начиная со старшего разряда, т.е. с ветви дерева, исходящей от корня. На практике вместо вероятностей символов используют абсолютные значения количества (*вес*) символов в передаваемом сообщении, так как количество символов пропорционально вероятности их появления. Для однозначного декодирования таблица вероятностей символов сообщается декодеру.

Для источника, выбирающего последовательности символов из некоторого алфавита объемом m, алгоритм построения бинарного кодового дерева Хаффмена может быть записан следующим образом:

Алгоритм 3.1

Сохранить *m* листьев в списке *List*;
ПОКА *L* содержит по крайней мере два узла
ВЫПОЛНЯТЬ
НАЧАЛО
Выбрать из *List* два узла *x* и *y* с наименьшими весами;
Создать новый узел *p* и назначить его родительским узлом *x* и *y*;
Вес узла *p*:= вес узла *x* + вес узла *y*;
Включить *p* в список *List* в соответствующее место с учетом убывания весов;
КОНЕЦ.

Узел, остающийся в *List* в конце процедуры, является корнем дерева Хаффмена.

Пример 3.2.

Закодировать файл символов, состоящий из букв от *A* до *H*. Построить кодовое дерево и таблицу кодирования символов, используя методику Хаффмена, если в результате первого прохода файла установлено, что он содержит 9 символов *A*, 9*B*, 5*C*, 5*D*, 2*E*, 2*F*, 2*G* и 2*H*.

Занесем символы в основной столбец таблицы в порядке уменьшения их веса (табл. 3.6).

Таблица 3.6

Символы	Вес символа	Этапы кодирования						
		I	II	III	IV	V	VI	VII
A	9	9A	9A	9A	10CD	17BGHEF	19ACD	36
B	9	9B	9B	9B	9A	10CD	17BGHEF	
C	5	5C	5C	8GHEF	9B	9A		
D	5	5D	5D	5C	8GHEF			
E	2	4GH	4GH	5D				
F	2	2E	4EF					
G	2	2F						
H	2							

На основании этой таблицы составим кодовое дерево (рис.3.3,а), из которого легко записать кодовые комбинации, отображающие соответствующие символы (рис.3.3,б).

Для кодирования заданного файла требуется

$$N_с = 9\times2 + 9\times2 + 5\times3 + 5\times3 + 2\times4 + 2\times4 + 2\times4 + 2\times4 = 98 \text{ бит.}$$

В случае простого двоичного кода для представления восьми символов нужно 3 бита, следовательно число битов для несжатой информации $N_б = 36 \times 3 = 108$. Коэффициент сжатия файла при этом равен

$$K_{сж} = N_с / N_б = 98 / 108 = 0{,}91 = 91\%.$$

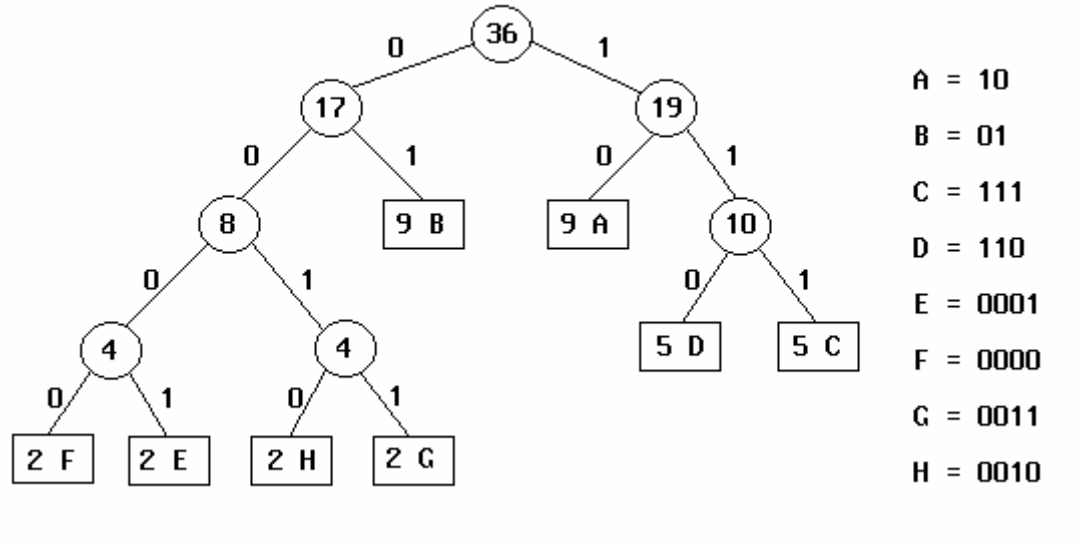
Пример 3.3.

Закодировать кодом Хаффмена файл, в котором веса символов распределены следующим образом: 18*A*, 10*B*, 2*C*, 2*D*, 1*E*, 1*F*, 1*G* и 1*H*.

Кодовое дерево Хаффмена, построенное аналогично предыдущему примеру, показано на рис.3.4,а, а таблица кодирования на рис.3.4,б. Число требуемых битов для передачи файла кодом Хаффмена равно

$$N_с = 18 \times 1 + 10 \times 1 + 2 \times 4 + 2 \times 4 + 4 \times 5 = 74 ,$$

а коэффициент сжатия $K_{сж} = 74 / 108 = 0{,}685 = 68{,}5\%$.

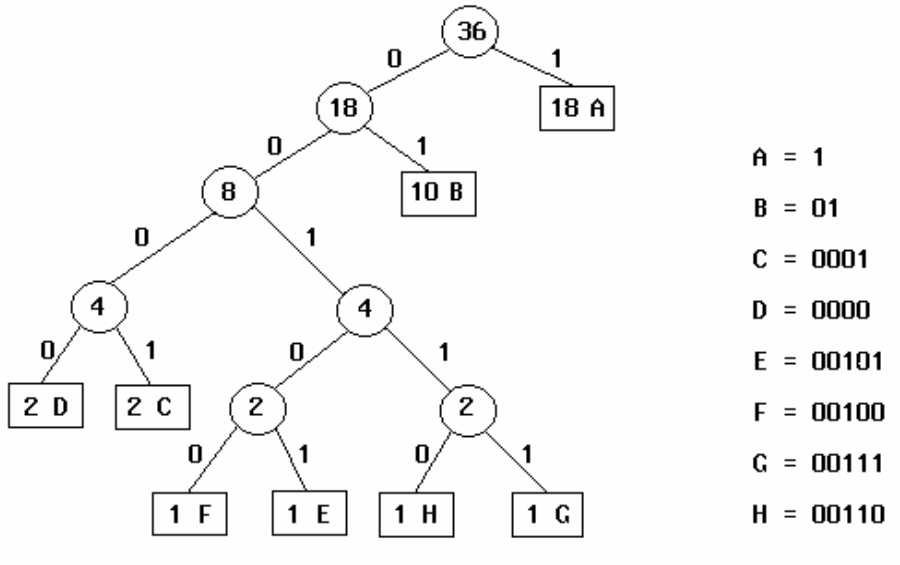


а) б)

Рис.3.3. Кодовое дерево Хаффмена для примера 3.2

Из примеров видно, что чем *больше различие в частоте* появления символов, *тем более эффективно кодирование* неравномерными кодами.

Рассмотрев методику построения оптимальных кодов, нетрудно убедиться в том, что эффективность сжатия достигается благодаря присвоению коротких кодовых комбинаций более вероятным символам, а более длинных - менее вероятным. Разделимость неравномерных кодовых комбинаций возможна за счет префиксного свойства кодов Хаффмена и Шеннона-Фано.



а) б)

Рис.3.4. Кодовое дерево для примера 3.3

Одной из проблем, возникающей при передаче сжатой информации, является сопряжение синхронного канала передачи данных с неравномерным поступлением битов от кодера, который выдает за равные промежутки времени комбинации переменной длины. В этом случае между кодером и каналом связи должно быть предусмотрено буферное устройство, накапливающее кодовые комбинации по мере поступления их с кодера и выдающее единичные элементы в канал с постоянной скоростью. Очевидно, что включение в тракт передачи буфера вносит задержку при передаче сообщений, которая возрастает с увеличением емкости буфера. Уменьшение объема буфера может привести к его переполнению и потери части сообщения. Поэтому задача выбора оптимальной емкости буфера в каждом конкретном случае передачи данных является весьма актуальной.

Вторая проблема появляется при передаче данных по каналу с помехами. Даже одиночная ошибка может преобразовать кодовую

комбинацию в другую разрешенную, равную ей по длительности, что приведет к сбою процесса декодирования. Поэтому для решения этой проблемы оптимальное кодирование применяется в совокупности с помехоустойчивым кодированием. В этом случае перед подачей принимаемой последовательности на вход декомпрессора в ней исправляются обнаруженные ошибки.

3.3. Арифметическое кодирование

При арифметическом сжатии сообщений алфавиту источника ставится в соответствие числовой, открытый справа, интервал [0, 1), а каждый символ алфавита сопоставляется с различными участками этой числовой оси. Ширина интервала (*диапазон*) каждого участка зависит от вероятности (*частоты*) появления символа в сообщении [33,34,39,44].

Рассмотрим в качестве примера алфавит, состоящий из шести символов A, B, C, D, E, ! с вероятностями появления соответственно 0,1; 0,1; 0,3; 0,2; 0,2 и 0,1. Тогда интервал [0, 1) может быть разделен на участки следующим образом:

A: [0 - 0,1); B: [0,1 - 0,2); C: [0,2 - 0,5);
D: [0,5 - 0,7); E: [0,7 - 0,9); !: [0,9 - 1,0).

Деление числовой оси иллюстрируется рис.3.5. Как видно из примера, разделение единичного интервала наиболее просто и наглядно производится суммированием вероятности каждого символа с границей предшествующего.

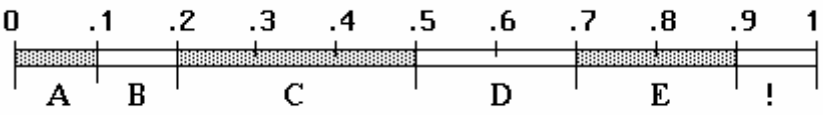


Рис.3.5. Распределение отрезков числовой оси между символами при арифметическом кодировании

В процессе сжатия строка текстового сообщения представляется двумя дробными числами, отображающими некоторый интервал числовой оси [0 - 1). До начала кодирования допустимой областью для строки сообщения является весь интервал [0 — 1). При поступлении от источника первого символа ему выделяется интервал, соответствующий расположению этого символа на оси [0—1) согласно вероятности

появления его в сообщении. Поступление следующего символа сужает выделенный интервал. При этом границы нового интервала определяются числовым диапазоном, соответствующим очередному символу, т.е. ширина диапазона пропорциональна вероятности символа. С приходом следующего символа производится дальнейшее сужение интервала. Таким образом, с увеличением длины кодируемой строки происходит постоянное сужение интервала. Теоретически его можно сужать сколь угодно долго, а на практике этот интервал ограничивается техническими возможностями кодирующих устройств (разрядностью слова).

Число битов, необходимое для представления интервала шириной s, равна $-\log s$. Основание логарифма здесь и в последующих выражениях равно 2. Ширина s последнего интервала строки сообщения, состоящего из N_C символов, определяется произведением вероятностей символов P_i сообщения

$$S = \prod_{i=1}^{N_c} P_i \ .$$

В связи с этим можно записать, что

$$-\log s = -\sum_{i=1}^{N_c} \log P_i = -\sum_{i=1}^{m} P(a_i) \log P(a_i) \ ,$$

где m -число различных символов сообщения a_i (i=1, ... , m). Таким образом, число битов, затрачиваемое на кодирование сообщения, при арифметическом кодировании в точности равняется энтропии источника.

Рассмотрим процедуру арифметического сжатия на примере кодирования строки сообщения CADA!. Здесь символ ! является признаком окончания строки. Вероятности появления символов и соответствующие им числовые диапазоны возьмем из предыдущего примера. Начальный числовой диапазон, выделяемый для кодера, равен единице [0 — 1). После поступления от источника первого символа С числовой интервал сужается до величины [0,2 — 0,3). С приходом следующей буквы А границы диапазона становятся равными [0,2 —0,23). Процедура кодирования наглядно иллюстрируется рис.3.6. Вертикальные линии отображают начальную числовую ось [0 — 1) и её последующие участки в увеличенном размере (промасштабированные на единичную ось) на различных этапах кодирования. Обратите внимание, что какая бы не была длинная строка, конечный диапазон всегда будет расположен в интервале первого кодируемого символа строки.

Расчет интервалов осуществляется по рекуррентной формуле, которая получена на основе рис.3.6:

$$(X_{max} - X_{min}) X'_{min} + X_{min}$$
$$(X_{max} - X_{min}) X'_{max} + X_{min}, \qquad (3.2)$$

где X_{min} и X_{max} - нижняя и верхняя границы для предыдущей итерации; X'_{min}, X'_{max} - границы интервала нового символа.

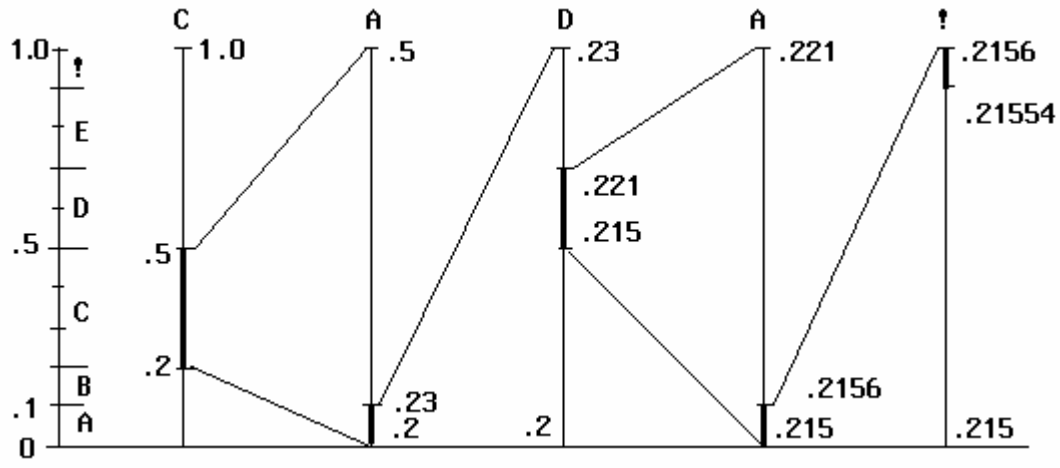


Рис.3.6. Иллюстрация процедуры арифметического кодирования

Алгоритм арифметического кодирования в общем виде записывается следующим образом:

Алгоритм 3.2

```
НАЧАЛО
Нижняя_Граница = 0.0 ;
Верхняя_Граница = 1.0 ;
ПОКА существует входной символ ВЫПОЛНЯТЬ
      Ввести очередной символ С;
      Интервал = Верхняя_Граница - Нижняя _Граница;
      Нижняя_Граница = Нижняя_Граница +
                        + Интервал × Нижняя_Граница(С);
      Верхняя_Граница = Нижняя_Граница +
                        + Интервал × Верхняя_Граница(С);
КОНЕЦ ПОКА;
ВЫДАТЬ  Нижняя_Граница ;
КОНЕЦ.
```

Границы интервалов для всех этапов кодирования приведены в табл.3.*7.* Таким образом, числовой интервал с границами [0,21554—0,2156) отображает строку сообщения CADA! Для кодирования этой строки может быть использовано любое число из полученного диапазона. Пусть таким числом является нижняя граница интервала, т. е. $X = 0{,}21554$.

Таблица 3.7

Символы	Нижняя граница интервала	Верхняя граница интервала
Исходное состояние	0,0	1,0
C	0,2	0,5
A	0,2	0,23
D	0,215	0,221
A	0,215	0,2156
!	0,21554	0,2156

В процессе декодирования осуществляется обратная процедура отождествления принятого числа (*интервала*) с закодированной строкой символов. Очевидно, что на приемной стороне должны быть известны вероятности декодируемых символов и соответствующие им интервалы числовой оси [0,1).

Пусть на вход декодера поступило число X = 0,21554. Так как оно находится в диапазоне 0,2 – 0,5, т. е. 0,2 ≤ X < 0,5, то на выход декомпрессора выдается символ C. Затем декодер определяет, в какую относительную долю отрезка [0,2 – 0,5) попадает число 0,21554, для чего он производит вычисления по формуле

$$X' = (X - X_{min}) / (X_{max} - X_{min}), \qquad (3.3)$$

здесь X'- относительное значение числа X на очередном участке числовой оси. В нашем случае X'= (0,21554 – 0,2)/ 0,3= 0,0518. Так как 0 < X' < 0,1, то на выход декомпрессора поступает символ A. Нетрудно заметить, что процедура определения выходного символа рекуррентна, при которой после нахождения первого диапазона для заданного числа X на единичной числовой оси определяется относительная доля (диапазон) этой части отрезка, куда попадает декодируемое сообщение.

Вычисление на следующем шаге производится по формуле (3.3) в предположении, что $X = X'$, где X'- относительное значение числа,

найденного в предыдущей итерации. Новые значения X_{min} и X_{max} определяются границами интервала, идентифицированного на предыдущей итерации. На этом шаге X = 0,0518, X_{min}=0, X_{max} =0,1, а X'= 0,0518 / 0,1 = 0,518.

Новое X' соответствует символу D, так как 0,5 < X'< 0,7. Далее, полагая X = 0,518, X_{min}= 0,5 и X_{max} = 0,7 , найдем аналогично X'=(0,518 – 0,5) / 0,2 = 0,09, что соответствует символу A. Так как на следующей итерации X'= 0,09 / 0,1 = 0,9 , то декодер выводит символ !. В связи с тем, что символ ! является признаком окончания строки сообщения, то декодирование на этом заканчивается.

Алгоритм арифметического декодирования строится на основе формулы (3.3) и в общем виде может быть представлен следующим образом:

Алгоритм 3.3

```
НАЧАЛО
ВВЕСТИ число X, соответствующее переданному
        сообщению;
НАЙТИ символ C, в интервал которого попадает
        число X;
ЕСЛИ C не конец сообщения ВЫПОЛНЯТЬ
ВЫДАТЬ символ C;
Интервал = Верх_Граница_Символа -
        - Ниж_Граница_Символа;
Число = (Число - Ниж_Граница_Символа) / Интервал;
Конец  НАЙТИ
КОНЕЦ .
```

Пример 3.4. Произвести компрессию и декомпрессию сообщения “ИНФОРМАЦИЯ”, используя метод арифметического кодирования.

Определим вероятности появления символов в заданном сообщении и распределим символы на единичном отрезке (табл. 3.8). На основании алгоритма кодирования рассчитаем нижние и верхние границы кодового числа, отображающего сжимаемое сообщение, в зависимости от количества закодированных символов. Данные расчета сведем в табл.3.9. В качестве кодового числа, представляющего сжатое сообщение, примем нижнюю границу этого числа: 0,1951261656. Используя алгоритм декодирования, произведем процедуру декомпрессии, этапы которой приведены в табл.3.10 .

Таблица 3.8

Символы	Вероятность появления	Расположение на отрезке
А	0,1	$0,0 \le X < 0,1$
И	0,2	$0,1 \le X < 0,3$
М	0,1	$0,3 \le X < 0,4$
Н	0,1	$0,4 \le X < 0,5$
О	0,1	$0,5 \le X < 0,6$
Р	0,1	$0,6 \le X < 0,7$
Ф	0,1	$0,7 \le X < 0,8$
Ц	0,1	$0,8 \le X < 0,9$
Я	0,1	$0,9 \le X < 1,0$

Таблица 3.9

Символы	Нижняя граница интервала	Верхняя граница интервала
Исходное состояние	0,0	1,0
И	0,1	0,3
Н	0,18	0,2
Ф	0,194	0,196
О	0,1950	0,1952
Р	0,19512	0,19514
М	0,195126	0,195128
А	0,195126	0,1951262
Ц	0,19512616	0,19512618
И	0,195126162	0,195126166
Я	0,1951261656	0,195126166

Таблица 3.10

Кодовое число	Выходной символ	Нижняя граница	Верхняя граница	Интервал
0,1951261656	И	0,1	0,3	0,2
0,4756300828	Н	0,4	0,5	0,1
0,75630828	Ф	0,7	0,8	0,1
0,5630828	О	0,5	0,6	0,1
0,630828	Р	0,6	0,7	0,1
0,30828	М	0,3	0,4	0,1
0,0828	А	0,0	0,1	0,1
0,828	Ц	0,8	0,9	0,1
0,28	И	0,1	0,3	0,2
0,9	Я	0,9	1,0	0,1
0,0 - конец декодирования				

3.4. Особенности программной реализации арифметического кодирования

Алгоритмы арифметического кодирования и декодирования с математической точки зрения являются несложными. Однако при практической их реализации возникает ряд проблем, одной из которых является чрезвычайно быстрое сужение кодируемого интервала и рост разрядности числа, отображающего этот интервал. Очевидно, разрядность результирующего числа зависит от длины кодируемого сообщения, которое на практике достигает десятков мегабитов. Имеется несколько способов решения данной проблемы. Один из них заключается в том, что сообщение перед обработкой разбивается на блоки, которые кодируются числом с плавающей запятой. Размер блока (обычно он не превышает 80 байтов) зависит от длины мантиссы и выбранного типа данных языка программирования. В этом случае исключается также необходимость контроля состояние переполнения разрядов типа данных. Но при этом происходит снижение степени сжатия, так как сообщение на выходе кодера представляет собой не одно, а несколько вещественных чисел.

Более эффективным является так называемое “накапливаемое кодирование”, при котором в процессе обработки очередного символа сообщения производится контроль переполнения, то есть проверяется, не переполнит ли двоичное представление числа выбранный для него тип данных. При обнаружении возможности переполнения производится передача старшего бита этого числа в промежуточный буфер. Так как величина приращения кода интервала на каждой итерации (в двоичном представлении) намного меньше значения его максимального разряда (см. табл.3.9), то дальнейшее изменение данного бита уже не будет происходить и его можно вносить в окончательную кодовую последовательность, накапливаемую в промежуточном буфере. Этот способ обеспечивает эффективное сжатие и достаточно высокое быстродействие.

При программной реализации процедуры декомпрессии необходимо решить проблемы контроля переполнения и быстрого поиска символьных интервалов. Алгоритмы арифметического кодирования и декодирования наиболее просто реализуются на основе стандартной 16- или 32-битовой целочисленной арифметики [37]. В процессе работы алгоритм отслеживает верхнюю и нижнюю границы кодового интервала. Сразу после старта нижняя граница устанавливается в 0,0, а верхняя - в 1,0. Первое упрощение для работы с целочисленной арифметикой состоит в замене 1 на бесконечную дробь 0,9999... в десятичной форме или 0,1111...— в двоичной. Такое представление облегчает процесс кодирования и декодирования. Для того, чтобы эти числа можно было разместить в целочисленных регистрах, их необходимо сначала выровнять влево, т. е., чтобы десятичная точка находилась на левой стороне регистров. Если компрессор реализован на основе 16-разрядного вычислителя, то начальные значения регистров в 16-ричной системе счисления будут FFFF для верхней и 0000 для нижней границ числовой оси. Поскольку предполагается, что значения в регистрах соответствуют бесконечным дробям, то впоследствии можно будет сдвигать их влево, вводя дополнительные биты по мере необходимости.

Рассмотрим работу алгоритма на примере кодирования сообщения “ИНФОРМАЦИЯ” (Табл.3.11). Для большей наглядности процедуры кодирования с выдвижением старшего разряда будем полагать, что кодер построен на основе гипотетического вычислителя, располагающего десятичными пятисимвольными регистрами. После процесса инициализации в регистрах будут храниться следующие значения:

регистр верхней границы 99999 (9999... до бесконечности);
регистр нижней границы 00000 (0000... до бесконечности).

Таблица 3.11

Символ и его интервал	Выводимый кодовый символ	Регистр верхней границы	Регистр нижней границы	Интервал	Код сообщения
Нач. состояние		99999	00000	100 000	
И (0,1 - 0,3)		29999	10000	20 000	
Н (0,4 - 0,5)		19999	18000		
	1	99999	80000	20 000	0,1
Ф (0,7 - 0,8)		95999	94000		
	9	59999	40000	20 000	0,19
О (0,5 - 0,6)		51999	50000		
	5	19999	00000	20 000	0,195
Р (0,6 - 0,7)		13999	12000		
	1	39999	20000	20 000	0,1951
М (0,3 - 0,4)		27999	26000		
	2	79999	60000	20 000	0,19512
А (0,0 - 0,1)		61999	60000		
	6	19999	00000	20 000	0,195126
Ц (0,8 - 0,9)		17999	16000		
	1	79999	60000	20 000	0,1951261
И (0,1 - 0,3)		65999	62000		
	6	59999	20000	40 000	0,19512616
Я (0,9 - 1,0)		59999	56000		
	5	99999	60000	40 000	0,195126165
	6	99999	00000		0,1951261656
	0	99999	00000		0,19512616560

На следующем шаге вычислим начальный интервал числовой оси. Следует помнить, что фактическая разница между верхней и нижней границами равняется 100 000, а не 99 999 так как в верхней границе неявно присутствует бесконечное число девяток. Затем вычислим новое значение верхней границы интервала. Верхняя граница для символа И равна 0,3, что должно привести к установке соответствующего регистра в состояние 30000. Однако перед сохранением нового числа в регистре верхней границы и принимая во внимание, что моделируется бесконечная дробь, это значение уменьшается на 1, то есть состояние регистра становится равным 29999. Вычисление нового значения нижней границы дает 10000 в соответствующем регистре. На этом этапе состояние регистров:

верхней границы 29999 (999...);
нижней границы 10000 (000...).

Так как наиболее значимые разряды в обоих регистрах не совпадают, то вводится новый символ кодируемого сообщения H и пересчитывается верхняя и нижняя границы. Теперь старшие разряды верхней и нижней границ совпадают и в силу природы алгоритма не изменятся до завершения процедуры кодирования. Поэтому старший разряд (*цифра 1*) выводится в буферный регистр в качестве первой цифры кода, отображающего кодируемое сообщение. Это осуществляется сдвигом содержимого обоих регистров влево на одну позицию и "втягивания" девятки на место младшей цифры верхней границы, и нуля - на место младшей цифры нижней границы. Далее по мере работы алгоритма верхняя и нижняя границы становятся все ближе и ближе друг к другу и при совпадении старших цифр в регистрах они выводятся в код сообщения.

После того, как обработаны все символы сообщения, необходимо выдвинуть два дополнительных разряда из регистра верхней или нижней границ, так как часть информации о кодируемом сообщении все еще находится в регистрах. В нашем примере выдвигаются две цифры регистра нижней границы. Теперь декодер будет в состоянии обработать входную последовательность.

При декомпрессии декодер должен использовать три регистра. Первые два такие же как и у кодера, а третий регистр кода - для хранения очередных битов декодируемого сообщения. В регистры границ из всех возможных значений интервалов заносятся границы того символа, при котором число в регистре кода находится между значениями верхней и нижней границ. Интервал между границами расположен не между значениями 0,0 и 1,0, а между двумя положительными 16-битовыми целыми числами. Текущая вероятность для обрабатываемого символа определяется путем деления *(Число - Нижняя_Граница)* на *(Верхняя_Граница - Нижняя_Граница +1)*.

Хотя большинство вычислителей работает с 32-х битовыми числами, весьма целесообразно, после подсчета частот символов в сообщении, осуществлять нормирование частот таким образом, чтобы вес любого символа не занимал более одного байта. Это связано с тем, что за счет ограничения весов можно производить все арифметические операции, используя целые 16-разрядные числа. При этом программа выполняется несколько быстрее и занимается меньший объем памяти для хранения значений частот. Исключается также возможность переполнения 16-битовых регистров. Следует обратить внимание на то, чтобы счетчик с ненулевым значением в процессе масштабирования не был обращен в нуль. В таком случае его полагают равным 1.

Контрольные вопросы к главе 3

1. За счет чего осуществляется сжатие информации при использовании методов кодирования Шеннона-Фано и Хаффмена ?
2. Как оценить степень приближения по эффективности разрабатываемого неравномерного кода к оптимальному ?
3. С какой целью при неравномерном кодировании осуществляется группирование символов ?
4. Закодируйте кодом Хаффмена сообщение, состоящее из 5-ти символов A, 3B, 4C, 20D и 27E. Постройте кодовое дерево.
5. В чем состоит состоит сущность арифметического кодирования и каково его преимущество?
6. Закодируйте арифметическим способом сообщение DECADA!. Значения вероятностей символов возьмите из примера раздела 3.3.
7. Каким образом осуществляется сжатие длинных сообщений при ограниченной длине машинного слова кодирующего процессора?

Глава 4

ДИНАМИЧЕСКОЕ КОДИРОВАНИЕ НЕРАВНОМЕРНЫМИ КОДАМИ

4.1. Динамическое кодирование Хаффмена

Классический метод кодирования Хаффмена предполагает до начала преобразования знание вероятностей появления символов на выходе источника информации, причем символы упорядочиваются по убыванию вероятностей их возникновения. Для упорядоченного списка составляется кодовая таблица, в которой длина выходной комбинации определяется вероятностью исходного символа. Естественно, что на передающей и приемной сторонах должны быть известны таблицы (кодовые деревья) для каждого сообщенния, подвергающегося сжатию. Этот метод обладает двумя существенными недостатками. Первый заключается в том, что для его реализации требуется два прохода кодируемого массива. При первом просмотре вычисляются вероятности появления каждого знака в сообщении и составляется таблица кода Хаффмена. На следующем этапе осуществляется кодирование на основании статической структуры дерева Хаффмена и передача символов в сжатом виде. Таким образом, выигрыш, полученный за счет сжатия данных, может заметно снижаться, особенно при передаче относительно коротких сообщений, в связи с необходимостью передавать декодеру дополнительную информацию о кодовом дереве. Второй недостаток - наличие задержки от момента поступления данных от источника до выдачи соответствующих кодовых комбинаций, что ограничивает использование неравномерного кодирования в синхронных сетях передачи информации, а также в системах, функционирующих в реальном времени.

В начале 70-х годов были разработаны однопроходные методы сжатия информации, основанные на классической процедуре кодирования Хаффмена [26,28,32,37,40,41]. Все эти методы незначительно отличаются друг от друга и их суть заключается в том, что передатчик строит дерево Хаффмена в темпе поступления данных от источника, т.е. *"на лету"*. В процессе кодирования происходит *"обучение"* кодера на основе

статистических характеристик источника сообщений, в ходе которого вычисляются оценки исходных вероятностей сообщения и производится соответствующая модификация кодового дерева Хаффмена. В связи с непрерывным изменением кодового дерева этот процесс получил название *динамического кодирования Хаффмена*. Очевидно, что для правильного восстановления сжатых данных, декодер также должен непрерывно "учиться" наряду с кодером, осуществляя синхронное изменение кодовой таблицы на приемной стороне. Для обеспечения синхронности процессов кодирования и декодирования кодер выдает символ в несжатом виде, если он впервые появился на выходе источника, и отмечает его на кодовом дереве. При повторном появлении символа на входе кодера он передается неравномерной кодовой комбинацией, определяемой позицией символа на текущем кодовом дереве. Кодер корректирует дерево Хаффмена увеличением частоты передачи символов, которые уже введены в дерево, или наращивает дерево, добавляя в него новые узлы.

Важнейшим условием, которое должно соблюдаться при модификации кодового дерева, является сохранение свойств хаффменовского дерева. Для формулирования этих свойств обратимся еще раз к алгоритму построения оптимального кода Хаффмена (п.3.2). При статическом кодировании символы размещаются в списке в невозрастающем порядке весов (вероятностей). Затем производится объединение двух узлов наименьшего веса W_i, W_j и замена их внутренним узлом с весом, равным сумме исходных весов W_i+W_j. Вновь образованный узел размещается в списке таким образом, чтобы не нарушался порядок расположения узлов по весам. Этот процесс повторяется до тех пор, пока в списке не останется один, так называемый корневой, узел.

Рассмотрим еще раз примеры построения деревьев оптимального кода Хаффмена (рис.3.3 и 3.4) и проанализируем их свойства. Как видно из рисунков, узлы дерева расположены в неубывающем порядке их весов при обходе дерева от крайнего нижнего узла к корню слева направо и снизу вверх. В связи с тем, что узлы с весами W_i и W_j объединяются попарно, то на одном уровне не может быть меньше двух узлов, причем пара узлов является дочерней, так как имеет общий родительский узел, вес которого равен сумме весов дочерних узлов. Нетрудно убедиться, что если при построении дерева предположить, что дочерний узел с большим весом соединен нулевой ветвью, а с меньшим - единичной, то хаффменовское дерево остается упорядоченным по возрастанию весов при движении от нижнего узла к корню по уровням справа налево.

При построении кодовых деревьев и их модификации осуществляется нумерация узлов с первого до $(2m-1)$-го в порядке увеличения их весов. Первый номер присваивается узлу с минимальным весом. Здесь m - число символов алфавита источника. На рис.4.1 показано

дерево Хаффмена, построенное для предыдущего сообщения 18*A*, 10*B*, 2*C*, 2*D*, 1*E*, 1*F*, 1*G* и 1*H*, приведенного в примере 3.3 при условии, что вес символа *C* увеличился на 1 и стал равным 3.

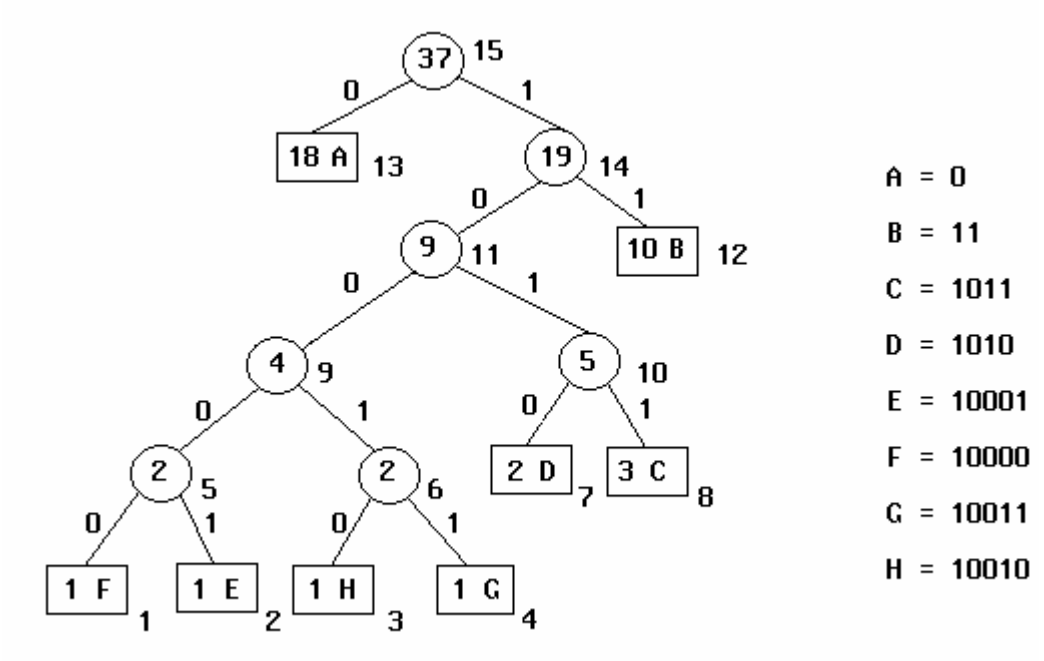


Рис.4.1. Модифицированное дерево Хаффмена для примера 3.3
после поступления очередного символа “C”

Произведем нумерацию узлов дерева, начиная с листа с минимальным весом (нижним крайним левым узлом). Как видно из рисунка, свойства хаффменовского дерева, построенного для статического кода, сохраняются, хотя оно и приобрело другую конфигурацию. При этом веса всех узлов на пути от листа с символом C до корня увеличились на 1.

При динамическом кодировании Хаффмена после получения от источника следующего символа (например C) должен увеличиться на 1 вес соответствующего ему листа (рис.3.4) и, как следствие, инкрементироваться веса всех узлов на пути от листа C к корню. Но это приведет к нарушению порядка расположения узлов по весам, присущего оптимальному кодовому дереву. Поэтому при получении очередного символа от источника необходимо осуществить модификацию кодового дерева, чтобы оно оставалось оптимальным, так называемым хаффменовским деревом.

Таким образом, на основании анализа свойств оптимальных кодовых деревьев можно сделать вывод, что кодовое дерево, имеющее *m* внешних узлов, является *хаффменовским*, если оно обладает следующими свойствами:

а) внешние узлы (листья) хафменовского кодового дерева имеют неотрицательный вес $W > 0$; каждый внутренний *(родительский)* узел имеет подчиненные (дочерние) узлы, а его вес равен сумме весов подчиненных *(дочерних)*;

б) на каждом уровне дерева (за исключением корневого) должно быть не менее одной пары узлов, имеющих общий родительский узел;

в) все узлы нумеруются в возрастающем порядке таким образом, что узлы с номерами *(2j - 1)* и *2j* являются узлами одного уровня для $1 \le j \le m-1$, а их общий родительский узел имеет более высокий уровень;

г) нумерация узлов соответствует тому порядку, в котором узлы объединяются в соответствии со статическим алгоритмом Хаффмена.

Ниже рассматриваются формальные методы построения динамического кода Хаффмена.

4.2. Алгоритм динамического кодирования методом FGK

Впервые алгоритм синтеза динамического кода Хаффмена был предложен Н.Феллером в 1973 году, а затем модифицирован Р.Галлагером и Д.Кнутом [28,32]. В связи с этим он получил название "*алгоритм FGK"*.

Введем некоторые обозначения, которые будут использоваться при анализе и синтезе динамического кодового дерева Хаффмена по алгоритму *FGK*: m — размер алфавита источника сообщений; z_j — j-й символ алфавита; $M(k) = z(1), z(2), \ldots, z(k)$ — первые k символов в сообщении; k — число символов в сообщении, обработанных до текущего момента времени; $z(k)$ — k-й символ в сообщении; K — количество различных символов, обработанных на текущий момент времени; W_j — число (вес) символов z_j, поступивших на момент обработки сообщения; l_j — расстояние от корня дерева до z_j-го листа.

Суть алгоритма синтеза динамического кодового дерева Хаффмена состоит в процедуре вычисления листьев и построении бинарного дерева с минимальным весом пути $\sum_j W_j l_j$. Процедуру кодирования можно условно разбить на два этапа, хотя при реализации алгоритма они могут легко быть объединены в один. На первом этапе дерево Хаффмена, построенное после обработки сообщения $M(k)$, преобразуется в другое, эквивалентное исходному, которое затем простым приращением весов может быть преобразовано в хаффменовское дерево для $M(k+1)$.

Первый этап начинается после получения от источника символа $z(k+1)$ с присвоения статуса *текущего* узла листу $z(k+1)$. Затем происходит обмен текущего узла (включая образованное им поддерево), с

узлом, имеющим *наибольший порядковый номер* с таким же весом. После этого в качестве нового текущего узла инициализируется родительский узел последнего текущего узла. Обмен узлами в случае необходимости многократно повторяется, пока не будет достигнут корень дерева. Несложно убедиться, что максимальное количество перестановок, которые могут понадобиться при модификации кодового дерева, равно высоте дерева $l_{\max}$.

На втором этапе инкрементируется лист дерева, соответствующий обрабатываемому символу и последующие промежуточные узлы, расположенные на пути движения от листа к корню дерева.

Пример 4.1.

В процессе обработки сообщения *M(k)= z(1),z(2), ... ,z(k),* состоящего из *k=32* символов и содержащих *K* различных знаков, построено хаффменовское дерево (рис.4.2). Вид и количество символов изображены на рисунке. Пусть следующим *z(k+1)*-м символом, поступающим от источника, является *"b"*. Требуется определить, какое кодовое слово будет выдавать кодер и модифицировать дерево после обработки *M(k+1)*-го сообщения, чтобы оно имело свойства хаффменовского.

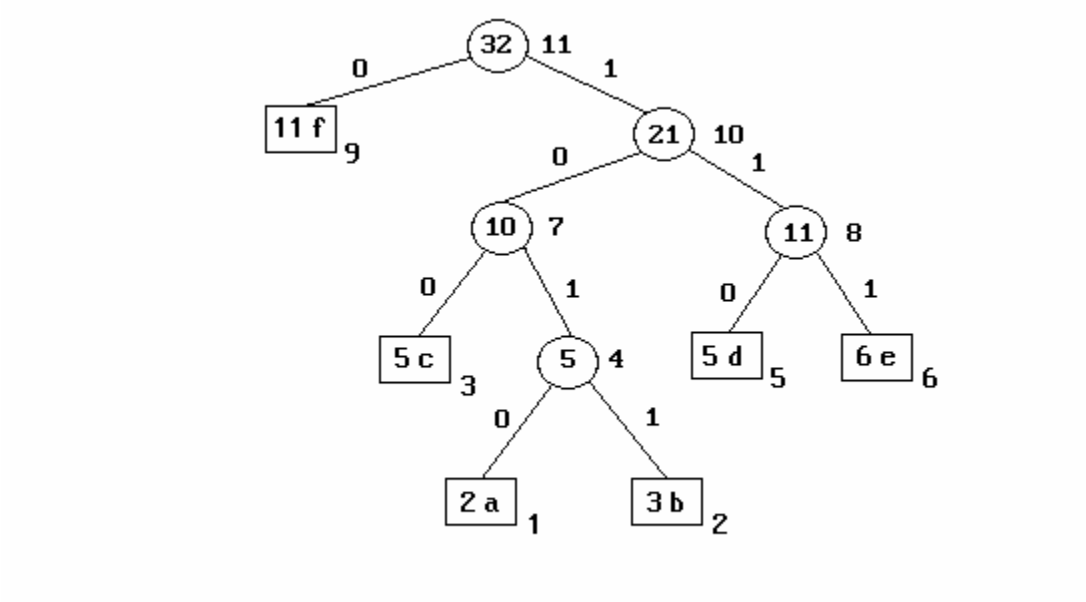


Рис.4.2. Дерево Хаффмена для сообщения, состоящего из 32-х символов

Из рисунка видно, что при поступлении символа *"b",* кодер выдаст кодовое слово *1011*, а вес узлов 2, 4, 7, 10 и 11 после этого должен увеличиться на 1, что нарушило бы порядок их расположения. Поэтому,

прежде чем инкрементировать веса узлов, преобразуем дерево в соответствии с вышеизложенной методикой.

Так как обрабатываться будет символ *"b",* то текущим узлом является узел 2. На дереве нет больше узла с весом *W*=3. Поэтому никакого обмена производиться не будет, а новым текущим узлом назначается 4-й узел, который образует поддерево с узлами 1 и 2.

Меняем местами узлы 5 и 4 (в составе всего поддерева) и назначаем следующим текущим узлом 8-й узел. Самым высоким номером такого же веса обладает узел 9, с которым и осуществляется обмен узла 8 вместе с образованным им поддеревом (узлы 1,2,4 и 6). Новым текущим узлом назначается 11-й. Но так как он является корневым, то на этом перестановка на дереве завершается.

Транспонированое дерево изображено на рис.4.3. Как видно из рисунка, дерево сохранило свои свойства. Правомочность таких перестановок вытекает из того, что перестановка узлов (в том числе совместно с образованными ими поддеревьями) с одинаковым весом не нарушает свойств оптимального хаффменовского дерева.

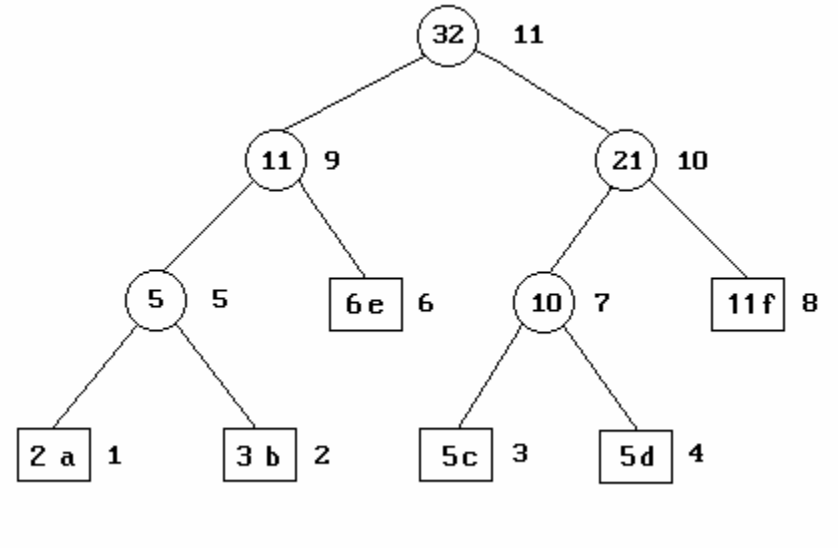


Рис.4.3. Транспонированное дерево Хаффмена

Увеличим теперь на единицу вес листа 2 и всех внутренних узлов (5, 9 и 11), расположенных на пути движения от листа к корню дерева. Дерево примет вид (рис.4.4*),* причем свойства хаффменовского дерева после инкрементации не нарушаются, так как при обмене узлами текущий узел располагался на позиции с максимальным порядковым номером такого же веса. Т.е., соседний узел имел вес минимум на единицу больше веса текущего узла. Новая кодовая комбинация, соответствующая символу "b", будет на один бит короче и имеет вид *001*.

Если построение дерева Хаффмена не завершено, то для отображения на дереве оставшихся *(m-K)* букв алфавита используют

вспомогательный узел с нулевой вероятностью появления символа, т. е. весом $W_o = 0$. Тогда при поступлении очередного символа *z(k+1),* отсутствующего на дереве, нулевой узел расщепляется на два листа, а сам становится родителем вновь созданных узлов. Один из листьев *(левый)* является новым нулевым узлом и соединяется нулевой ветвью с родительским узлом, а второй отображает вновь поступивший символ и соединяется единичной ветвью со своим родителем. Дальнейшее построение и модификация дерева осуществляется в соответствии с описанной выше процедурой.

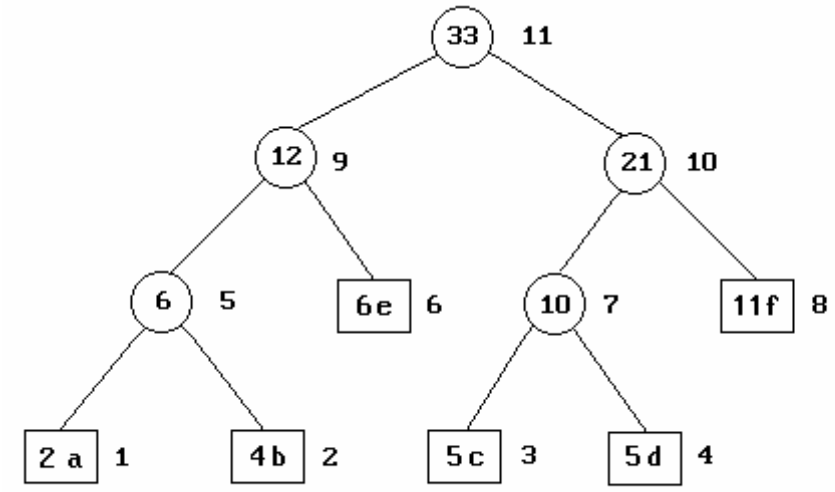


Рис.4.4. Дерево Хаффмена для сообщения из 33-х символов

Основой алгоритма FGK хаффменовского кодирования-декодирования является процедура *построения и динамической модификации кодового дерева* Хаффмена. Алгоритм модификации дерева сводится к следующим формальным действиям:

Алгоритм 4.1

НАЧАЛО

q:= лист дерева, соответствующий символу *z(k+1);*

ЕСЛИ *q* является нулевым узлом и $K < (m - 1)$, ТО

Н а ч а л о

Заменить *q* нулевым узлом ветвления с двумя дочерними узлами и пронумеровать их в возрастающем порядке, начиная с левого подчиненного узла: *левый, правый и родительский*;

q:= только что созданный *правый дочерний узел*

```
    К о н е ц.
  ЕСЛИ  q  и нулевой узел являются дочерними узлами
                одного родителя  ТО
    Н а ч а л о
        Поменять местами q и лист, имеющий максимальный
                номер того же самого веса;
        Увеличить вес q на единицу;
        q:= родитель q   {Новым текущим узлом становится
                родитель q}
    К о н е ц;

      ПОКА  q не стало корнем дерева  ВЫПОЛНЯТЬ
    Н а ч а л о   {Основная петля}
        Обменять q с узлом с максимальным номером
                того же самого веса; {q является теперь узлом
                с максимальным номером того же веса};
        Увеличить вес q на единицу;
        q:= родитель q   {Новым текущим узлом становится
                родитель q}
    К о н е ц

  КОНЕЦ.
```

Рассмотрим алгоритм построения и модификации кодового дерева Хаффмена на примере кодирования строки символов "abcbdefe". Последовательность преобразований показана на рис.4.5 - 4.8. Кодер и декодер начинают строить кодовое дерево с корневого узла *(*рис.4.5,*а*), соединенного нулевой ветвью с листом с нулевым весом, обозначенного символом *. Существует лишь один такой узел в дереве и его позиция (кодовое слово) изменяется при построении хаффменовского дерева. Кодер считывает первый символ строки "a" и создает новый узел. Так как на каждом уровне, за исключением корневого, должно быть не менее двух узлов, имеющих общий родительский узел, кодер соединяет его с корневым узлом единичной ветвью (рис.4.5,*б*). Образованный лист является первичным месторасположением символа "a". На приемную сторону эта буква передается в несжатом виде стандартным *ASCII*-кодом.

Дерево декодера на начальном этапе имеет вид, подобный дереву кодера до получения первого символа (рис.4.5,*а*) и таблица кодов Хаффмена остается пустой. Декодер интерпретирует этот символ как неподвергавшийся сжатию с дальнейшим обозначением его на дереве аналогично передающей стороне.

Для каждого последующего символа кодер и декодер производят сверку с символами, которые к этому времени представлены на дереве. Если символ уже имеется, то он передается в сжатом виде, причем кодовое слово определяется согласно позиции символа на дереве, где старший разряд является первым со стороны корня.

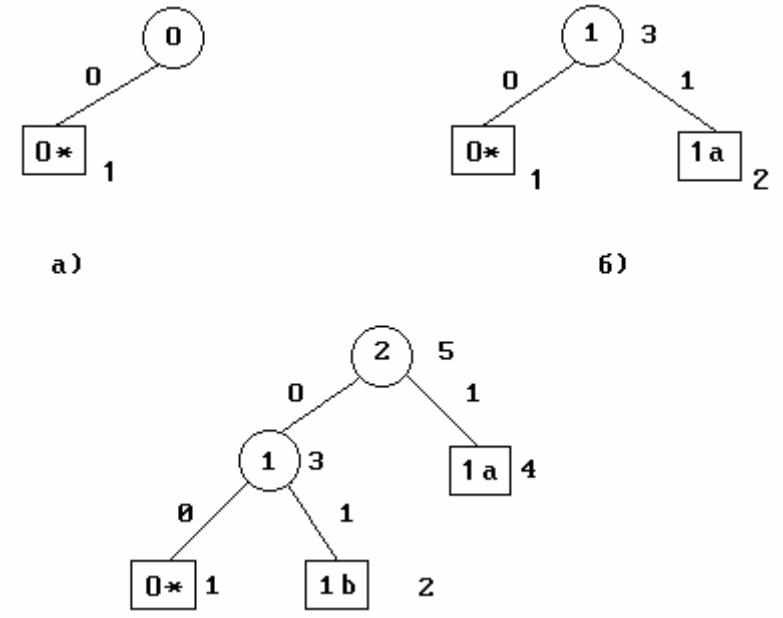


Рис.4.5. Пример построения и модификациии хаффменовского дерева для сообщения “abc”

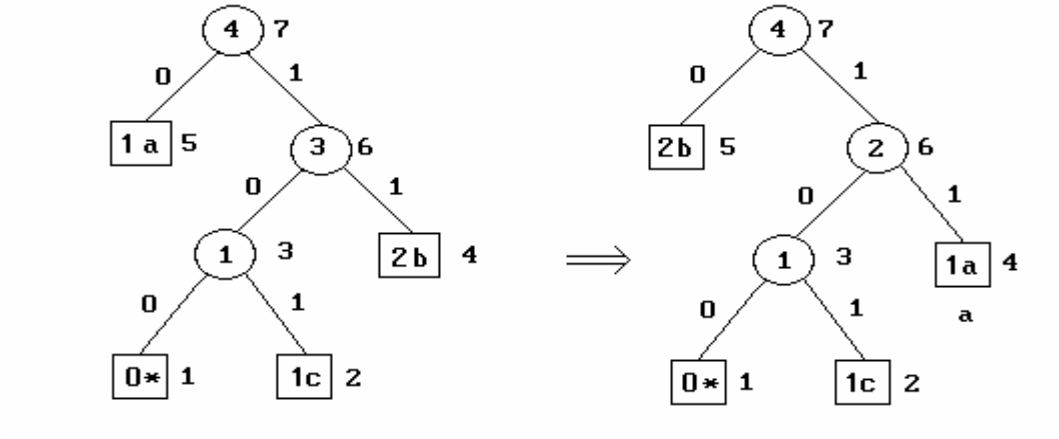
Если символ не обнаружен, то он выдается в несжатом виде, а нулевой узел превращается в родительский, расщепляясь на пару листьев: нулевой, соединенный нулевой ветвью с родительским и лист для нового

символа, соединенный единичной ветвью. Таким образом, при поступлении от источника буквы "b" кодовое дерево будет иметь вид, изображенный на *рис.4.5,в*. Как видно из рисунка, дерево является хаффменовским.

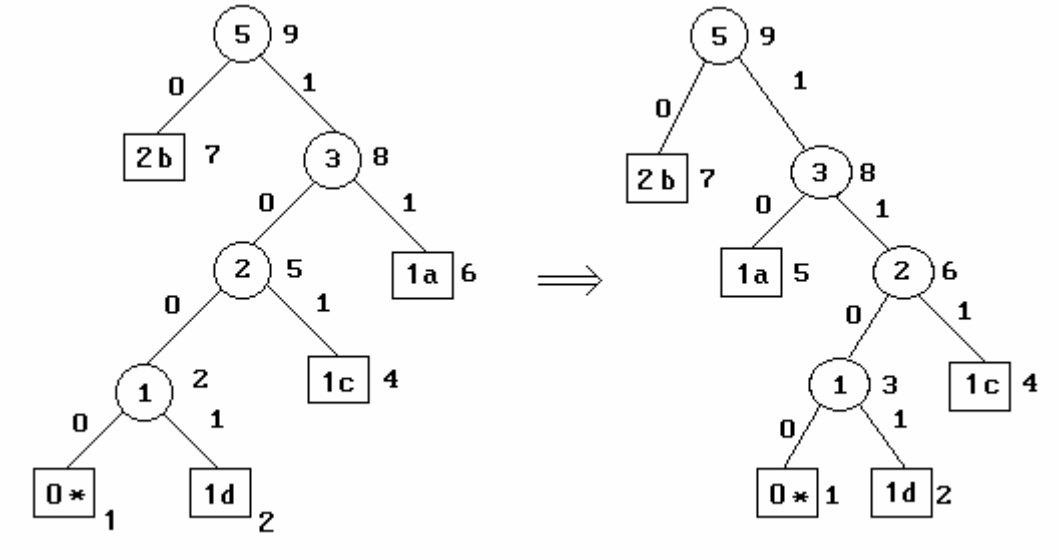
Введение в дерево третьего символа "c" нарушает свойства хаффменовского дерева и оно нуждается в модификации. Текущим узлом дерева на рис.4.4,г является лист 2. Так как символ "c" появился впервые, то вес $q = 0$. В связи с тем, что на дереве нет листьев, имеющих больший порядковый номер с нулевым весом, то увеличиваем q на 1 и назначаем новым текущим узлом родителя узла 2. Узел 3 имеет пока еще нулевой вес и нет узла аналогичного веса с большим порядковым номером. Поэтому обмена узлами не происходит, вес узла 3 увеличивается на единицу и следующим текущим узлом назначается 5-й узел, имеющий на данный момент вес 1. Обмениваем его вместе с образованным им поддеревом (узлы 1,2,3 и 4) с узлом 6 и увеличиваем вес q на единицу. Следующим текущим узлом назначается узел 7, который является корневым. Следовательно модификация дерева завершена. Скорректированное дерево Хаффмена изображено на правой части рис.4.5,г.

Появление на входе кодера символа "b", который уже отмечен на дереве, также нарушает его свойства. Изменение конфигурации дерева осуществляется в соответствии с алгоритмом. Текущим узлом теперь является лист 4, имеющий вес $W=1$. Обмениваем его с узлом 5 и увеличиваем вес текущего узла на единицу. На последующих рис.4.6 — 4.8 показаны исходные и модифицированные кодовые деревья при поступлении остальных символов сообщения. Как видно из примера реализации алгоритма, обе фазы модификации кодового дерева совмещены.

Важнейшей проблемой при построении динамических кодов Хаффмена является однозначность декодирования в случае приема сообщения, когда еще не все символы используемого источником алфавита отмечены на кодовом дереве. Без специальных мер декомпрессор не в состоянии различить, отображает ли поступившая кодовая комбинация несжатый 8-ми битовый символ, либо это одна или несколько кодовых комбинаций неравномерного кода Хаффмена, уже отмеченных на дереве. Например, если в момент передачи несжатого символа 8-ми разрядной кодовой комбинации *01111101* в кодере и декодере было сформировано дерево, изображенное на рис.4.6,б, то это кодовое слово будет воспринято как три комбинации неравномерного кода: *0, 111* и *1101*. Получателю при этом будет выдана последовательность символов b, c и d.



а)

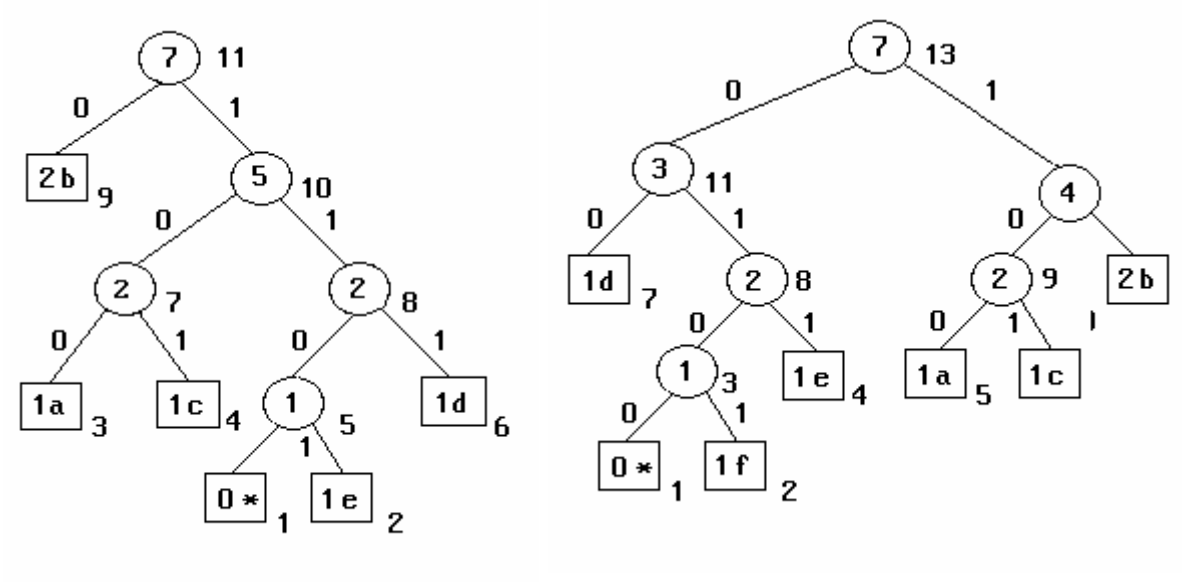


б)

Рис.4.6. Модификация дерева после получения второго символа “b”

Одним из способов решения этой проблемы является создание в кодере и декодере хаффменовского дерева, построенного для всего используемого алфавита до начала процедуры компрессии и декомпрессии. Затем при запуске алгоритма сжатия в кодере и декодере осуществляется синхронная модификация деревьев с учетом частоты появления символов в кодируемом сообщении.

При построении исходного дерева можно предположить, что все символы являются равновероятными. Это позволяет избежать длинных кодовых комбинаций на начальном этапе модификации дерева.



а) б)

Рис.4.7. Модификация дерева после получения второго символа “e” (а) и после поступления символа “f”

Недостатком такого способа является дополнительный расход памяти для хранения конфигурации начального дерева, а также затраты времени на манипуляции с деревом, на котором отмечены все возможные символы алфавита, в то время как в сжимаемом сообщении используется лишь часть из них.

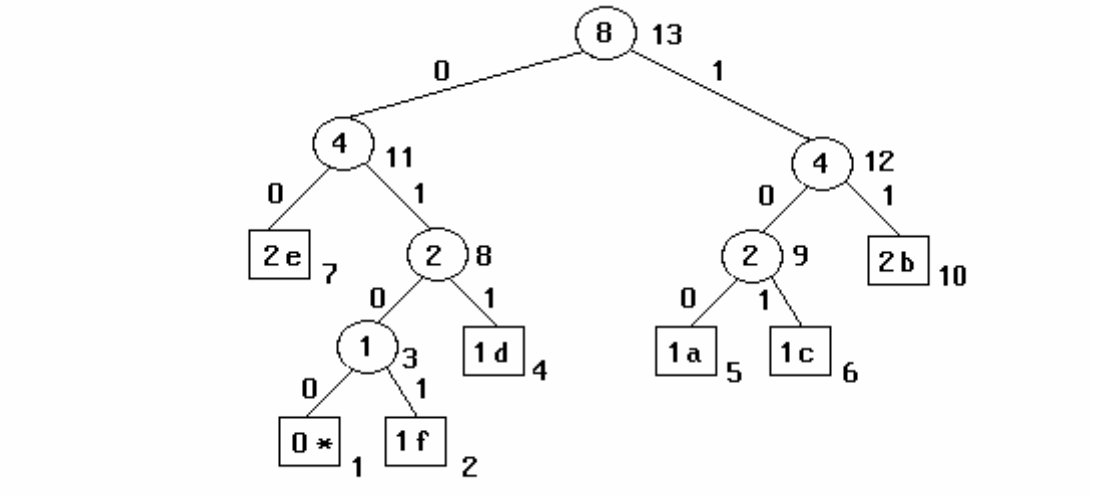


Рис.4.8. Дерево Хаффмена после получения последнего символа сообщения

Оригинальный способ решения этой проблемы был предложен Д.Кнутом [32]. Его суть состоит в следующем. Все символы используемого

алфавита, которые еще не появились на выходе источника, отмечаются на дереве нулевым узлом. Поэтому, когда символ, подлежащий сжатию, генерируется впервые, кодер "отмечает" его, вырабатывая *префиксную комбинацию.* Вслед за ней на выход поступает код несжатого символа. В качестве префиксной комбинации используется код, соответствующий *листу нулевого веса*. Для кодирования символов, поступивших от источника впервые, используется не 8-ми разрядная комбинация *ASCII*-кода, а минимальный префиксный код, построенный на основании следующих соображений. Если имеется сообщение, состоящее из m различных символов $a_1,..., a_m$, то число m можно представить в виде целой степени двойки и целого числа r

$$m = 2^e + r \ , \qquad (4.1)$$

где $0 \le r < 2^e$. Тогда a_k -й символ кодируется $(e + 1)$ — битовой комбинацией числа k-1, если $1 \le k \le 2r$. В противном случае - e-битовой комбинацией числа $k - r - 1$. Например, если $m = 6$, то $e = 2$ и $r = 2$, а символы источника будут отображаться комбинациями:

a_1=> 000; a_2 => 001; a_3 => 010; a_4 => 011; a_5 => 10; a_6 => 11.

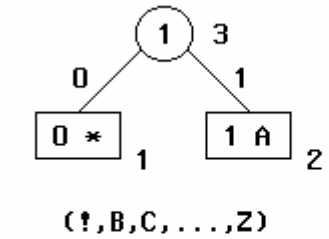
Нетрудно заметить, что полученный код обладает свойством префикса. Этот код является оптимальным, если символы имеют одинаковую вероятность.

Пример 4.2.

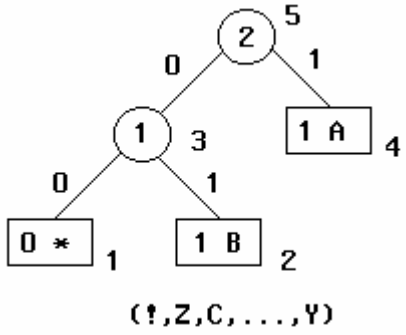
Построить кодовое дерево и осуществить сжатие динамическим кодом Хаффмена фразу "*ABRACADABRA*!". Здесь символ ! означает конец строки. В качестве алфавита источника использовать только буквы латинского алфавита.

Перед началом компрессии кодеру и декодеру известен только размер алфавита и алгоритм кодирования. В данном случае $m = 27$. Представим m в форме (4.1), т.е. $m = 27 = 2^4 + 11$. Поскольку A является первой буквой алфавита, то $k = 1$. Следовательно, комбинация префиксного кода, отображающая символ A, будет *00000*. Эта комбинация поступает на выход кодера, а символ A отмечается на кодовом дереве листом с весом 1, который соединен единичной ветвью с корнем дерева (рис.4.9,а). Декодер легко расшифровывает первую комбинацию благодаря префиксным свойствам используемого кода. Лист нулевого веса дерева отображает оставшиеся символы алфавита. Чтобы не перемещать все элементы массива после исключения A из списка символов алфавита, он

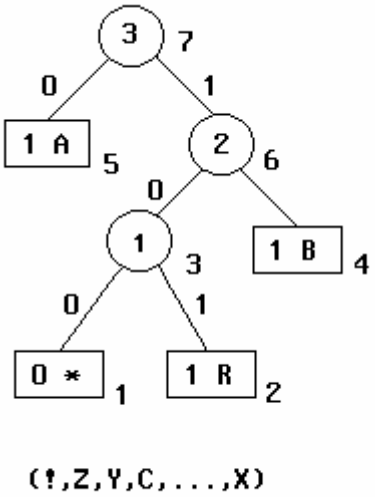
обменивается местом с последним элементом списка, которым является знак !.



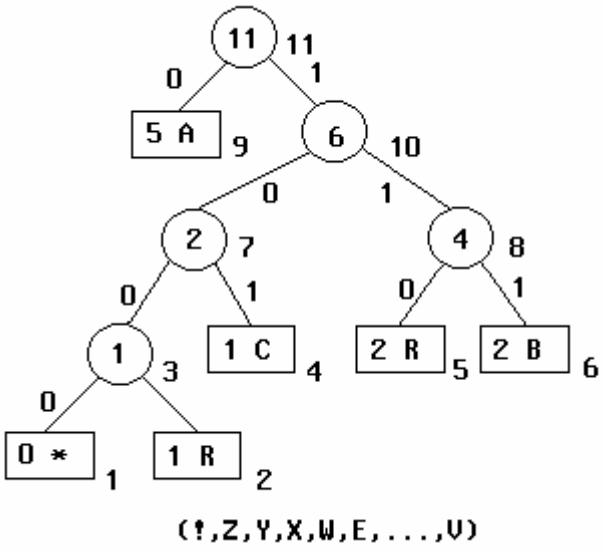
а)



б)



в)



г)

Рис.4.9. Пример построения кодовых деревьев для последовательности “ABRACADABRA”

С поступлением от источника символа *B* на выход кодера выдается *0 - код нулевого узла*, а за ним кодовая комбинация, отображающая *B*, которая определяется по формуле (4.1). Поэтому *B => 00001*. Декодер, зафиксировав первым нулевой бит, определяет, что это лист нулевого веса. Следовательно, за ним будут поступать биты символа, который еще не подвергался компрессии. Дерево Хаффмена при этом приобретает вид рис.4.9,б. Буква *R* является 18-й в латинском алфавите, т.е. *k* = 18, а соответствующий ей код - *10001*.

В связи с тем, что код нулевого узла дерева равен *00*, то символ *R* будет закодирован семиразрядной комбинацией *0010001*. Кодовое дерево после его модификации примет вид, показанный на рис.4.9,в. Продолжая

процедуру кодирования до завершения строки, получаем набор кодовых комбинаций, соответствующих поступившим символам:

A => 0; *C* => 100 00010; *A* => 0; *D* => 1100 00011;
A => 0; *B* => 110; *R* => 110; *A* => 0.

Дерево Хаффмена, построенное к этому моменту времени, изображено на рис.4.9,г. Конечный элемент строки ! будет закодирован комбинацией *1000 00000*.Как видно из рассмотренного примера, предложенный Д.Кнутом способ, обеспечивает однозначность декодирования в процессе построения хаффменовского дерева и снижает количество битов, затрачиваемых на кодирование символов, поступающих от источника впервые.

4.3. Алгоритм динамического кодирования Виттера

Дальнейшее усовершенствование алгоритма динамического кодирования данных неравномерными кодами FGK было предложено Д.Виттером [42]. Этот алгоритм получил название *алгоритма V*. При поиске путей оптимизации процедуры кодирования данных кодом Хаффмена автор исходил из того, что в новом алгоритме число обменов узлов в процессе модификации кодового дерева должно ограничиваться некоторым малым числом (в лучшем случае единицей), а динамическое хаффменовское дерево должно строиться таким образом, чтобы минимизировать не только суммарную длину внешнего пути $\sum W_j l_j$, но и величины $\sum l_j$, $max\{l_j\}$. Минимизация высоты дерева $H = \max\{ l_j \}$ позволит предотвратить образование длинных кодовых комбинаций при кодировании очередного символа в сообщении. Виттеру в значительной степени удалось решить поставленную задачу. Разработанный им алгоритм обладает по сравнению с алгоритмом FGK следующими двумя преимуществами.

1. Количество обменов узлами, при котором текущий узел перемещается вверх по кодовому дереву в процессе его модификации, ограничивается единицей. В алгоритме FGK верхняя граница числа обменов составляет l_j / 2, где l_j - длина кодового слова для $z_j(k+1)$-го символа до начала процедуры модификации.

2. Алгоритм *V* минимизирует длину внешнего пути дерева l_j и гарантирует дерево минимальной высоты $H = \max\{l_j\}$ при условии минимизации суммарной длины внешнего пути дерева $\sum W_j l_j$.

Суть усовершенствования алгоритма *V* состоит во введении новой системы нумерации узлов кодового дерева, которая получила название неявной нумерации *(Implicit numbering)*. При неявной нумерации узлы хаффменовского дерева нумеруются в порядке увеличения по уровням слева направо и снизу вверх, т.е., узлы более низкого уровня имеют номера меньше, чем узлы последующего уровня. Важнейшей особенностью неявной нумерации является соблюдение необходимого условия построения дерева, которое формулируется следующим образом:

Для каждого веса *W* все внешние узлы (листья) дерева с весом *W* должны предшествовать всем внутренним узлам веса *W*.	(*)

Нетрудно заметить, что это условие является одной из отличительных черт неявной нумерации по сравнению с другими алгоритмами.

Нумерация узлов, производимая согласно алгоритмому FGK, не всегда соответствует неявной нумерации. Так например, хаффменовское дерево, изображенное на рис.4.3, соответствует условию неявной нумерации, в то же время как нумерация узлов в кодовых деревьях, показанных на рис.4.7-4.8, отличается от неявной тем, что внутренний узел с весом *W=1* предшествует листьям с таким же весом. Очевидно, что если на рис.4.8 поменять местами узлы 3 и 6 , а на рис.4.7 - узлы 5 и 6, то нумерация в модифицированном дереве будет удовлетворять условию (*).

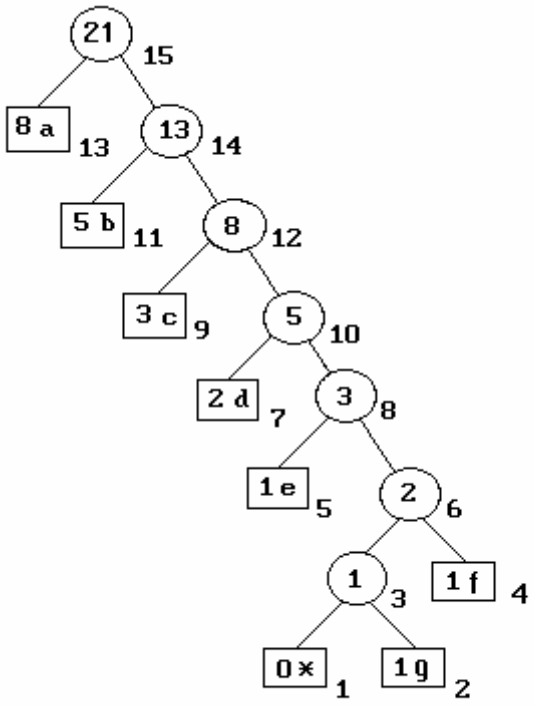


Рис.4.10. Пример модификации дерева Хаффмена по алгоритму Виттера (этап 1)

Для иллюстрации целесообразности применения алгоритма *V* покажем на примере результат модификации хаффменовского дерева

(рис.4.10) в соответствии с неявной нумерацией. Как видно из этого рисунка кодовое дерево отвечает всем требованиям оптимального хаффменовского дерева, но порядок расположения узлов одного веса не соответствует необходимому условию (*) для дерева с неявной нумерацией. Так внутренний узел с весом 1 расположен между двумя листами с таким же весом.

Внутренний узел с весом 2 имеет меньше номер, чем лист с весом 2 и т.д. А по условию (*) неявной нумерации для каждого веса W лист с

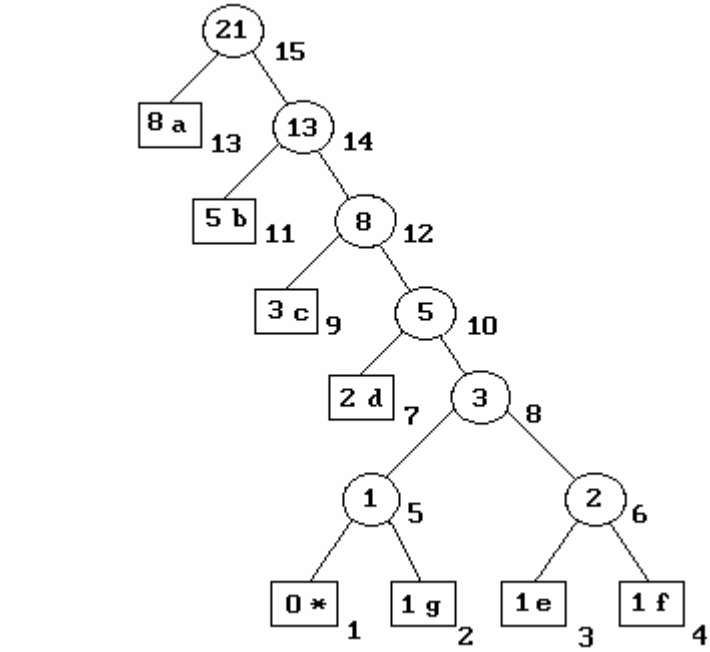


Рис.4.11. Пример модификации дерева Хаффмена по алгоритму Виттера (этап 2)

весом W_k должен предшествовать внутреннему узлу с весом W_{k}. Для того, чтобы листья с весом 1 (рис.4.10) предшествовали внутреннему узлу с весом 1 необходимо узлы 3 и 5 поменять местами. Не забывайте, что внутренний узел должен перемещаться со всем образованным им поддеревом, так как его вес включает веса дочерних узлов. В результате обмена дерево принимает вид (рис.4.11).

Затем, после перестановки узлов 6 и 7, дерево имеет конфигурацию, изображенную на рис.4.12. Проводя последовательно аналогичным образом перемещение узлов, окончательно получим модифицированное дерево, представленное на рис.4.13. Общее количество битов, затраченных на кодирование сообщения, для которого построено хаффменовское дерево (рис.4.10 и рис.4.13) для обоих алгоритмов одинаково, т.е. $\sum W_j l_j$. Однако

длина внешнего пути модифицированного дерева $\sum l_j$ уменьшилась с 28 до 22, а его высота $H = \max\{l_j\}$ - с 7 до 4-х.

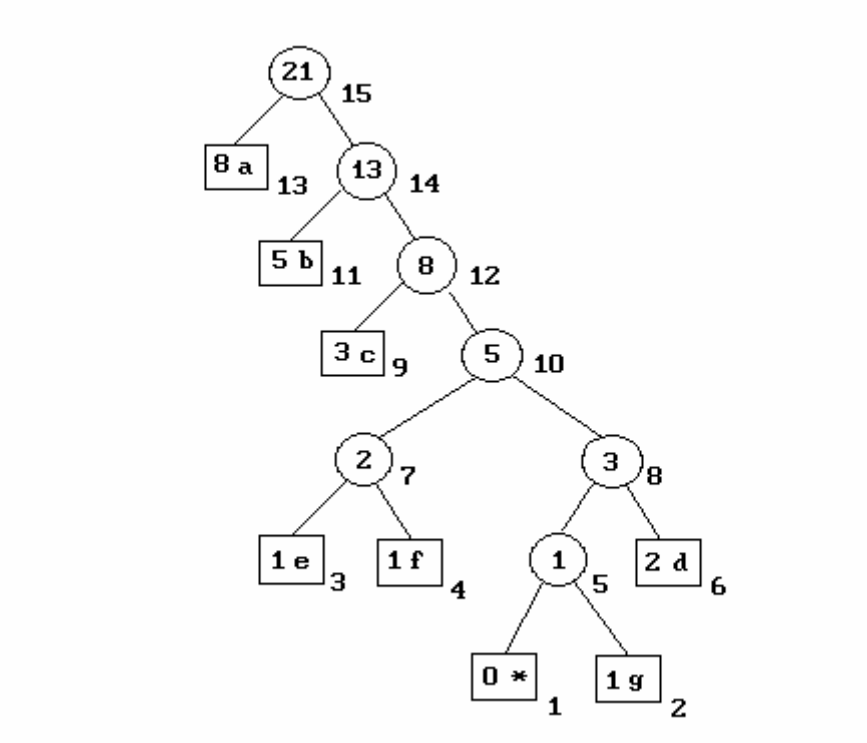


Рис.4.12. Пример модификации дерева Хаффмена
по алгоритму Виттера (этап 3)

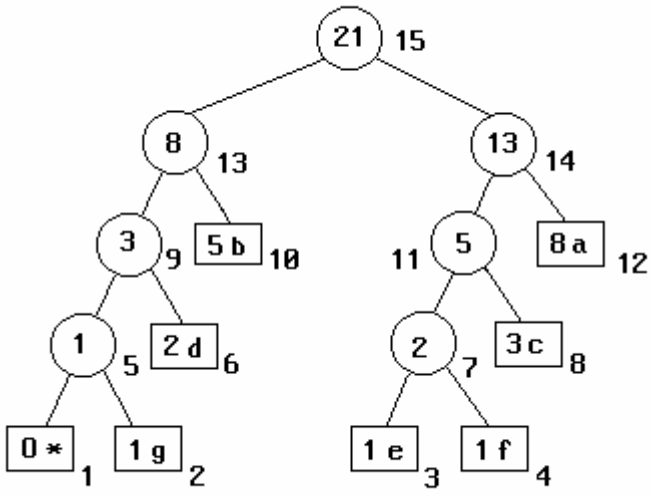


Рис.4.13. Модифицированное по алгоритму Виттера
кодовое дерево Хаффмена

Кодовые комбинации для отображения одинаковых символов одного и того же сообщения будут различны. Так при поступлении от источника следующего символа f он будет закодирован по алгоритму *FGK* словом *111111*, а по алгоритму *V* отображающая его комбинация *1001* будет на два бита короче. Поэтому, если передача продолжится с символов d,c,g,e,f или с еще не использованных символов, то формируемые кодовые слова в алгоритме *V* будут короче, чем в *FGK*, т.е. стратегия минимизации внешнего пути и высоты дерева оптимальна при предположении, что любой появляющийся символ равновероятен.

Второй отличительной чертой алгоритма *V* является введение понятия блока эквивалентных узлов. При этом узлы *x* и *y* эквивалентны, если они имеют одинаковый вес и оба являются либо внутренними, либо внешними. Узел блока, имеющий (при неявной нумерации) самый высокий номер, называется *лидером* блока. Блоки упорядочиваются по увеличению веса, причем блок листьев веса *W* должен предшествовать блоку внутренних узлов того же самого веса.

Третья отличительная особенность алгоритма *V* заключается в способе модификации дерева после получения очередного символа. Главной операцией алгоритма по поддержанию условия неявной нумерации (*) является скольжение и приращение (*SlideAnd Increment*). Суть этой операции состоит в том, что узел, объявленный текущим, обменивается с лидером своего блока и затем скользит в направлении корня дерева по соседнему блоку, который непосредственно примыкает к блоку текущего узла. Скольжение продолжается до тех пор, пока текущий узел не пройдет весь блок и будет установлен во главу этого блока. Затем осуществляется приращение веса текущего узла и новым текущим узлом назначается родитель старого текущего узла. Операция скольжения с приращением продолжается до достижения корня дерева. При этом выбор родительского узла зависит от того, являлся ли текущий узел листом или внутренним узлом. Если текущий узел был листом, то новым текущим узлом становится родитель, с которым оказался связан текущий узел после завершения скольжения. А в случае, если текущим узлом был внутренний узел, то новым текущим узлом назначается его родительский узел, с которым был связан текущий до начала скольжения.

На рис.4.14 приведен пример процедуры *Slide AndIncrement.* На рис.4.14*а* (1) изображена ситуация, когда текущим узлом *p* является лист с весом 4, который в процессе обмена стал лидером блока листьев с весом 4. На рис.4.14*б* (1) приведен случай, при котором текущим узлом является внутренний узел с весом 8. Очевидно, что если текущий узел образует блок, состоящий из одного узла, то этот узел является и текущим и лидером одновременно. Все узлы в блоке сдвигаются влево на одну позицию, чтобы освободить ячейку памяти для текущего узла *p*. В

процессе скольжения узел *p* занимает в соседнем блоке самую крайнюю позицию справа, то есть становится лидером блока (Рис.4.14*а,б* позиция (2). На позиции (3) обеих рисунков показана ситуация после приращения веса текущего узла на 1, а на позиции (4) - новое положение текущего узла.

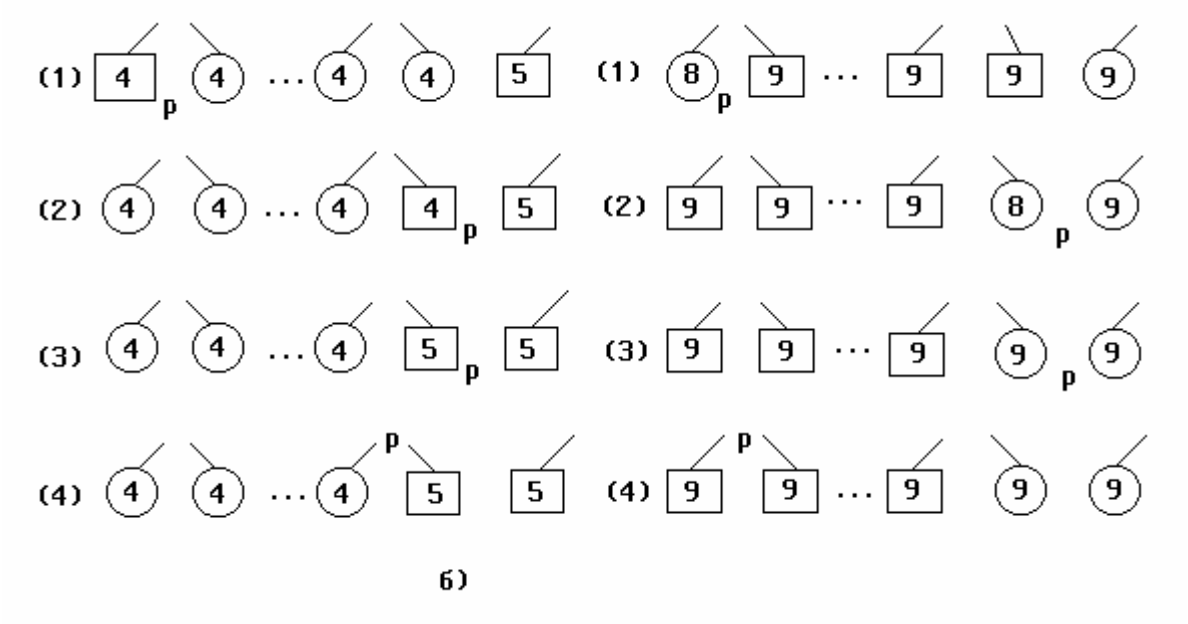


Рис.4.14. Модификация кодового дерева путем скольжения и приращения

Алгоритм Виттера динамического сжатия данных сводится к следующему:

Алгоритм 4.2

```
        Procedure  Vitterupdate;
   BEGIN
LeafToIncrement:=0; {Установить начальное
                                значение переменной}
  q:= лист дерева, соответствующий символу Z i(k+);
IF  (q является 0-м узлом) и (K < N - 1)   THEN
    Begin                    { Специальный случай N1 }
     Заменить  q  на внутренний 0-й узел дерева и
              создать два его дочерних узла: левый, нулевой узел и
              правый, содержащий символ  Z i (k+1);
  q:=внутренний, только что созданный узел; {Определить текущий
              узел}
LeafToIncrement :=правый дочерний узел текущего узла q
    End
```

```
        ELSE  Begin
              Обменять q с лидером его блока;
              IF  q образует пару с 0-м узлом   THEN
                Begin              { Специальный случай N 2}
                  LeafToIncrement:= q;
                  q:= родитель q
                End
              End;
    WHILE  q не является корнем хаффменовского дерева  DO
       {Основной цикл; q должно быть лидером этого блока }
            SlideAndIncrement (q);      {Вызов процедуры
                                          сколжение - приращение}
            IF  LeafToIncrement < > 0  THEN
              SlideAndIncrement(LeafToIncrement)
        END;

    ;-------------------------------------------------------------
          PROCEDURE  SlideAndIncrement (p);
        Begin
          wt:= вес p;          {p - текущий узел дерева}
          b:= следующий за p-м блок в компонуемом списке;
          IF  (( p-лист) AND (b-блок внутренних узлов
                                весом wt))
          OR  (( p-является внутренним узлом ) AND
              (b-является блоком листьев весом wt+1))  THEN
         Begin
              Скольжение p во главу блока узлов b;
              p-й вес:=wt+1;
     IF p является листом  THEN
          p:=новый родитель p {родитель p после скольжения}
    ELSE p:=прежний родитель p {родитель p до скольжения}
        End
    End.
```

СТРУКТУРА ДАННЫХ. При реализации алгоритма V компрессии сообщений необходимо так организовать структуру данных, чтобы она позволяла выполнять следующии функции:

- представлять двоичное дерево Хаффмена с положительными весами в соответствии с необходимым условием неявной нумерации (*);

- сохранять непрерывный список внутренних узлов дерева в порядке возрастания весов в соответствии с неявной нумерацией; аналогичный список составляется и для внешних узлов (листьев) дерева;

- позволять находить лидера блока узлов для любого заданного узла на основе неявной нумерации;

- обеспечивать обмен содержимым двух листьев одного и того же веса;

- обеспечивать увеличение на 1 веса лидера блока, скольжение его в следующий блок после перенумерации узлов следующего блока путем уменьшения на 1 номера каждого из его узлов;

- представлять соответствие между K символами алфавита, которые появились в сообщении, и листьями дерева;

- представлять m - K символов алфавита, которые еще не появлялись в сообщении, одиночным листом - узлом с нулевым весом на дереве Хаффмена.

Следует заметить, что структура данных использует явную нумерацию, которой соответствует физическое расположение ячеек памяти, используемых для хранения информации о узлах, и ее нельзя смешивать с неявной нумерацией, определенной выше. Листья дерева явно нумеруются в непрерывном порядке от 1 до m, а внутренние узлы - от m+1 до 2m - 1. При этом узел с номером q является листом, если $q \leq m$.

Имеется тесная связь между явной и неявной нумерациями. Так если для двух внутренних узлов p и q при неявной нумерации выполняется условие $p < q$, то оно справедливо и при явной нумерации, т. е. $p < q$. Это условие выполняется также и для двух листьев p и q .

Рассмотренная структура данных дерева называется "*плавающим деревом*" потому, что указатели родительского и его дочерних узлов поддерживаются неявно. Каждый блок имеет лишь указатели родительского и правого дочернего узлов лидера блока. Из-за непрерывности памяти, где хранятся внешние и внутренние узлы, положение родительских и дочерних узлов дерева других узлов блока могут быть определены за фиксированное время путем вычисления смещения от указателей родителя и его правого дочернего узла лидера блока. Это позволяет узлу скользить по блоку, модифицируя при этом постоянное число указателей. Таким образом, выполнение процедуры *SlideAndIncrement* занимает постоянное время, что позволяет осуществлять кодирование и декодирование в реальном времени. Это свойство является весьма важным при использовании компрессии в синхронных системах передачи данных.

4.4. Особенности программной реализации динамической компрессии данных неравномерными кодами

Для программной реализации процедуры динамического сжатия сообщений необходимо организовать данные таким образом, чтобы можно было осуществлять текущую оценку вероятностей появления символов кодируемой последовательности, а также поддерживать динамическую структуру кодового дерева Хаффмена. Оценка вероятности входного символа алфавита пропорциональна количеству появления этого символа в кодируемом сообщении. Поэтому вместо оценки вероятности целесообразно вести счет числа появлений для каждого символа входного алфавита, т.е. с приходом *i*-го символа состояние соответствующего ему *i*-го счетчика Cn_i увеличивается на 1. Для исключения возможности переполнения необходимо контролировать текущее число состояний всех используемых счетчиков. При достижении одним из них максимально допустимой величины следует умножать каждое из значений счетчика на некоторое фиксированное число $\propto_k$ <1. С целью упрощения реализации операции умножения и исключения уменьшения точности оценки за счет округления после умножения результата счета, целесообразно выбрать $\propto_k$ = 0,5. В этом случае умножение может быть заменено сдвигом содержимого счетчика на один разряд.

Следует заметить, что при реализации алгоритма сжатия необходимо обеспечивать счет для каждого узла текущего кодового дерева. На основании свойств дочерних узлов вероятностного кодового дерева Хаффмена, если эти узлы можно расположить по убыванию состояния счетчиков так, что каждый узел является смежным парному ему дочернему узлу, то код Хаффмена будет оптимальным для текущих оценок вероятностей входных символов.

Для отображения структуры текущего дерева Хаффмена в памяти вычислителя должен содержаться список дочерних пар узлов дерева. Для алфавита длиной *m* символов существует *m - 1* таких дочерних пар узлов. В структуру данных для каждой дочерней пары дерева должны входить счетчики весов левого (нулевого) *Count*0 и правого (единичного) *Count*1 узлов, а также указатели, индицирующие связь этой пары с ее родительским узлом *FP*, и связи нулевого и единичного узлов с порожденными ими дочерними узлами. Первый тип указателей называют прямыми *FP*, а второй — обратными *BP* указателями. Так как в соответствии со свойствами хаффменовского дерева каждый дочерний узел имеет парный ему узел, то указатели обоих типов указывают сразу на пару узлов дерева. В связи с тем, что каждая дочерняя пара узлов в общем

случае имеет один родительский и две пары дочерних узлов, то для описания ее связей необходимо иметь один прямой *FP* и два обратных *BP0* и *BP1* указателя. При этом в прямом указателе имеется однобитовый индикатор, показывающий, соединен ли родитель данной пары со своим родителем нулевой или единичной ветвью. Очевидно, что указатель дочерней пары, родителем которой является корень дерева, равен нулю. Обратные указатели левого (нулевого) *BP0* и правого (единичного) *BP1* узлов выше расположенной пары дочерних узлов кодового дерева указывают соответственно на порожденные ими ниже лежащие дочерние пары. Обратите внимание на то, что если дочерняя пара состоит из одного или двух внешних узлов, то указатель *BP0* индицирует лист дерева, соединенный со своим родителем нулевой ветвью, а *BP1* - единичной.

Таким образом, в структуру данных для каждой дочерней пары входит 5 полей, два из которых отображают текущее состояние счетчиков для левого и правого дочерних узлов, а три остальных являются указателями. Структура данных должна быть организована так, чтобы каждые значения счетчиков для верхней дочерней пары были выше или равны значениям счетчиков предыдущей пары и так далее до конца списка.

На рис. 4.15 показан пример хаффменовского кодового дерева, а на рис.4.16 — соответствующая ему структура данных, на которой стрелками показаны ветви дерева, а цифры возле них - вид связи. Здесь же условно изображены состояния бит-индикаторов соответствующих прямых указателей.

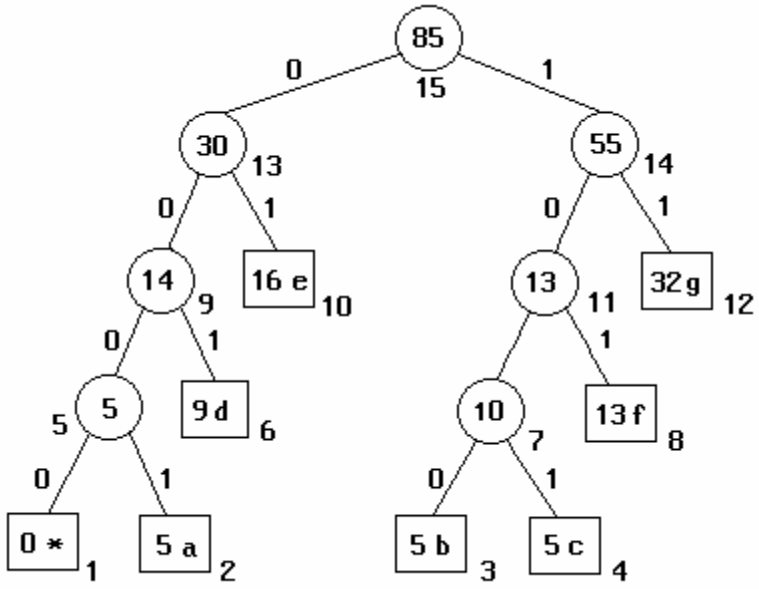


Рис.4.15. Пример кодового дерева для иллюстрации структуры данных

Символы источника занимают отдельную область памяти, а каждый из символов содержит только указатель текущего расположения в списке дочерних пар узлов (рис. 4.16).

В поле указателя находится бит-индикатор, который отображает вид связи листа дерева с его родительским узлом (0 или 1). Рассмотрим

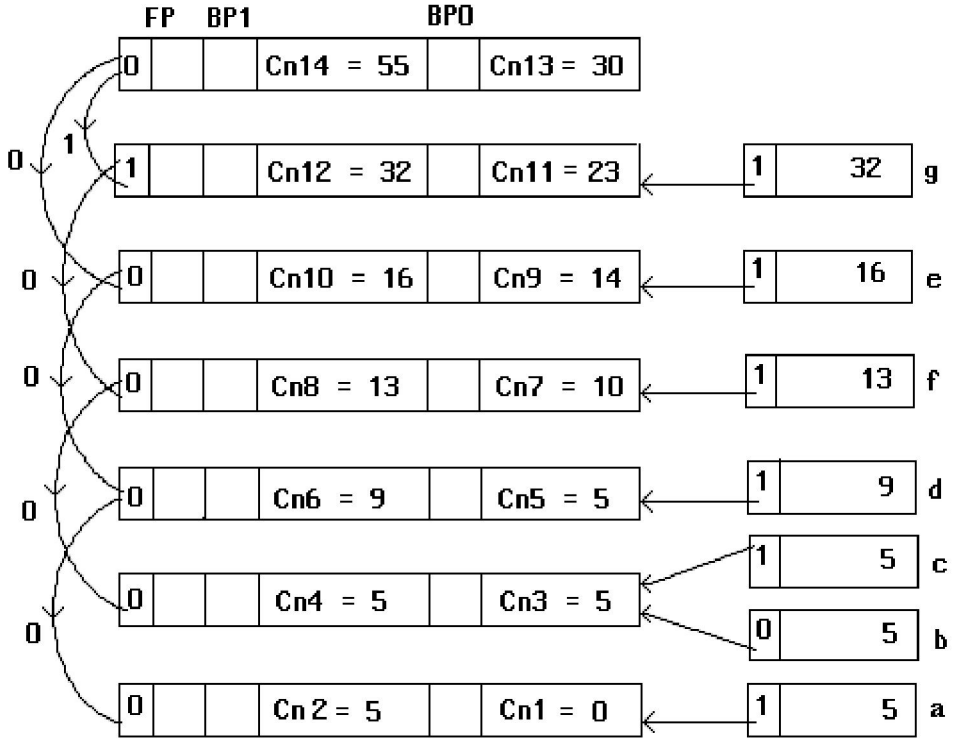


Рис.4.16. Структура данных для представления кодового дерева Хаффмена

ситуацию, когда после кодирования сообщения, состоящего из 85 символов, было построено кодовое дерево (см. рис.4.15).

Затем от источника поступает следующий символ *f*. В этом случае младшим разрядом отображающего его кодового слова будет 1, так как бит-индикатор указателя символа *f* указывает на единичную ветвь, соединяющую внешний узел 8 дерева с внутренним 11. Вторым разрядом слова будет 0, в связи с тем, что бит-индикатор указателя пары дочерних узлов 7 и 8 отображает нулевую ветвь, связывающую их родителя (узел 11) с родителем пары (узлом 14), в которую входит узел 11. И последним, старшим, разрядом кодового слова символа f будет 1, так как указатель пары узлов (11 и 12) установлен в 1, потому что родитель дочерней пары узлов 11 и 12, узел 14, связан со своим родителем 15 единичной ветвью.

После считывания кода символа f производится модификация дерева в соответствии с используемым алгоритмом. Перемещение узлов осуществляется путем изменения их прямых указателей. Если необходимо поменять два узла местами, то достаточно выполнить обмен указателями этих узлов. Увеличение на 1 веса одного из листьев дерева приводит к необходимости инкрементировать содержимое всех внутренних узлов, включая корень, на пути от заданного листа до корня.

Процедура декодирования начинается с корня дерева. Пусть на вход декомпрессора поступает последовательность битов вида *0011...* . После получения первого бита 0 указатель *BP0* узлов дочерней пары корня укажет на узел 13. И если этот узел не является листом, то с приходом второго нуля *BP0* 13-го узла выведет на узел 9. Так как этот узел внутренний, то декодирование продолжается. При поступлении третьего бита 1 указатель узла 9 укажет на ячейку памяти, где находится декодируемый символ *d*.

На этом декодирование очередного символа заканчивается, производится модификация дерева. В данном случае она сводится к увеличению на 1 веса всех узлов, включая лист и корень, на пути от листа к корню дерева. С приходом следующей 1 декодирование пойдет по правой ветви дерева. Такая структура данных позволяет экономно расходовать память вычислителя и заметно упрощает процесс разработки программ кодирования и декодирования.

Контрольные вопросы к разделу 4

1. В чем состоит отличие динамического метода кодирования от статического?

2. Назовите свойства хаффменовского кодового дерева.

3. Покажите на конкретном примере, как декодер разделяет принимаемые символы и различает, что принятая последовательность битов относится к символу, передаваемому впервые.

4. Проиллюстрируйте на примере, как осуществляется построение “на лету” кодового дерева на декодирующей стороне.

5. Объясните на примере процедуру построения и модификации кодового дерева по алгоритму FGK.

6. Как следует организовать структуру данных при динамическом кодировании Хаффмена ?

7. Приведите пример описания структуры кодового дерева на языке Pascal.

Глава 5

ДИНАМИЧЕСКОЕ КОДИРОВАНИЕ СТРОК ПЕРЕМЕННОЙ ДЛИНЫ

5.1. Кодирование методом Лемпеля-Зива

Независимо от языка и характера текстовых сообщений в различных его частях можно обнаружить группы символов или даже целые слова, которые неоднократно встречаются в тексте. Поэтому с целью сокращения избыточности сообщения целесообразно вместо передачи повторно встречающихся групп символов делать *ссылку* на аналогичные группы, переданные в предыдущей части сообщения. Эта идея лежит в основе метода компрессии данных, предложенного израильскими учеными А.Лемпелем и Я.Зивом [46]. В связи с тем, что метод сжатия был опубликован в 1977 году, он получил название *LZ*-77. Метод *Лемпеля-Зива* (*LZ*-77) относится к однопроходным адаптивным методам сжатия последовательных данных, не требующим априорных знаний статистических характеристик источника сообщений. Алгоритм состоит из правила для синтаксического анализа строк сжимаемого сообщения, состоящего из символов конечного алфавита, и схемы кодирования. В процессе анализа из строк сообщения выделяются подстроки (*слова*) переменной длины. При кодировании подстрокам переменной длины ставятся в соответствие кодовые комбинации, состоящие из фиксированного числа элементов. В связи с этим способ сжатия относится к группе кодов, получивших название *variable-to-fixed length coding*.

При синтаксическом анализе из некоторой строки сообщения выделяются подстроки различной длины и затем осуществляется поиск идентичных подстрок в предшествующей строке. После отыскания совпадающих подстрок выбирается подстрока максимальной длины и производится ее замена в текущей строке сообщения соответствующей кодовой комбинацией. Эта комбинация включает координату позиции подстроки в предыдущей строке сообщения, с которой начиналась аналогичная подстрока, и длину этой подстроки. В процессе анализа совпадающей подстроки может быть не обнаружено. В таком случае в кодовой комбинации не содержится никакой информации. Поэтому в

качестве разделительного компонента между подстроками в кодовую комбинацию добавляется символ текущей части строки, следующей за совпадающей подстрокой.

Рассмотрим более подробно алгоритм компрессии методом Лемпеля-Зива. Сообщения, поступающие от источника, представляются в виде совокупности строк S. При синтаксическом анализе эти строки разбиваются на отдельные подстроки (слова) $S=S_1, S_2, \ldots, S_i$. Длина S_i -го слова является переменной, но не может превышать определенной величины L_S. В процессе кодирования каждому слову ставится в соответствие кодовая комбинация. Все операции со строкой сообщения производятся в буфере длиной n_B символов. Весь буфер разделяется на две части: *текущую часть строки* длиной $L_S \le n_B$ и *предыдущую*, размер которой равен $n_B - L_S$ (рис. 5.1).

Рис.5.1. Распределение позиций буфера кодирования

Формирование кодируемого слова начинается с символа, расположенного на первой позиции текущей части строки, т. е. на n_B-L_S+1 -й позиции от начала буфера. Затем осуществляется поиск аналогичного символа в предыдущей части строки на позициях с 1 и по n_B -L_S -ю. Если совпадение обнаружено, то размер подстроки увеличивается на один символ, и поиск ее копии в предыдущей части буфера продолжается. Таким образом осуществляется поиск подстроки максимальной длины, совпадающей с данной. При отыскании такой подстроки фиксируется расстояние p_i от начала регистра до места расположения найденной подстроки и количество символов l_i, входящих в эту подстроку. Значение указателя p_i может изменяться от 1 до n_B - L_S, а l_i - от 1 до L_S - 1. В состав кодируемого слова, кроме найденной подстроки добавляется следующий за ней символ. Поэтому общая длина слова L_i на один символ больше длины подстроки, т.е. $L_i = l_i+1$. Кодовая комбинация C_i, отображающая i-ю кодируемую подстроку, будет содержать следующие три компоненты *(триплет)*:

$$Ci = C_{i1}\, C_{i2}\, C_{i3}\ , \qquad (5.1)$$

где C_{i1} — значение p_i – 1, представленное в системе счисления с основанием m, равным размеру алфавита источника; C_{i2} — значение L_i – 1, представленное в системе счисления по основанию m; C_{i3} —символ, следующий за кодируемой подстрокой источника. Таким образом, сжатие сообщения будет иметь место, если длина кодируемой подстроки будет больше фиксированной длины кодового слова.

Длина кодового слова $L(C_i)$ равна сумме длин его компонент, выраженная количеством символов используемого алфавита источника:

$$L(C_i) = \lceil \log(n_в - L_S) \rceil + \lceil \log L_S \rceil + 1, \qquad (5.2)$$

где $\lceil x \rceil$ — наименьшее целое, не меньше x. Основание логарифма зависит от принятой системы счисления. В данном алгоритме оно равно размеру алфавита источника m.

Естественно предположить, что чем больше будет длина анализируемой части строки $n_B - L_S$, тем выше вероятность нахождения в ней подстроки максимальной длины, которая входит в состав текущей части строки длиной L_S.

На практике длину буфера следует выбирать [46] из соотношения:

$$n_B = L_S \; m^{h L S}, \qquad (5.3)$$

где $0 < h < 1$. Например, при $L_S = 8$, $h = 0{,}5$ и $m = 2$ длина буфера $n_B = 8 \times 2^4 = 128$ бит.

В начале процесса кодирования предполагается, что выходу источника предшествует строка Z, состоящая из $n_B - L_S$ нулей. Обозначим i-ю строку, находящуюся в буфере, через B_i. Тогда первая строка B_1, расположенная в буфере (рис.5.1), представляет собой *конкатенацию* (объединение) двух строк Z и S, то есть $B_1 = ZS(1,L_S)$. Здесь символом $S(i,j)$ обозначается подстрока, начинающаяся с позиции i и заканчивающаяся в позиции j.

На втором этапе кодирования формируются подстроки, состоящие из символов сообщения, расположенных начиная с $n_B - L_S + 1$ -й позиции буфера. В общем случае эта подстрока будет состоять из j символов $s(1), s(2),..., s(j),$ причем j может изменяться от 1 до $L_S - 1$, так как $(j + 1)$ — й символ, стоящий после строки, является обязательным компонентом кодового слова (5.1). Если в предыдущей части буфера в диапазоне позиций от 1 до $n_B - L_S$ найдена подстрока $S(1, j),$ то первым кодируемым словом источника, с учетом дополнительного символа C_{i3} , будет подстрока $S(1, j + 1),$ а его длина равна $L_1 = j + 1$. Подстроку $S(1,j)$ называют *восстанавливаемым расширением* строки $B(1,n_B-L_S)$ до строки

$B(1,\ n_B - L_S + j)$, так как она кодируется словом C_i, из которого при декодировании восстанавливается подстрока сообщения.

На третьем этапе определяются параметры p_1 и L_1, и на их основе формируется кодовая комбинация C_1. После завершения процедуры кодирования последовательность в буфере сдвигается влево на L_1 позиций, а освободившееся место заполняется следующими L_1 символами. При этом в буфере образуется строка

$$B_2 = B_1 (L_1+1,\ n_B)\ S(L_S+1,\ L_S + L_1) .$$

Далее процедура сжатия повторяется вновь со второго этапа.

Пример 5.1.

Закодировать *LZ*-способом последовательность символов *S*, состоящих из элементов троичного алфавита A (m=3; A= 0, 1, 2), если размер буфера n_B = 18 символов, а максимальная длина кодового слова L_S = 9. Последовательность на выходе источника имеет вид:

$$S = 00101021021021202102120 0... .$$

Определим по (5.2) длину кодовой комбинации.

$$L(C) = log_3 (18 - 9) + log_3 9 + 1 = 5 .$$

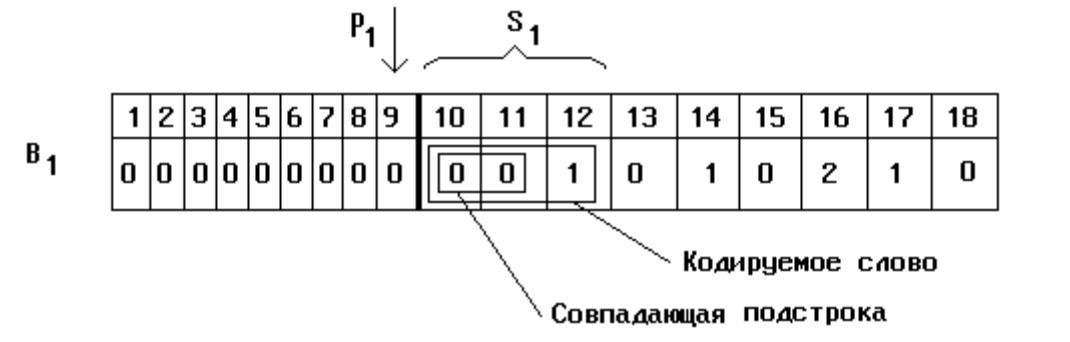
Загружаем буфер $n_B - L_S$ нулями и L_S = 9 первых символов строки *S*. Содержимое буфера принимает вид (рис.5.2,а). Для строки, начинающейся с позиции $n_B - L_S + 1 = 10$, ищем совпадающую подстроку максимальной длины. Подстроке $B_1(10,11)$ соответствует подстрока максимальной длины, начинающаяся с любой позиции в диапазоне от 1 до 9. Условимся указатель смещения p_1, определяющий начало этой подстроки, принять равным $n_B - L_S$ - й позиции (это условие действительно только для начального этапа кодирования). Таким образом, p_1=9.

Длина кодируемого слова источника превышает длину совпадающей подстроки на 1 и равна $L_1 = 3$. Первое кодовое слово начинается с 10-й позиции, занимает 3 символа и имеет вид $S_1 = 001$. В соответствии с (5.1) определим компоненты первой кодовой комбинации:

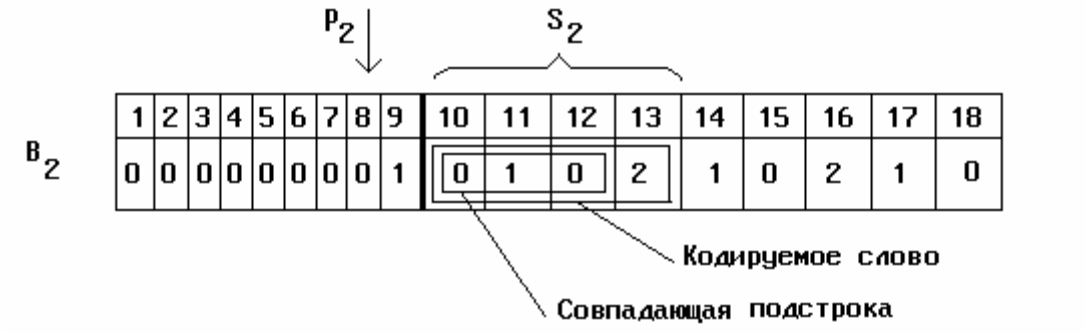
$$C_{11} = (p_1 - 1) = 8 = 22_3 ;$$

$$C_{12} = (L_1 - 1) = 2 = 02_3 ;$$

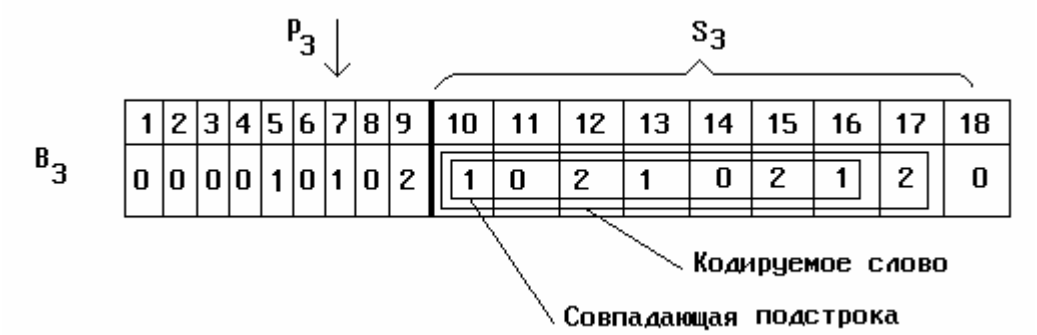
$$C_{13} = 1 .$$



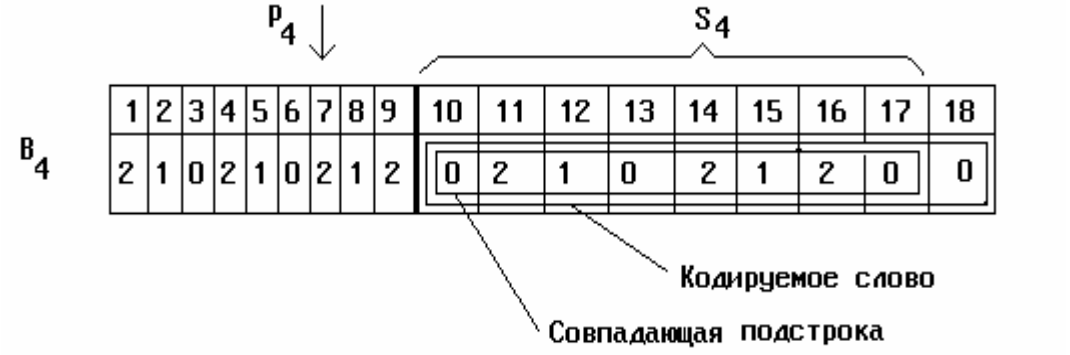
а)



б)



в)



г)

Рис.5.2. Состояние буфера при кодировании методом LZ

Кодовая комбинация C_1 = 22021 соответствует первому кодируемому слову источника. Сдвигая содержимое буфера B_1 на величину длины L_1 =3 первого слова источника S_1 и заполняя освободившиеся позиции символами из входной последовательности, получаем строку B_2 (рис.5.2,б). Повторяя процедуру поиска, находим подстроку максимальной длительности 010, длина которой равна 3. Кодовое слово и указатели соответственно равны:

$$S_2 = 0102; \quad L_2 = 4; \quad p_2 = 8.$$

Определим компоненты второй кодовой комбинации:

$$C_{21} = (p_1 - 1) = 7 = 21_3;$$

$$C_{22} = (L_2 - 1) = 3 = 10_3;$$

$$C_{23} = 2.$$

Второму кодируемому слову источника соответствует кодовая комбинация C_2 = 21102. Для третьего (S_3 =10210212) и четвертого (S_4 = 021021200) кодируемых слов содержимое буферов B_2 и B_3 показано на рис.5.2,в и г. Отображающие их кодовые комбинации соответственно равны:

$$C_3 = 20212; \quad C_4 = 02220.$$

Таким образом, полученная сжатая последовательность имеет вид:

$$C = 220212110220212022 0,$$

а коэффициент сжатия последовательности $K_{сж} = 20 / 24 = 0,83$.

Процесс декодирования обратен процедуре кодирования. На первом шаге буфер заполняется $n_B - L_S$ нулями, а в оставшиеся L_S позиций загружается принимаемая сжатая последовательность. Кодовые комбинации декодируются следующим образом. Из первого компонента кодовой комбинации C_{i1} формируется указатель p_i , по которому из указанных ячеек буфера выбираются символы, число которых определяет второй компонент C_{i2} и к этим символам добавляется символ из C_{i3}.

Из приведенного описания видно, что *LZ*77 является очень простым адаптивным способом сжатия, не требующим знания статистических характеристик источника сообщений. Благодаря достаточно высокой эффективности сжатия сообщений *LZ*77 явился основой таких популярных архиваторов данных как *LHArc, ARJ, Zip* и *PKZIP*.

Степень сжатия сообщений рассмотренным способом в значительной мере ограничивается длиной буфера n_B . Существует ряд

способов, с помощью которых процедура *LZ* может быть сделана более эффективной. Многие улучшения *LZ* алгоритма связаны с повышением эффективности кодирования триплетов (5.1) в связи с тем, что использование триплета неэффективно, особенно, если длинные строки в сообщении встречаются редко. Избавление от этой неэффективности заключается в простом добавлении флагового бита, показывающего, следует ли кодовое слово за простым символом. Используя этот флаг, избавляются также от необходимости в трех элементах триплета. Теперь все, что необходимо сделать- отправить пару значений, соответствующих смещению и длине совпадения. Эта модификация алгоритма *LZ*77 используется в архиваторе *LZSS*.

При разработке алгоритма *LZ*77 неявно предполагалось, что в сжимаемом сообщении одинаковые подстроки будут располагаться довольно близко друг к другу. Это приводит к использованию таких подстрок в предыдущей части строки в качестве словаря для кодирования последующих частей. Однако, любая подстрока, которая повторяется через период, длиннее чем строка, охваченная регистром кодера, не будет захвачена и вместо сжатия может произойти расширение сообщения. В современных системах сжатия величину n_B повышают до 64 Кбайт. Но рост размера буфера приводит к существенному увеличению времени поиска , для уменьшения которого требуется применение более эффективных стратегий поиска.

В результате решения этой проблемы Лемпелем и Зивом в 1978 году была опубликована статья с описанием улучшенного алгоритма сжатия строк переменной длины [47], который получил название *LZ*78. В соответствии с предложенным ими алгоритмом составляется таблица *(словарь)* подстрок символов, которые встречались в предыдущей части сообщения. Тем самым устраняются ограничения на расстояние между текущей комбинацией символов и комбинацией, имевшей место в переданной части сообщения. Выделяемые из поступающего входного сообщения подстроки кодируются двумя кодовыми комбинациями - *дуплетом (i,C),* где *i*-индекс, соответствующий самой длинной подстроке в словаре, совпадающей с выделенной подстрокой, и C - код символа входного потока, следующего за совпадающей подстрокой. Этот дуплет затем становится новой записью в словаре, которой присваивается очередной индекс таблицы подстрок.

Предложенный способ самостоятельного применения практически не нашел, а явился основой широко распространенного алгоритма компрессии *LZW* и метода сжатия, применяемого в модемах с протоколом *V*.42 *bis,* которые рассматриваются ниже.

5.2. Динамическое кодирование методом LZW

Этот метод компрессии данных предложен Т.Уелчем в 1984 г. [45]. Он получил название *"метод LZW",* так как является дальнейшим развитием метода *LZ*88. При кодировании *LZW*-методом используется таблица строк, состоящая как из одиночных символов, так и из некоторых буквосочетаний. Причем каждой строке соответствует своя двоичная комбинация *(кодовое слово).* Примером сокращенной таблицы, содержащей строки, составленные всего из трех символов a, b и c, является табл.5.1,а. В ней кодовые слова для экономии записи представлены десятичными эквивалентами.

Таблица 5.1

Строка	Кодовое слово
a	1
b	2
c	3
ab	4
ba	5
abc	6
cb	7
bab	8
baba	9
aa	10
aaa	11
aaaa	12

а)

Строка	Кодовое слово
a	1
b	2
c	3
1b	4
2a	5
4c	6
3b	7
5b	8
8a	9
1a	10
10a	11
11a	12

б)

Табл.5.2,б является другой формой представления табл.5.1,а, в которой каждая последующая строка состоит из кодового слова (в десятичной форме) одной из предыдущих строк и одиночного символа.

Обратите внимание, что таблица строк обладает префиксным свойством, т.е. более длинная строка состоит из *префикса* - комбинации некоторого числа символов *p* и одиночного символа *C*, причем префикс является одной из строк таблицы. Символ *C* называют *расширением* префиксной строки *p*.

Процедура динамического кодирования *LZW*-методом состоит в следующем. При начальном заполнении *LZW*-таблицы в нее вносятся

строки, которые отображают статистику сообщений на определенном языке. При отсутствии такой статистики таблица содержит только строки, состоящие из одиночных символов. Затем, по мере поступления данных от источника информации, формируются строки, состоящие из нескольких символов. Так как таблица имеет ограниченный размер, то строки, встречающиеся в тексте редко, исключаются, а на их место вносятся строки, имеющие большую частоту появления. Таким образом в процессе накопления статистики о сжимаемом сообщении происходит динамическая перестройка таблицы кодирования и адаптация ее к характеру передаваемых данных.

Компрессия данных начинается с инициализации таблицы, при которой в нее вносятся строки, состоящие из одиночных символов. Затем поступает первый входной символ, рассматриваемый как префикс некоторой строки PREFIX. После этого вводится следующий символ CHARACTER и образуется расширенная строка путем объединения префикса и одиночного символа PREFIX + CHARACTER. Далее осуществляется сопоставление вновь образованной строки со строками, существующими в таблице кодирования. Если строка PREFIX + CHARACTER имеется в таблице, то она становится новым префиксом, т.е. PREFIX = PREFIX + CHARACTER и вводится следующий символ CHARACTER и процедура сопоставления строки в таблице повторяется снова. В противном случае, если последовательности PREFIX + CHARACTER в таблице строк нет, то на выход кодера выводится кодовая комбинация, соответствующая строке PREFIX, в таблицу вносится дополнительная строка PREFIX + CHARACTER, а символ CHARACTER становится новым префиксом. Если входная последовательность не исчерпана, то процедура образования строк и их сопоставления повторяется. Алгоритм сжатия данных имеет следующий вид:

Алгоритм 5.1

```
            ROUTINE LZW_COMPRESS
Initiale table to contain single-character strings
        INPUT  CHARACTER
        PREFIX = CHARACTER
        WHILE exists iput character  DO:
Next:   INPUT NEXT_CHARACTER
        CHARACTER = NEXT_CHARACTER
Step:   IF PREFIX + CHARACTER exists in the table THEN:
        PREFIX = PREFIX + CHARACTER
```

```
REPEAT Step
ELSE
OUTPUT the code for PREFIX
ADD PREFIX + CHARACTER  to the strig table
PREFIX = CHARACTER
REPEAT Next
END.
```

Пример 5.2.

Последовательность, поступающая с источника информации, имеет вид: tentomentoentetento. Предполагается, что начальная таблица кодирования содержит 256 одиночных символов с номерами кодовых слов от 0 до 255, а номера расширенных строк начинается с 256-го. Требуется произвести сжатие заданной последовательности методом *LZW* и дополнить таблицу кодовыми словами, соответствующими образованным строкам.

В соответствии с приведенным выше алгоритмом первый поступивший символ *t* обозначим переменной PREFIX, а следующий символ - переменной CHARACTER и объединим эти два символа в строку *te*. Так как строка *te* в кодовой таблице отсутствует, то префикс *t* выдается на выход, а первая строка *te* вносится в кодовую таблицу и ей присваивается очередной номер 256. Затем оставшийся символ е становится новым префиксом, а вновь поступивший символ *n* обозначается переменной CHARACTER. Так как строка *en* отсутствует в кодовой таблице, то она вносится туда вместе с соответствующей ей комбинацией 257, на выход кодера поступает символ *e*, а символ *n* становится новым префиксом. Полностью процесс компрессии всей последовательности представлен в табл.5.2.

Однозначность декомпрессии сжатых сообщений достигается тем, что в процессе *LZW*-декодирования создается такая же таблица строк, что и при кодировании. Полученная кодовая комбинация разделяется с помощью таблицы строк на префиксную строку PREFIX и одиночный символ CHARACTER. Символ расширения CHARACTER выдается на выход, а префиксная строка PREFIX вновь разделяется на префикс и одиночный символ. Эта операция рекурсивна и продолжается до тех пор, пока префиксная строка не выродится в одиночный символ, который и завершает процедуру декодирования. Символ, выводимый последним, является крайним левым элементом строки, то есть первым символом, с которым оперировал компрессор при синтаксическом анализе поступающих данных.

Таблица 5.2

Состояние кодера					Таблица строк	
Вход кодера	PREFIX	CHARACTER	PREFIX+ CHARACTER	Выход кодера	Строка	Код
t	t					
e	t	e	te	t	te	256
n	e	n	en	e	en	257
t	n	t	nt	n	nt	258
o	t	o	to	t	to	259
m	o	m	om	o	om	260
e	m	e	me	m	me	261
n	e	n	en		en	
t	en	t	ent	257	ent	262
o	t	o	to		to	
e	to	e	toe	259	toe	263
n	e	n	en		en	
t	en	t	ent		ent	
e	ent	e	ente	262	ente	264
t	e	t	et	e	et	265
e	t	e	te		te	
n	te	n	ten	256	ten	266
t	n	t	nt		nt	
o	nt	o	nto	258	nto	267

Изменения в таблице строк осуществляются для каждой полученной кодовой комбинации за исключением единственного первого символа. В процессе преобразования кода последний символ, используемый в качестве расширения, объединяется с предыдущей строкой, образуя тем самым новую строку. Этой строке ставится в соответствии очередная кодовая комбинация, которая заносится в таблицу строк декодера. Очевидно, что внесенная комбинация совпадает с аналогичной строкой таблицы, образованной компрессором. Таким образом, декомпрессор путем наращивания создает такую же самую таблицу строк, которая используется при сжатии.

Процедура декодирования сводится к следующему. Первая поступившая кодовая комбинация, обозначаемая переменной OLD_Code, всегда является кодом одиночного символа, и она без изменения выдается

на выход декодера. Затем вводится новое кодовое слово New_Code, которое для дальнейшего использования при формировании новой строки сохраняется в виде переменной NEX_Code. Если NEW_Code отображает расширенную строку PREFIX+ CHARACTER, то выделяется символ расширения строки CHARACTER и выдается на выход, а префикс вновь подвергается синтаксическому анализу. Операция повторяется до тех пор, пока префикс не будет представлен одиночным символом CHARACTER. После этого в таблице строк декодера образуется новая строка путем объединения первой комбинации OLD_Code и идентифицированным последним символом, а новое кодовое слово NEXT_Code становится для следующего цикла старым кодом OLD_Code. Затем вводится очередная комбинация и процедура повторяется снова. Алгоритм декомпрессии может быть представлен в следующем виде:

Алгоритм 5.2

```
    ROUTINE LZW_DECOMPRESS
READ  OLD_Code
OUTPUT  OLD_Code
 WHILE  exists input character  DO:
  READ NEW_Code
  NEXT_Code = NEW_Code
   IF NEW_Code = PREFIX + CHARACTER
     OUTPUT CHARACTER
     NEW_Code = PREFIX
   END of IF
   ELSE  IF NEW_Code = CHARACTER
   OUTPUT CHACTER
  ADD OLD_CODE + CHARACTER to string table
  OLD_Code =  NEXT_Code
 END of WHILE.
```

Пример 5.3.

На вход декодера поступает последовательность *"tentom* 257 259 262 *e* 258 *o*", которая была сформирована в предыдущем примере. Необходимо произвести декомпрессию последовательности и построить таблицу декодера в соответствии с изложенным алгоритмом.

Процедура декомпрессии иллюстрируется табл.5.3. Построенная таблица состоит из двух частей. Колонки первой части отображают

состояние переменных в процессе декомпрессии данных, а вторая часть представляет собой таблицу декодера, содержащую кодовые слова вновь образованных строк, совпадающих со строками таблицы кодера.

Таблица 5.3

Состояние декодера					Таблица строк	
Вход декодера	NEXT_ Code	NEW_ Code	OLD_ Code	Выход декодера	Строка	Код
t			t	t		
e	e	e	t	e	te	256
n	n	n	e	n	en	257
t	t	t	n	t	nt	258
o	o	o	t	o	to	259
m	m	m	o	m	om	260
257	en	en	m	n		
	en	e	m	e	me	261
259	to	to	en	o		
	to	t	en	t	ent	262
262	ent	ent	to	t		
	ent	en	to	n		
	ent	e	to	e	toe	253
e	e	e	ent	e	ente	264
256	te	te	e	e		
	te	t	e	t	et	265
258	nt	nt	te	t		
	nt	n	te	n	ten	266
o	o	o	nt	o	nto	267

Отличительной особенностью рассмотренного *LZW*-алгоритма является простота реализации и относительно высокая скорость декомпрессии. Однако ему присущи два недостатка. Первый заключается в том, что декомпрессор восстанавливает символы обрабатываемой строки в обратном порядке, то есть первым выделяется последний символ строки.

Устранить этот недостаток просто, если в процессе декодирования строки выходные символы перед выдачей их потребителю помещать в стек, а по завершении декодирования символы считывать из стека в нужной последовательности (поступивший последним, выводится

первым), т.е. в той, в которой они поступали от источника. Второй недостаток - аварийный сбой процедуры декомпрессии при наличии во входной символьной строке повторяющихся групп символов вида *CsCsCs* (здесь *C*- одиночный символ, *s*-некоторая произвольная строка), при условии, что строка *Cs* уже содержится в таблице компрессора.

В этом случае кодер, анализируя строку, выдает из таблицы код *(Cs)* и вводит в таблицу следующую строку *CsC*. Затем, обнаружив во входной последовательности строку *CsC*, кодер выводит только что образованную кодовую комбинацию *(CsC)*. Декомпрессор не может декодировать эту комбинацию, так как ее еще нет в таблице строк декомпрессора. Очевидно, что она не может быть сформирована на основе предыдущей строки *Cs*, поскольку декомпрессор не знает символа расширения (в данном случае *C*) до приема следующей строки.

Пример 5.4.

Пусть на вход компрессора в некоторый момент времени поступает последовательность ABCABCABCABCABCABC... . Во время сжатия сообщения могут поступить все возможные символы ASCII - кода, в связи с чем до начала процедуры компрессии таблицы кодера и декодера заполнены одиночными символами с кодовыми комбинациями от 0 до 255 включительно. Требуется построить таблицы кодирования и декодирования в соответствии с алгоритмом *LZW*.

Процедура кодирования объясняется табл.5.4. На выход кодера выдается кодовая последовательность *ABC* 256 258 257 259 262... . Таблица декодирования, построенная в соответствии с изложенным выше алгоритмом, представлена табл.5.5. Очевидно, что декомпрессор не в состоянии декодировать группу символов *ABCA* (код 262), так как в его таблице такая комбинация отсутствует. В этом случае декодер может использовать предыдущую комбинацию, то есть он выбирает комбинацию OLD_Code вместо NEW_Code.

Алгоритм 5.3

```
  ROUTINE MOD_LZW_Decompress
READ  FIRST_Code
OLD_Code:=FIRST_Code
FINchar:=OLD_Code
OUTPUT  OLD_Code
  WHILE  exist input character  DO:
    READ  NEW_Code
    NEXT_Code:=NEW_Code
```

```
   IF  NEW_Code not defined (Special Case):
      OUTPUT  FINchar
      NEW_Code:=OLD_Code
      NEXT_Code:=OLD_Code + FINchar
   END of IF
   IF  NEW_Code = PREFIX + CHARACTER
      PUSH  CHARACTER in Stack
      NEW_Code:=PREFIX
   END  of  IF
   ELSE IF  NEW_Code = CHARACTER
      OUTPUT  CHARACTER
      FINchar:=CHARACTER
      DO  while Stack not empty
      POP  Stack
   ADD  OLD_Code+CHARACTER to string table
   OLD_Code:=NEXT_Code
END  of  WHILE.
```

Таблица 5.4

Входная последовательность: ABCABCABCABCABCABC...		
Строка	Кодовое слово	Выход
:		
A		
B		
C		
:		
AB	256	A
BC	257	B
CA	258	C
ABC	259	256 = AB
CAB	260	258 = CA
BCA	261	257 = BC
ABCA	262	259 = ABC
ABCAB	263	262 = ABCA

Таблица 5.5

FIRST_ Code (Вход декодера)	NEXT_ Code	NEW_ Code	OLD_ Code	CHARACTER (Выход декодера)	Таблица строк	
					Строка (OLD_Code+ CHARACTER)	Код
A			A	A		
B	B	B	A	B	AB	256
C	C	C	B	C	BC	257
256	AB	AB	C	B		
	AB	A	C	A	CA	258
258	CA	CA	AB	A		
	CA	C	AB	C	ABC	259
257	BC	BC	CA	C		
	BC	B	CA	B	CAB	260
259	ABC	ABC	BC	C		
	ABC	AB	BC	B		
	ABC	A	BC	A	BCA	261
262=ABCA	?	?	ABC	?		

Выходная комбинация формируется путем прибавления к последней комбинации OLD_Code (*ABC*) последнего, выданного на выход символа. Для нашего примера: *ABC + A = ABC*. Модифицированный алгоритм, свободный от указанных недостатков, имеет следующий вид:

Метод компрессии *LZW* запатентован автором и находится в пользовании фирмы *UNISYS*, что ограничило его массовое применение. После некоторых изменений этот метод был использован в качестве основы протокола сжатия информации *V*.42*bis*, используемого в аппаратуре передачи данных (модемах).

5.3. Особенности программной реализации кодирования строк переменой длины

Алгоритмы сжатия данных методом *LZW* довольно просты. Однако программная реализация алгоритмов компрессии характеризуется двумя особенностями, которые приводят к большому расходу памяти вычислителя и заметному снижению быстродействия процесса

кодирования поступающих сообщений. Первая особенность состоит в том, что количество возможных комбинаций символов, объединяемых в строку, может превышать сотни тысяч. А каждая комбинация должна быть внесена в соответствующую строку кодовой таблицы, которая содержит код номера строки (*индекс*) и кодовые комбинации соответствующих символов. Таким образом, кодовая таблица может быстро исчерпать всю доступную память, объем которой ограничен. Вторая особенность касается процедуры доступа к кодовой таблице при поиске в ней очередной сформированной строки PREFIX + CHARACTER. Поиск каждой строки в таблице размером M_C строк требует $log_2\ M_C$ сравнений строки с комбинациями символов из таблицы. Так при $M_C = 4096$ строк операция сравнения производится 12 раз, что существенно увеличивает время выполнения процедуры сжатия.

Объем занимаемой памяти может быть в значительной степени снижен при хранении строк в виде объединения кода префикса и кода символа расширения. При такой записи кодовая таблица 5.2 будет иметь вид табл.5.6.

Таблица 5.6

Состояние переменных					Кодовая таблица		
Вход кодера	PREFIX	CHARACTER	PREFIX+ CHARACTER	Выход кодера	Code_ value	Prefix_ Code	append_ character
t	t						
e	t	e	te	t	256	код t	код e
n	e	n	en	e	257	e	n
t	n	t	nt	n	258	n	t
o	t	o	to	t	259	t	o
m	o	m	om	o	260	o	m
e	m	e	me	m	261	m	e
n	e	n	en				
t	en	t	ent	257	262	257	t
o	t	o	to				
e	to	e	toe	259	263	259	e
n	e	n	en				
t	en	t	ent				
e	ent	e	ente	262	264	262	e
t	e	t	et	e	265	e	t
e	t	e	te				
n	te	n	ten	256	266	256	n
t	n	t	nt				
o	nt	o	nto	258	267	258	o

Для хранения кодовой таблицы можно использовать массив, элементами которого является запись, содержащая три поля: код последовательности PREFIX + CHARACTER (переменная *code_value*), код префикса *(prefix_code)* и код символа расширения *(append_character)*. Так например, строка *ente* с кодом 264 в Табл.5.2 при длине кодовой комбинации 12 бит занимает в памяти 6 байт, а при записи в Табл.5.6 в виде 262*e* — 3 байта. Очевидно, что с увеличением длины строк эффективность такого способа хранения повышается.

Уменьшение времени поиска строки в кодовой таблице может быть достигнуто применением метода непосредственного доступа к таблице, который получил название *"хеширование"*. Идея хеширования состоит в том, чтобы взять некоторые характеристики ключа поиска *Key* и использовать полученную частичную информацию в качестве основы поиска. Над ключом производятся некоторые математические операции и затем вычисленная хеш-функция *h(Key),* используемая в качестве адреса начала поиска. Находить вид подходящей хеш-функции довольно сложно, так как для большого числа простых преобразований ключей имеет место явление, при котором для различных ключей получается одна и та же хеш-функция, то есть $h(Key_i) = h(Key_j)$ при $Key_i \neq Key_j$. Такое явление получило название *"коллизия"*. Для разрешения коллизий разработаны различные методы, о которых будет сказано ниже.

Хорошая хеш-функция должна удовлетворять двум требованиям:

а) ее вычисление должно быть очень быстрым;

б) она должна вызывать минимальное число коллизий.

Свойство *а)* в большой степени зависит от характеристик вычислителя и вида хеш-функции, а свойство *б)*-от характеристик данных. Если бы ключи были случайными величинами, то можно было бы просто выделить группу битов и использовать их для хеш-функции. Но на практике, чтобы удовлетворить *б)* почти всегда нужна функция, зависящая от всех битов. Следует заметить, что существует большое количество различных методов хеширования, но ни один из них не оказался предпочтительнее простых схем деления и умножения.

При методе деления используется остаток от деления на число *M:*

$$h(Key) = Key \bmod M . \qquad (5.4)$$

Здесь важную роль играет выбор числа *M*. Рекомендации по выбору числа *M* приведены в [11]. Особо эффективной оказывается процедура деления, основанная на алгебраической теории кодирования [21,22]. В качестве делителя в этом случае вместо целого числа *M* используется *неприводимый* многочлен *P(x)* вида:

$$P(x) = p_m x^m + p_{m-1} x^{m-1} + \ldots + p_1 x + p_o \ .$$

Неприводимым является многочлен, который делится только на самого себя и на 1. Ключ поиска, представленный в двоичной форме $Key = k_{n-1}\ldots k_1 k_o$, рассматривается как алгебраический многочлен:

$$Key(x) = k_{n-1} x^{n-1} + \ldots + k_1 x + k_o \ .$$

Хеш-функция представляет собой остаток от деления $Key(x)$ на полином $P(x)$, т.е.:

$$h(Key) = Key(x) \bmod P(x) = h_{m-1} x^{m-1} + \ldots + h_1 x + h_o \ . \quad (5.5)$$

Процедура деления использует полиномиальную арифметику по модулю 2. Она похожа на обычное деление, однако вместо операции вычитания используется сложение по модулю 2. При правильном выборе $P(x)$ такая хеш-функция позволяет избежать коллизий между почти равными ключами.

При сжатии строк переменной длины достаточно быстрой является процедура формирования ключей поиска на основе всех входящих в строку символов, причем символы можно сначала объединить в одно слово, а затем производить деление или умножение. Для комбинирования символов можно использовать суммирования по модулю 2. Достоинством такой операции является зависимость ее результата от всех битов аргументов. Кроме этого сложение по модулю 2 не приводит к арифметическому переполнению. Однко следует помнить, что эта операция коммутативна, поэтому и ключи (x,y) и (y,x) будут задавать один и тот же адрес. Для исключения такой операции нужно предварительно делать циклический сдвиг.

Очень эффективной в системах сжатия методом *LZW* может оказаться функция хеширования, разработанная специально для строк переменной длины и применяемая в криптографии. В качестве входных данных она использует N символов C_1, C_2, ... , C_N. Каждый символ представлен одним байтом, индекс которого изменяется в диапазоне 0 - 255. Для вычисления хеш-функции используется таблица Tx, состоящая из 256 псевдослучайных байтов. В процессе вычислений выполняется рекурентная процедура сложения по модулю 2 хеш-функции предыдущего индекса и текущего символа строки. В результате на основе n символов строки формируется адрес этой строки в кодовой таблице $h(N)$. Алгоритм вычисления такой хеш-функции имеет вид:

```
h[0] := 0;
for i = 1 to N
h[ i ] := Tx[ h[i-1] xor C[ i ] ];
return h[ N ].
```

Из приведенного алгоритма очевидно, что если последний входной символ *C[N]* является случайным, то все конечные значения равновероятны. Две входные строки одной длины, различающиеся лишь одним символом, не могут выдавать одинаковые значения хеш-функции. При использовании этого алгоритма в программах сжатия следует учитывать, что размер символа превышает один байт.

К сожалению ни одна хеш-функция не может гарантировать отсутствие коллизий. Для их разрешения разработан ряд методов. Самым простым и достаточно эффективным методом разрешения коллизий при хешировании является *смещение* данной записи в следующую свободную позицию, т. е. осуществляется повторное хеширование с использованием циклической последовательности:

$$h(Key),\ h(Key)+1,...,\ M_T-2,\ M_T-1,\ 0,\ 1,...,\ h(Key)-1\ .$$

Здесь M_T - размер кодовой таблицы. Этот метод получил название *"линейное опробование"* или открытой адресации. Алгорим, реализующий метод открытой адресации, позволяет разыскать данный ключ *k1* в таблице из M_T строк. Если *k1* в таблице отсутствует и она неполная, то *k1* вставляется в строку.

Обозначим строки таблицы кодирования TABLE [i], $0 \le i \le M_T$. Они могут быть сбодны и заняты. Занятая содержит КЕУ[i] и другие поля. Введем вспомогательную переменную *Nt*, содержащую число занятых строк таблицы. Эта переменная рассматривается как часть таблицы, и при вставке нового ключа ее значение увеличивается на 1. Алгоритм хеширования с разрешением коллизий имеет вид:

Алгоритм 5.4

```
    ROUTINE   HASH
1.  i := h(Key), 0 <= i < Мт ;          { Хеширование }
2.  IF TABLE [ i ] свободна GO TO 8
3.  IF Key[ i ] = Key  GO TO 10;        { Если ключ найден,}
                                        { закончить поиск }
4.  i := i+1, Nt := Nt + 1;             {Если нет переполнения, то}
5.  IF Nt <= Мт - 1 GO TO 2 ;           {повторное хеширование}
7.  ELSE   GO TO   9 ;
```

```
 8. Insert Key, Nt := Nt + 1;          {Вставить ключ }
 9. IF Nt > Мт - 1  HALT ;             {Останов по переполнению}
10. END  .                             {Конец поиска }
```

При смещении индекса в процессе повторного хеширования может оказаться, что два ключа, которые хешируются в разные позиции, конкурируют друг с другом при повторном хешировании. Это явление называется *скручиванием*. Одним из способов исключения скручивания является двойное хеширование с открытой адресацией, которое использует две хеш-функции *h1(Key)* и *h2(Key)*. Алгоритм такого хеширования имеет вид:

Алгоритм 5.5

```
      ROUTINE  HASH_double
 1. i := h1(Key), 0 <= i < M ;       {Первичное хеширование}
 2. IF TABLE [ i ]  свободна  GO TO 8;
 3. IF Key[ i ] = Key  GO TO 10;  {Поиск удачен, закончить}
 4. Nt := Nt + 1;
 5. IF  Nt > Мт -1  GO TO 11;     {Останов по переполнению}
 6. c := h2(Key) ;                  {Вторичное хеширование}
 7. i := i + c, GO TO 2
 8. Insert Key, Nt := Nt + 1;       {Вставить ключ}
 9. IF Nt > Мт -1 GO TO 11;         {Если переполнение, останов}
10.  END ;                          {Конец поиска}
11.  HALT.
```

Для выбора функции вторичного хеширования предложено ряд способов [11]. В качестве достаточно эффективных можно рекомендовать следующие соотношения:

$$h2(Key) = 1, \quad \text{Если } h1(Key) = 0;$$
$$h2(Key) = Мт - h1(Key), \quad \text{Если } h1(Key) > 0.$$

Либо:
$$h1(Key) = Key \bmod Мт;$$
$$h2(Key) = 1 + [Key \bmod (Мт - 1)].$$

Для уменьшения затрат времени на поиск в таблице целесообразно вначале “пометить” ее свободные строки путем записи в поле хранения кодовых комбинаций символов *code_value* некоторого числа, которое не может совпадать ни с одним допустимым значением. Таким числом является, например минус 1, так как отрицательной кодовая комбинация не

может быть. Вычислив на основе очередной поступившей от источника группы символов индекс записи, можно быстро определить, содержится ли искомая строка в таблице. Для этого достаточно проанализировать только указанное поле записи. Если в поле *code_value* находится -1, то значит запись с таким индексом свободна и эта группа символов еще не занесена в таблицу. Этим самим избавимся от необходимости длительного поэлементного сравнения кодов каждого символа строки.

При программной реализации процедуры компрессии методом *LZW* таблицу кодирования целесообразно представить в форме массива записей, включающих три поля: код строки символов PREFIX + CHARACTER с именем *code_value*, код префикса *(prefix_code)* и кода символа расширения *(append_character)*. В табл. 5.6 приведен пример компрессии строки *tentomentoentetento* с учетом нового способа хранения строк и использования операции хеширования. Для лучшего понимания процедуры компрессии её можно представить в виде схемы, приведенной на рис.5.3.

code_value := counter
prefix_code := PREFIX
append_character := CHARACTER.

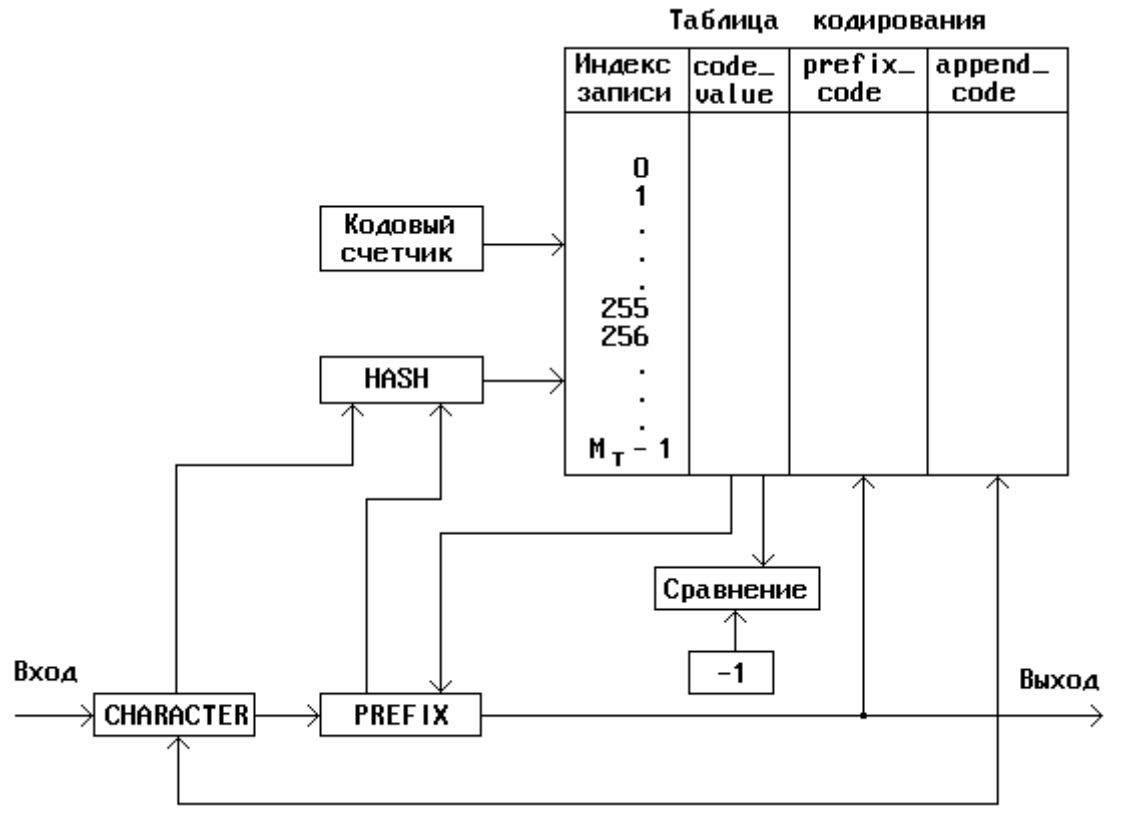


Рис. 5.3. Схема сжатия сообщений методом LZW

В исходном состоянии после инициализации таблицы поля *code_value* все элементы массива записей содержат минус 1, а кодовый счетчик - число 256. Первый поступивший на вход символ инициализирует PREFIX, а второй - CHARACTER. Из комбинации символов PREFIX + CHARACTER с помощью хеширования формируется индекс строки кодовой таблицы. Если поле *code_value* записи с номером *index* содержит значение -1, то эта позиция таблицы свободна и комбинацию PREFIX + CHARACTER нужно сохранить в таблице. Поля записи с номером *index* принимают следующие значения:

Затем состояние кодового счетчика увеличивается на 1. Значение PREFIX выдается на выход и после этого PREFIX принимает состояние CHARACTER, а CHARCTER - значение очередного символа источника. Из PREFIX + CHARACTER формируется следующий индекс, и если поле *code_value* записи с индексом *index* имеет значение, отличное от -1, то это значит, что такая комбинация символов уже есть в таблице. В этом случае PREFIX принимает состояние поля *code_value* записи с номером *index*, т.е. код группы символов PREFIX+ CHARACTER, уже размещенной в таблице. Далее вводится следующий символ сообщения и цикл повторяется вновь.

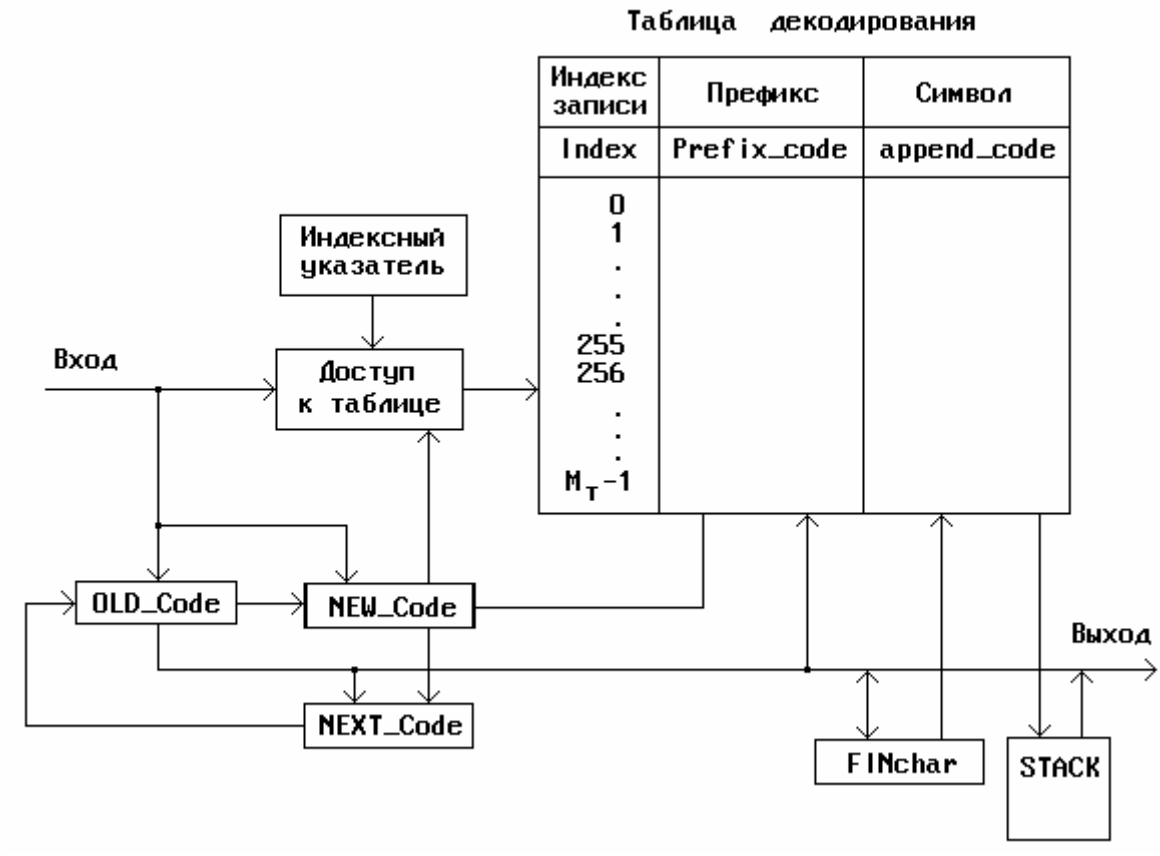


Рис.5.4. Схема декомпрессии сообщений методом LZW

Процедура декомпрессии по сравнению с компрессией является более простой, так как отпадает потребность в хешировании. Это связано с тем, что поступающий код строки символов одновременно является индексом таблицы декодирования. Таким образом, после получения кодовой комбинации соответствующей строке PREFIX+ CHARACTER, декодер заносит в стек символ расширения CHARACTER, а выделенный префикс является новым индексом таблицы.

Процесс декомпрессии повторяется до тех пор, пока PREFIX не превратится в одиночный символ. Таблицу декодирования также целесообразно представить в форме массива записей с полями *code_value, prefix_code* и *append_ character*, значения которых соответствуют аналогичным полям таблицы кодирования.

Схема процесса декомпрессии, с учетом возможности появления на входе декодера неопределенных комбинаций, показана на рис.5.4. Она полностью соответствует алгоритму MOD_LZW_ Decompress и не нуждается в дополнительных пояснениях.

В приложении П3 и П4 приведены программы компрессии и декомпрессии текстовых сообщений, написанных с учетом изложенных особенностей и реализованных на языке Pascal.

5.4. Сжатие информации по рекомендации V.42*bis*

Рекомендация Международного консультативного комитета по телефонии и телеграфии МККТТ*(CCITT* в настоящее время *ITU-T*) *V*.42*bis* регламентирует процедуру сжатия данных для использования в аппаратуре передачи данных (АПД) серии *V*. При разработке этой рекомендации были исследованы алгоритмы сжатия, используемые в протоколах *MNP*-5 и -7, протокол сжатия *CommPressor* фирмы Adaptive Computer Technologies и модифицированный фирмой British Telecommunication алгоритм Лемпеля-Зива (*BTLZ*). Для рекомендации V.42*bis* выбран алгоритм BTLZ как более эффективный по использованию памяти и ресурсов процессора.

Отличием *BTLZ* является наличие словаря варьируемого объема, в котором хранятся кодовые комбинации для передачи повторяющихся строк символов. Строки словаря динамически формируются на передающей и приемной сторонах. Степень сжатия тем выше, чем больший объем памяти занят словарем. В других упомянутых алгоритмах объем словаря фиксирован. Эффективность сжатия во всех исследованных протоколах примерно равна. В то же время при объеме словаря 2048 слов *MNP-7* требует объема ОЗУ на 30% больше, чем *BTLZ* и на 50% больше,

чем *CommPressor*. Однако время обработки в *CommPressor* при меньшем объеме ОЗУ в 8 раз больше, чем в *BTLZ* и примерно в 3 раза выше, чем в MNP-7.

Другим преимуществом *BTLZ* является его способность распознавать наличие последовательностей данных, по характеру близких к случайному. При обнаружении этого явления процедура сжатия выключается и ведется передача данных в несжатом виде. При этом контроль поступающей последовательности продолжается и после пропадания случайного характера последовательности данных они снова подвергаются сжатию. Другие протоколы в такой ситуации снижают пропускную способность до величины, меньшей технической скорости передачи модема.

Так как при передаче сообщений по каналам связи возникают ошибки, которые могут вызвать сбой при декомпрессии данных, то сжатие сообщений в аппаратуре передачи данных осуществляется только совместно с процедурой коррекции ошибок, регламентированной рекомендацией МККТТ *V*.42 или *V*.120. Сжатие по рекомендации *V*.42*bis* выполняется в двух режимах: *сжатия* и *"прозрачном"*. Переход из одного режима в другой выполняется автоматически на основании анализа содержания характера получаемых от источника данных. В "прозрачном" режиме данные передаются в несжатом виде, а в поток данных могут включаться управляющие кодовые комбинации. Основу процедуры сжатия данных составляет алгоритм *LZW*, при котором двоичное кодовое слово фиксированной длины отображает некоторую подстроку (совокупность) символов, поступающих от оконечного оборудования данных (ООД). Для компрессии и декомпресии данных используются соответствующие *словари* (таблицы строк), динамично корректируемые в процессе работы.

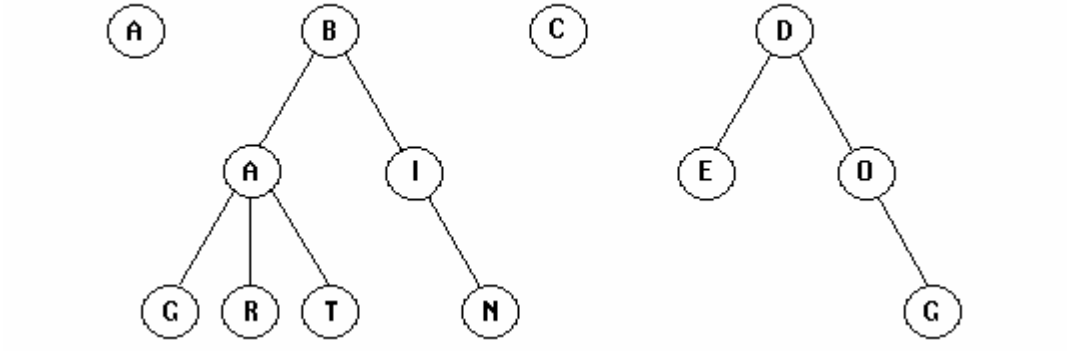


Рис.5.5. Древовидные структуры таблицы кодирования

Словарь можно логически представить с помощью абстрактных структур данных, содержащих набор древовидных структур, в которых каждый корень соответствует определенному знаку алфавита. При 8-разрядном формате символа количество таких деревьев равно 256. Древовидные структуры представляют собой набор известных строк, которые начинаются одним определенным символом, а каждый узел дерева представляет одну строку из этого набора.

Древовидные структуры (рис.5.5) отображают строки: *A, B, BA, BAG, BAR, BAT, BI, BIN, C, D, DE, DO и DOG. C* каждым узлом связано кодовое слово, используемое для идентификации узла. В процессе инициализации осуществляется установка словарей кодера и декодера в начальное состояние, при котором каждое дерево содержит только корневой узел. Вновь образованные строки заносятся в таблицы кодера и декодера в виде статей словаря. Причем индекс первой статьи $N_{СТ} = m_1 + N_У$, где m_1=256 - количество символов первичного алфавита источника; $N_У = 3$ - количество управляющих слов. Новая строка формируется путем добавления очередного одиночного символа к существующей строке, что выражается в добавлении нового узла к кодовому дереву. В процессе компрессии данных выполняются пять основных операций.

1. *Сопоставление строки*, при котором последовательность символов из ООД сопоставляется со статьей словаря. Если строка соответствует статье словаря и не является созданной во время последнего вызова процедуры сопоставления строки, то вводится следующий символ и добавляется к строке. Затем этот шаг повторяется. Если строка не соответствует статье словаря, или соответствует статье, созданной во время последнего вызова процедуры сопоставления строк, то последний символ убирается. Полученная укороченная строка представляет таким образом самую длинную сопоставленную строку, а последний (отброшенный) символ является несопоставленным символом.

2. *Кодирование*, при котором кодовое слово соответствующей статьи словаря представляется двоичной комбинацией длиной $L_{КС}$ бит.

3. *Передача кодовых слов* в устройство защиты от ошибок.

4. *Модернизация словаря*, в процессе которой создается новая статья словаря с помощью соответстующей статьи словаря и несоответствующего символа. Новая строка не вносится в словарь только в случае превышения максимальной длины строки.

5. *Восстановление статьи словаря* для повторного использования после заполнения всех доступных статей.

При декодировании также применяется операция сопоставления строк, в процессе которой восстанавливаются закодированные символы. Детально алгоритмы кодирования и декодированияданных рассмотрены в п.5.2. Важной особенностью процедуры *V*.42*bis* является то, что

максимальное количество используемых кодовых слов и соответственно число битов в кодовой комбинации рекомендацией МККТТ не ограничивается. Обычно граничное значение устанавливается взаимным соглашением передатчика и приемника. Причем, его допускается изменять в процессе обмена данными путем передачи соответствующего управляющего слова на приемную сторону. Увеличение количества статей в словаре свыше определенного предела может привести к снижению эффективности сжатия. Для большинства типов данных высокая степень сжатия достигается при использовании 11-разрядных кодовых слов, общее число которых в этом случае равно 2048.

При определенном характере потока информации, поступающей от источника, когда на продолжительных временных отрезках практически нет повторяющихся строк, может оказаться, что объем информации после сжатия увеличился в связи с тем, что на каждый символ затрачивается более 8 битов. В этом случае эффективнее оказывается передача данных в несжатом виде. Для исключения такой ситуации в процедуре *V*.42*bis* рекомендуется в процессе передачи оценивать эффективность сжатия информации. Это может осуществляться путем сравнения количества битов, требуемых для представления несжатого сегмента сообщения и соответствующего ему сжатого.

Последней особенностью процедуры сжатия по рекомендации V.42*bis* является сжатие последовательностей, содержащих одинаковые или повторяющиеся символы. В связи с тем, что компрессор может модернизировать свой словарь с упреждением (сначала обнаруживается строка, которая отсутствует в словаре, а лишь затем она вносится в словарь кодера), а декодер может изменить свой словарь только на основании ранее декодированных данных, необходимо убедиться, что кодер не использует новые статьи словаря прежде, чем они переданы на декодер. Пусть от источника поступает последовательность символов "*CCCC*..." (См. табл.5.7. В таблице условно предполагается, что выходной код символа “С” равен 3).

В процессе сжатия этих данных сначала вводится первый символ *C* и сравнивается со статьями словаря. Затем считывается второй символ *C* и присоединяется к первому, образуя строку *"CC"*. Так как в словаре отсутствует "*CC*", процедура сопоставления строки завершается, и в словарь вносится новая строка "*CC*". На выход кодера выдается код первого символа *C*, а последующее сопоставление строк начинается со второго (несопоставленного) символа *C*. Затем вводится третий символ *C* и добавляется ко второму, формируя строку "*CC*", и в таблице словаря ищется эта строка.

Таблица 5.7

Номер входного символа	1	2	3	4	5	6	7	8	9	10
Входная последовательность	C	C	C	C	C	C	C	C	C	C
Выходной код		3	3		4		4		5	
Новые статьи		4= CC			5= CCC					6= CCCC

В связи с тем, что она уже присутствует в словаре, но является статьей, созданной во время последнего сопоставления строки, то процедура сопоставления строки оперирует с сопоставленной строкой "*C*" (вторым символом входной последовательности) и несопоставленным символом *C* (третий входной символ). При этом в канал выдается кодовая комбинация для "*C*" и дальнейшее сопоставление строки начинается с третьего *C*. Четвертый символ *C* входной последовательности добавляется к третьему и в словаре осуществляется поиск строки "*CC*". Если "*CC*" обнаружена и не соответствует статье, созданной во время последнего сопоставления строки, считывается пятый знак *C* и добавляется к строке.

Контрольные вопросы к главе 5

1. За счет чего достигается сжатие текстовых сообщений при использовании метода *LZ* ?
2. Как представляется кодовая комбинация, отображающая кодируемую подстроку ?
3. Чем отличается алгоритм *LZ 78* от *LZ 77* ?
4. Как повышается эффективность кодирования методом *LZ* ?
5. Опишите процедуру кодирования строк методом *LZW* .
6. Как происходит заполнение строк таблицы кодирования при сжатии методом *LZW*?
7. Каковы недостатки алгоритма компрессии методом *LZW* и как они могут быть устранены ?
8. Проиллюстрируйте на конкретном примере процесс компрессии и декомпрессии текстовых сообщений методом *LZW*.

9. Как обеспечить быстрый поиск строки символов в кодовой таблице ?

10. Как устраняются коллизии при хэшировании ?

11. В чем состоит особенность реализации процедуры сжатия *V*.42*bis* по сравнению с *LZW* ?

12. Как организована структура данных в процедуре V.42*bis* и в чем состоит ее преимущество?

Глава 6

СЖАТИЕ ФАКСИМИЛЬНЫХ СООБЩЕНИЙ

6.1. Особенности представления графической информации

Факсимильной связью называется область электросвязи, которая занимается передачей *неподвижных* изображений по каналам связи. Факсимильный способ передачи является универсальным, так как им можно передавать чертежи и графики, черно-белые и цветные фотографии, микрофильмы и текстовые документы. Принцип факсимильной связи состоит в том, что передаваемое изображение — *оригинал* разбивается на элементарные площадки. Яркость площадок при отражении падающего на них светового потока преобразуется в электрические импульсы, которые последовательно передаются по каналу связи. На приемной стороне эти электрические сигналы в той же последовательности преобразуются в соответствующие элементы изображения на носителе записи. В результате получается копия изображения *(факсимиле).*

В современных цифровых факсимильных аппаратах *(факсах)* для считывания изображения применяется состоящая из миниатюрных фотодиодов линейка *(сканер),* длина которой равна ширине поля *(строки)* изображения. В процессе перемещения носителя изображения (или самой линейки) все изображение разбивается на строки, состоящие из дискретного набора сигналов — *растрэлементов*. Таким образом, можно считать, что все изображение разбито на элементарные прямоугольные площадки — растрэлементы, размеры которых определяются разрешающей способностью оптической системы факсимильного аппарата. Разрешающая способность некоторых современных факсов имеет значение порядка 30 линий на миллиметр, при которой размеры растрэлемента составляют 0,03 × 0,03 мм^2. Если текстовое сообщение состоит из символов размером 2 × 3 мм^2, то изображение символа представляется 6670-ю растрэлементами черного и белого цветов. Причем,

белому цвету поля соответствует сигнал "0", а черному — "1". Изображение знака в этом случае можно записать в виде матрицы, состоящей из "0" и "1". Таким образом каждый символ рассмотренного текстового сообщения будет иметь свое факсимильное отображение, состоящее из 6670 бит. Нетрудно заметить, что факсимильный способ передачи обладает очень высокой избыточностью. Так для передачи любого символа, принадлежащего коду КОИ-8, достаточно 8 бит. При одной и той же скорости за время передачи символа факсимильным способом можно было бы передать кодовым способом 6670 / 8= 833 символа.

Низкая эффективность поэлементной передачи изображений имеет и свое преимущество, так как высокая избыточность позволяет осуществлять передачу по каналам со значительным уровнем помех. Искажение одного или нескольких битов в факсимильном сообщении приводит к появлению на изображении такого же количества малозаметных точек другого цвета размером всего лишь 30 × 30 мкм. В то же время аналогичные искажение при кодовой передаче вызывают ошибочную регистрацию одного или нескольких символов сообщения.

В современных факсимильных аппаратах документ формата А4 разбивается на 1188 строк по 1728 элементов в каждой строке. Это достигается при разрешающей способности сканирующего устройства 3,85 строк/мм (4 точки/мм) и 7,7 линий/мм (8 точек/мм). Непосредственная передача этого сообщения по каналу тональной частоты со скоростью 4800 бит/с заняла бы около 7 минут. Сокращение времени передачи факсимильного сообщения достигается за счет использования методов сжатия информации. Это позволяет по скорости передачи успешно конкурировать с кодовыми методами передачи текстовых сообщений. Поскольку передача текста происходит построчно, то можно считать, что сигналы строк являются реализациями некоторой случайной последовательности. Статистические характеристики этой последовательности позволяют оценить среднее количество информации, содержащееся в элементе изображения, а следовательно, и возможный коэффициент сжатия.

6.2. Общая характеристика методов кодирования факсимильных сообщений

Поэлементный метод передачи и хранения изображений требует большого объема данных. С целью его сокращения используются различные способы эффективного кодирования, позволяющие существенно уменьшить избыточность исходного сообщения. В зависимости от вида факсимильных сообщений существующие методы их эффективного кодирования можно разбить на две группы.

К первой группе относятся методы, связанные с кодированием *двухградационных* изображений, к которым принадлежат схемы, чертежы, синоптические карты, газетный, машинописный и рукописный тексты и т.д. Для воспроизведения таких изображений достаточно использовать два уровня яркости: уровень поля "0" и уровень яркости знаков и линий "1", т.е. двухградационное изображение является двоичным.

Ко второй группе относятся методы, ориентированные на кодирование многоградационных *(полутоновых)* черно-белых и цветных изображений, к которым относятся фотографии, телевизионные изображения и др. Сжатие многоградационных изображений основывается на комплексной обработке сигналов изображения, включающих дискретизацию и квантование, спектральную обработку и эффективное кодирование. Методы сжатия таких изображений рассматриваются в следуещей главе.

Методы кодирования двух- и многоградационных изображений в свою очередь можно разделить на две подгруппы. Первая подгруппа включает методы кодирования информации без потерь, которые позволяют так закодировать сообщение, что при отсутствии ошибок в канале связи декодирование позволяет точно восстановить исходное изображение. Эти методы получили название *неискажающих* методов сжатия данных *(Lossless Compression)*. Вторая подгруппа — кодирование с *частичной потерей информации (Lossy Compression)* отличается тем, что в процессе сжатия в сообщение вносятся определенные неустранимые искажения, которые считаются допустимыми для получателя. Такие методы используются при сжатии многоградационных и цветных изображений .

Методы кодирования с сохранением информации также можно разделить на методы *одномерного* и методы *двумерного* кодирования. В алгоритмах обработки, использующих одномерные методы, все сканируемые строки обрабатываются независимо друг от друга. В алгоритмах с применением двумерного способа одновременно обрабатывается несколько сканируемых строк. Поскольку при двумерном кодировании наряду с горизонтальной учитывается и вертикальная

корреляция, то при таких алгоритмах можно достичь большего сжатия данных, чем при одномерном кодировании. Однако системы, построеные на основе двумерного кодирования, более чувствительны к ошибкам передачи.

В методах двумерного кодирования можно выделить группу методов одновременного кодирования n строк и группу методов построчного кодирования. В первом случае параллельно обрабатываются элементы изображения n последовательно расположенных строк, а все группы из n строк обрабатываются независимо друг от друга. Поэтому вероятностная корреляция, существующая между двумя последовательными группами, не учитывается. В построчном последовательном кодировании при обработке каждой последующей строки используются элементы одной - двух предшествующих строк, а каждая строка кодируется с учетом существующей вертикальной корреляции между ней и предшествующими строками. Именно поэтому построчное последовательное кодирование в большинстве случаев более эффективно, чем одновременное кодирование n строк, при условии, что число строк в группе невелико. По использованию статистической зависимости элементов сообщения различают методы *нестатистического* и *статистического* кодирования. При нестатистическом кодировании строки двухградационного изображения интерпретируются как чередующиеся, статистически независимые серии белых и черных отрезков, причем на выход кодера выдаются не сами серии, а коды цвета и длины отрезков. При статистическом кодировании изображение рассматривается как статистический источник сообщений, который генерирует последовательности черных и белых элементов изображения с различной вероятностью. Затем эти последовательности кодируются неравномерным префиксным кодом.

6.3. Модели факсимильных сообщений

Простейшей моделью видеосигнала двухградационного изображения является случайная последовательность с независимыми значениями черного и белого ("1" и "0"). Для полного статистического описания достаточно задать вероятности появления значения сигналов $P(1)$ и $P(0)$. На практике эти вероятности определяются экспериментально путем подсчета числа белых и черных элементов двухградационного изображения. Так для большинства текстовых факсимильных сообщений $P(1) = 0{,}34$, а $P(0) = 0{,}66$. При этом энтропия, приходящаяся на один элемент изображения, равна

$$H = - P(1)\ log_2 (P1) - P(0)\ log_2 P(0) = 0{,}925 \text{ бит/эл.}$$

При равновероятном появлении "1" и "0" $H_{\max} = log_2 2 = 1$ бит/эл. Тогда избыточность текстового сообщения $R = 1 - H / H_{\max} = 0{,}075$.

В действительности элементы изображения не являются статистически независимыми. На изображении имеются протяженные участки однородной градации яркости. Поэтому для видеосигналов двухградационных изображений характерно наличие чередующихся серий (подстрок) единиц и нулей. В этом случае более точной моделью факсимильного сигнала является *марковская цепь первого порядка*. Степень статистической зависимости элементов в сигнале характеризуется переходной вероятностью $P(S_j / S_i)$, которая определяет вероятность появления сигнала в момент времени j при условии, что предыдущий элемент имел значение S_i . Для черно-белых факсимильных изображений, когда сигнал принимает два значения "1" и "0", для полного задания марковской последовательности необходимо указать начальные вероятности появления $S_o = 1$ и $S_o = 0$: $P_0(1)$ и $P_0(0)$, а также матрицу переходных вероятностей:

$$P(Sj/Si) = \begin{vmatrix} P(S_j=1/S_i=1) & P(S_j=0/S_i=1) \\ P(S_j=0/S_i=0) & P(S_j=1/S_i=0) \end{vmatrix}. \qquad (6.1)$$

Вероятность совместного распределения n значений марковской последовательности $S_O, S_1,..., S_n$ определяется выражением

$$P(S_0, S_1,\ldots,S_n) = P(S_0)\prod_{i=1}^{n} P(S_i / S_{i-1}) . \qquad (6.2)$$

Пример 6.1. Вычислить среднее количество информации и избыточность изображений машинописных документов и газетного текста для которых экспериментально измеренные переходные вероятности равны [13]:

$P_м(1) = 0{,}4$;	$P_м(0) = 0{,}6$;	
$P_м(1/1) = 0{,}65$;	$P_м(0/1) = 0{,}35$;	Для сигналов
$P_м(0/0) = 0{,}85$;	$P_м(1/0) = 0{,}15$.	машинописного текста.
$P_г(1) = 0{,}4$;	$P_г(0) = 0{,}6$;	
$P_г(1/1) = 0{,}95$;	$P_г(0/1) = 0{,}05$;	Для сигналов
$P_г(0/0) = 0{,}99$;	$P_г(1/0) = 0{,}01$	газетного текста.

Различие в значениях переходных вероятностей для сигналов газетного и машинописного текстов объясняется неодинаковой разрешающей способностью сканирующих устройств факсимильных аппаратов, используемых для передачи соответствующих текстов.

Требуется определить вероятности совместного появления трех черных элементов подряд для газетного и машинописного текстов и сравнить их с аналогичными вероятностями для изображения с независимыми элементами.

Количество информации для источника с зависимыми сообщениями определяется по формуле (2.3):

$$I_м = - P(0)[P(0/0)\log P(0/0)+P(1/0)\log P(1/0)] -$$
$$- P(1)[P(0/1)\log P(0/1)+P(1/1)\log P(1/1)] = 0{,}74 \text{ бит.}$$
$$I_г = 0{,}16 \text{ бит.}$$

Вероятность совместного появления трех черных элементов для газетного текста $P_г(1,1,1) = P_г(1)\ P_г(1/1) = 0{,}4 \times 0{,}952 = 0{,}36$, а для машинописного — $P_м(1,1,1) = P_м(1)\ P_м(1/1) = 0{,}4 \times 0{,}652 = 0{,}17$.

Для изображения со статистически независимыми элементами при той же вероятности появления черного элемента $P(1) = 0{,}4$ получим $P(1,1,1) = P^3(1) = 0{,}064$. Избыточность изображений газетного и машинописного текстов соответственно равна

$$R_г = 1 - I_г / I_{max} = 0{,}84 ; \quad R_м = 1 - I_м / I_{max} = 0{,}26.$$

Для источника, описываемого моделью с независимыми элементами и неравными вероятностями $P(1) = 0{,}4$ и $P(0) = 0{,}6$, избыточность $R = 1 - 0{,}97 = 0{,}03$. Коэффициенты сжатия $K_{сж} = I_{max} / I = 1 / (1 - R)$ для машинописного и газетного текстов соответственно равны 1,35 и 6,25.

Особенностью факсимильных двухградационных изображений является наличие чередующихся последовательностей *(серий),* состоящих только из черных или белых элементов. Следовательно, изображение можно представить *потоком отрезков* различной длины черного и белого цветов. Целесообразность подобного представления обусловливается существенной неравномерностью распределения вероятностей длин этих серий. Наиболее адекватной моделью такого сигнала является марковская цепь. Зная статистические характеристики марковского процесса можно определить вероятности длин серий.

Вероятность того, что серия длиной n содержит только белые элементы $P(0; n)$ определяется по формуле

$$P(0; n) = P(0)P(0/0)^{n-1}. \qquad (6.3)$$

Здесь в качестве $P(0)$ следует рассматривать не начальную вероятность появления "0", а стационарную вероятность значения возникновения "0" на любом шаге, которая находится из соотношения

$$P(0) = P(0/1) / [P(0/1) + P(1/0)],$$

а стационарная вероятность возникновения "1"

$$P(1) = 1 - P(0) = P(1/0) / [P(0/1) + P(1/0)]. \qquad (6.4)$$

Средняя длина серии определяется как математическое ожидание случайной величины

$$M(n) = \bar{n} = \sum_i n_i P(0, n_i). \qquad (6.5)$$

С учетом (6.3) получим $\bar{n} = n[1 - P(0)P(0/0)^{n-1}]$. Тогда количество информации, содержащееся в серии средней длины n

$$I_{\text{сер}} = \bar{n} I = I\{n[1 - P(0) P(0/0)^{n-1}] + 1\},$$

где I — количество информации, приходящееся на один элемент изображения. Максимальное количество информации в серии длиной n содержалось бы в случае равномерного распределения длин серий $I_{\text{сер max}} = n I$. В этом случае коэффициент сжатия для каждой серии длиной n определяется выражением

$$K_{\text{сж}}(n) = n / \{n[1 - P(0) P(0/0)^{n-1}] + 1\}. \qquad (6.6)$$

Анализируя это выражение, можно заметить, что существует некоторое оптимальное значение длины n, при котором $K_{\text{сж}}$ получается максимальным. Оно определяется прежде всего стационарной и переходной вероятностями, зависящих в свою очередь от разрешающей способности сканера и вида конкретного изображения.

6.4. Алгоритмы одномерного кодирования

При одномерном кодировании каждая из строк изображения кодируется без учета других. В зависимости от использования статистических свойств последовательностей белых и черных элементов в строке различают нестатистические и статистические методы кодирования. Нестатистическое кодирование не использует информацию о распределении вероятностей последовательностей черных и белых элементов изображения. Существует несколько разновидностей таких кодов [20].

Кодирование длин сегментов. Это название является общим для ряда алгоритмов сжатого описания цифрового видеосигнала, при котором строка двухградационного изображения представляется в виде последовательности отрезков *(сегментов)* однородных элементов. Одним из простейших является алгоритм кодирования длин сегментов КДС-1. В соответствии с этим алгоритмом кодирование начинается с отрезка любого цвета. Однородные отрезки разбиваются на группы по четыре элемента в каждой. Выделяются группы, содержащие только белые (0000) и только черные (1111) элементы. Количество одинаковых групп, следующих друг за другом, кодируется четырехразрядными двоичными числами. Группы, в которых содержатся разнородные элементы, не кодируются, а передаются в исходном виде.

Для представления длины сегмента используется шестибитовое число $b_6b_5...b_1$. Старшие два бита b_6b_5 определяют цвет группы: 00 - белый, 01 - черный, 10 - некодируемая группа, 11 - признак сигнала управления. Четыре младших разряда слова отображают в 16-м коде количество следующих подряд однородных групп. Если сегмент строки содержит более 15 одноцветных групп, то передается несколько шестиразрядных слов, в которых комбинации признака черного 01 или белого 00 повторяются, что позволяет декодеру легко определить наличие длинных сегментов. Биты $b_4b_3b_2b_1$ первого слова содержат младший разряд, а последующие - старшие разряды 16-ричного числа, отображающего длину сегмента. Для синхронизации кодера и декодера в начале каждой строки передается управляющая 6-разрядная комбинация начала строки, признаком которой являются единичные значения двух старших битов b_6b_5.

Пример 6.2. Закодировать две подстроки, состоящие из 32 элементов каждая, вид которых показан на рис. 6.1, а также строку из 1728 черных элементов.

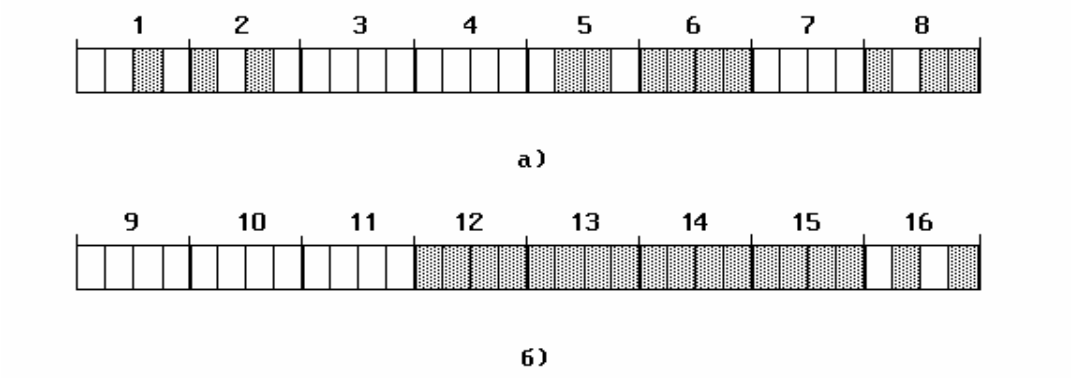


Рис.6.1. Одномерное кодирование длин сегментов

В связи с тем, что 1, 2, 5, 8 и 16-я группы подстрок а) и б) содержат разнородные элементы, то они представляются в некодированном виде соответственно словами 100010, 101010, 100110, 101011 и 100101. Здесь крайнему слева элементу изображения группы соответствует разряд слова b_4. Сегменты, состоящие из групп только белых или только черных элементов, кодируются следующим образом: 3 и 4 группы — 000010; 6 — 010001; 7 — 000001; 9, 10 и 11 — 000011; 12, 13, 14 и 15 — 010100.

Строка из 1728 элементов состоит из 432 групп по четыре элемента и кодируется тремя 6-битовыми словами, в которых два старших разряда содержат код черного 01, а четыре младших представляют в 16-ричной системе счисления код числа однородных групп в сегменте.

Некоторые другие способы кодирования длин сегментов учитывают наличие в факсимильных изображениях более длинных отрезков белого цвета. Так, например, в алгоритме КДС-2 отрезки белого кодируют 6-битовыми словами, а отрезки черного - трехбитовыми. Кодирование строки начинается с белого. Для задания начала строки формируется служебное слово. Если длина однородных сегментов превышает максимальное число, которым может быть закодирован соответствующий цвет, длина сегмента представляется несколькими словами.

Модифицированный код Хаффмена. Повышение степени сжатия факсимильных сообщений может быт достигнуто при использовании оптимального неравномерного кода Хаффмена. Используя статистику длин сегментов *(серий)* белого и черного, можно построить два кода Хаффмена: для серий белого и серий черного цветов. Очевидно, что в каждом коде число кодовых слов равно количеству элементов в строке (для стандартного факса 1728). Такой объем существенно усложняет реализацию кода Хаффмена. Поэтому он был модернизирован с учетом особенностей факсимильных сообщений. *Модифицированный код Хаффмена (МКХ)* относится к группе одномерных способов кодирования. Каждая строка изображения рассматривается как последовательность

чередующихся серий черных и белых элементов. Для обеспечения синхронизации при декодировании все строки начинаются с кодовой комбинации белого цвета. Если очередная считанная строка начинается с черного отрезка, то в начале строки передается белый отрезок с указанием нулевой длины.

Модификация кода состоит в том, что кодовые таблицы для белых и черных отрезков разбиваются на две части: основные кодовые слова и завершающие кодовые слова. Отрезки длиной более 63 элементов представляются в виде суммы основного отрезка с длиной кратной 64*N*, где *N*=1,2,...,27, и завершающего, длина которого дополняет основной отрезок до фактической длины. Для отображения сегментов длиной 64*N* применяются основные кодовые слова, а для дополнительных - завершающие кодовые слова. Если длина текущего сегмента меньше 64 элементов, то он кодируется только завершающим кодовым словом, а для серий длиной от 64 до 1728 элементов используется два слова. Кодовая последовательность сначала передается основным кодовым словом, представляющим длину отрезка, которая равна или короче требуемой длины, а затем следует завершающее кодовое слово, отображающее длину отрезка, дополняющего длину сегмента до истинного значения. Таблица МКХ приведена в приложении П8. Пробелы в кодовых словах таблицы введены только для удобства пользования ею.

Кодирование каждой строки изображения продолжается до тех пор, пока не будут переданы все сегменты строки. За каждой кодируемой строкой следуют кодовое слово конца строки (*EOL*), кодовая комбинация которого имеет вид 0000 0000 0001. Кроме того, эта комбинация появляется перед первой строкой документа.

***Пример 6.3*.**

Закодировать сегменты строки факсимильного сообщения, состоящие из следующих элементов: 8 белых; 46 черных; 920 белых и 1728 белых.

Воспользовавшись таблицей модифицированного кода Хаффмена (Приложение П8), запишем кодовые комбинации для

8 белых: 10011;
46 черных: 000001010110;
920 белых: (920 = 896 + 24) 011010011 0101000;
1728 белых: (1728 = 1728 + 0) 010011011 00110101.

Коэффициент сжатия при использовании модифицированного кода Хаффмена в среднем равен 5 (20%). Это позволяет реализовать передачу бланка форматом А4 по каналу тональной частоты со скоростью 9600 бит/с менее, чем за 1 минуту.

6.5. Алгоритмы двумерного кодирования

Из-за большой вертикальной (межстрочной) корреляции изображения целесообразно в процессе его сжатия обрабатывать несколько строк одновременно, т. е. использовать двумерное кодирование. На практике такое кодирование ограничивается одновременным кодированием двух, реже - трех строк. Существует несколько способов одновременного кодирования n строк.

Кодирование длин серий. В соответствии с этим способом одновременно анализируется несколько строк (например две, n=2) изображения, состоящеего из L_c строк по M_e элементов в каждой. При этом последовательно выделяется m-й фрагмент изображения, состоящий из двух элементов, один из которых берется из ($2L_c$ -*1*)-й строки, а второй - из соседней $2L_c$ -й строки (рис.6.2).

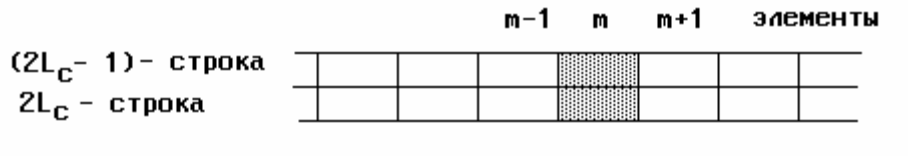


Рис.6.2. Двумерное кодирование длин серий

Выделенный фрагмент может находиться в четырех различных состояниях S_i (i=0,1,2,3), так как каждый элемент изображения принимает только одно из двух значений цвета: белый (Б) и черный (Ч). После такого преобразования исходное двухуровневое изображение, состоящее из $L_c \times M_e$ элементов, превращается в четырехуровневое с количеством элементов M_eL_c / 2.

Новое изображение можно рассматривать как марковский источник 1-го порядка, диаграмма состояний которого имеет вид, показанный рис.6.3.

Для обыкновенных изображений вероятности перехода P_{ij} из i-го в j-е состояние сильно отличаются друг от друга. В качестве примера в табл.6.1 приведены начальные вероятности состояний P_i и вероятности переходов P_{ij} для изображений, состоящих из таблиц, карт и текстов при сканировании с разрешающей способностью 6,6 строк/мм и 6,6 элементов/мм [18]. Эти вероятности неявным образом учитывают избыточность, вызванную вертикальной корреляцией.

Как видно из таблицы, длины серий вида Б/Б, Ч/Ч велики, а серий вида Ч/Б , Б/Ч очень малы.

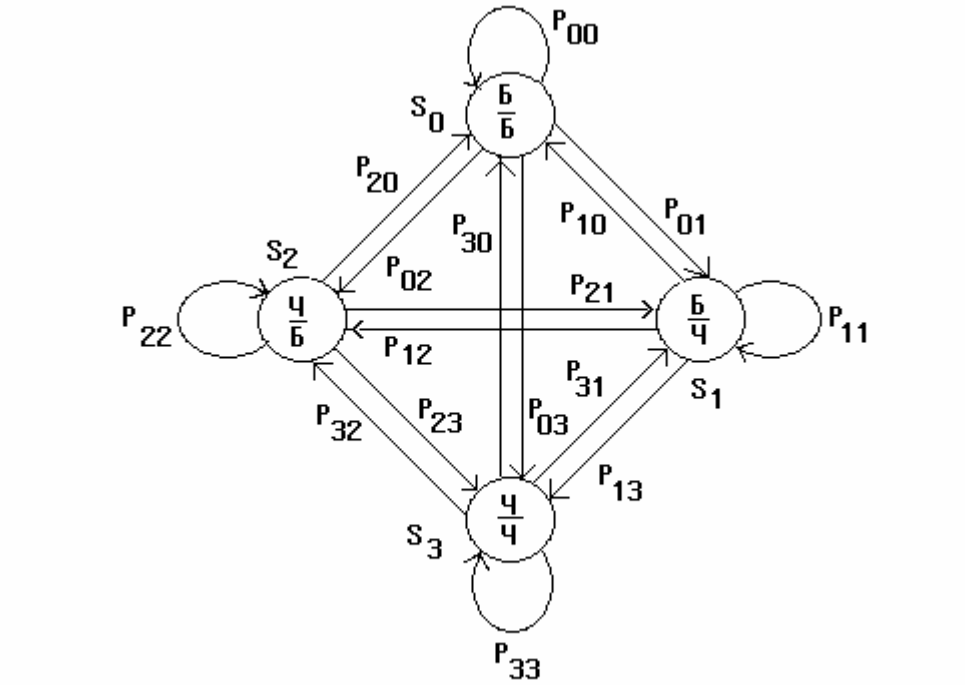


Рис. 6.3. Диаграмма состояний марковского источника 1-го порядка

Для представления таких последовательностей наиболее эффективны коды длин серий, построеные с учетом свойств серий каждого вида. При этом для маркировки видов серий требуется дополнительный код.

Таблица 6.1

Вероятности перехода P_{ij}				
i	0	1	2	3
Начальная вероятность P_i	0,889	0,021	0,020	0,067
j				
0	0,976	0,272	0,251	0,145
1	0,006	0,519	0,011	0,068
2	0,006	0,011	0,530	0,061
3	0,012	0,198	0,208	0,726

Наиболее целесообразно использовать для этой цели неравномерный код в связи с существенным различием вероятностей появления видов серий.

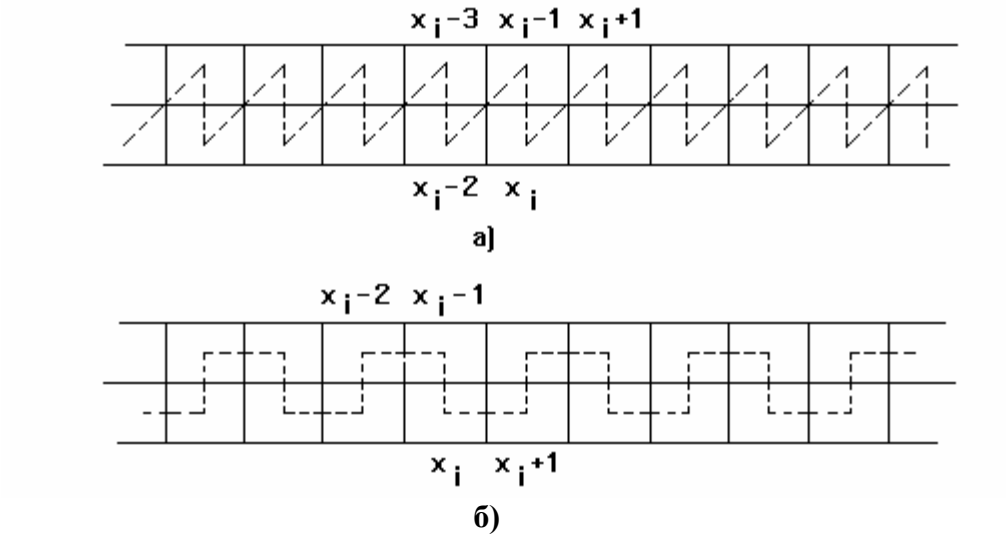


Рис.6.4. Схема зигзагообразного сканирования

Кодирование с зигзагообразным сканированием. При таком способе анализ изображения смежных строк основан на зигзагообразном сканировании (рис. 6.4), которое может иметь пилообразный (а) или волнообразный вид (*б*). В этом случае можно считать, что исходное двоичное изображение размером $L_c \times M_e$ имеет $L_c/2$ строк и $2M_e$ элементов. К каждой из удлиненых таким образом строк можно применить один из способов одномерного кодирования. Как видно из рис.6.4, *i*-му элементу изображения непосредственно предшествуют *i*-1, *i*-2 и *i*-3 -й элементы, которые расположены в непосредственной близости. В связи с этим полученную двоичную последовательность нельзя представить марковской цепью 1-го порядка, а ее следует рассматривать по крайней мере как марковский источник 3-го порядка. В этом случае целесообразно использовать специальные методы сжатия, основанные на предсказании (п.2.5).

Каскадное кодирование. В рассматриваемых выше способах группа строк обрабатывается поэлементно или поблочно. При каскадном кодировании элементы изображения в строке или группе строк обрабатываются одновременно. На первом шаге анализа кодер просматривает одновременно две соседние строки и разбивает их на четыре субблока одинаковой длины. Если субблок содержит по крайней мере одну "1", то кодер присваивает ему значение "1", в противном случае - "0". После первого шага преобразования образуется четырехразрядная кодовая комбинация A_1. На следующем шаге кодер обрабатывает только те субблоки, которым соответствует "1" в коде A_1. После разбивки каждого субблока на четыре новых кодер присваивает значение "1" субблокам,

содержащим "1", и значение "0" субблокам, которые состоят только из белых элементов, то есть из "0".

Таким образом формируется код A_2. Указанная процедура повторяется до тех пор, пока не сформируется четыре субблока, состоящих только из одного элемента. Для них кодер формирует комбинацию единичного блока. Закодированное изображение состоит из последовательности кодовых комбинаций A_1 A_2 A_3... и единичного блока. На рис.6.5 показан пример кодирования двух строк изображения, каждая их которых состоит из $M_e = 128$ элементов.

При программной реализации каскадного способа целесообразно использовать дерево кодирования (рис.6.6). На этом дереве все узлы одного уровня формируются на текущем шаге кодирования. Каждый узел соответствует определенному субблоку, полученному при разбиении на 4 части блоков предыдущего шага, причем узел блока, в котором присутствует хотя бы один черный элемент, принимает значение "1", а узел блока с белыми элементами — "0".

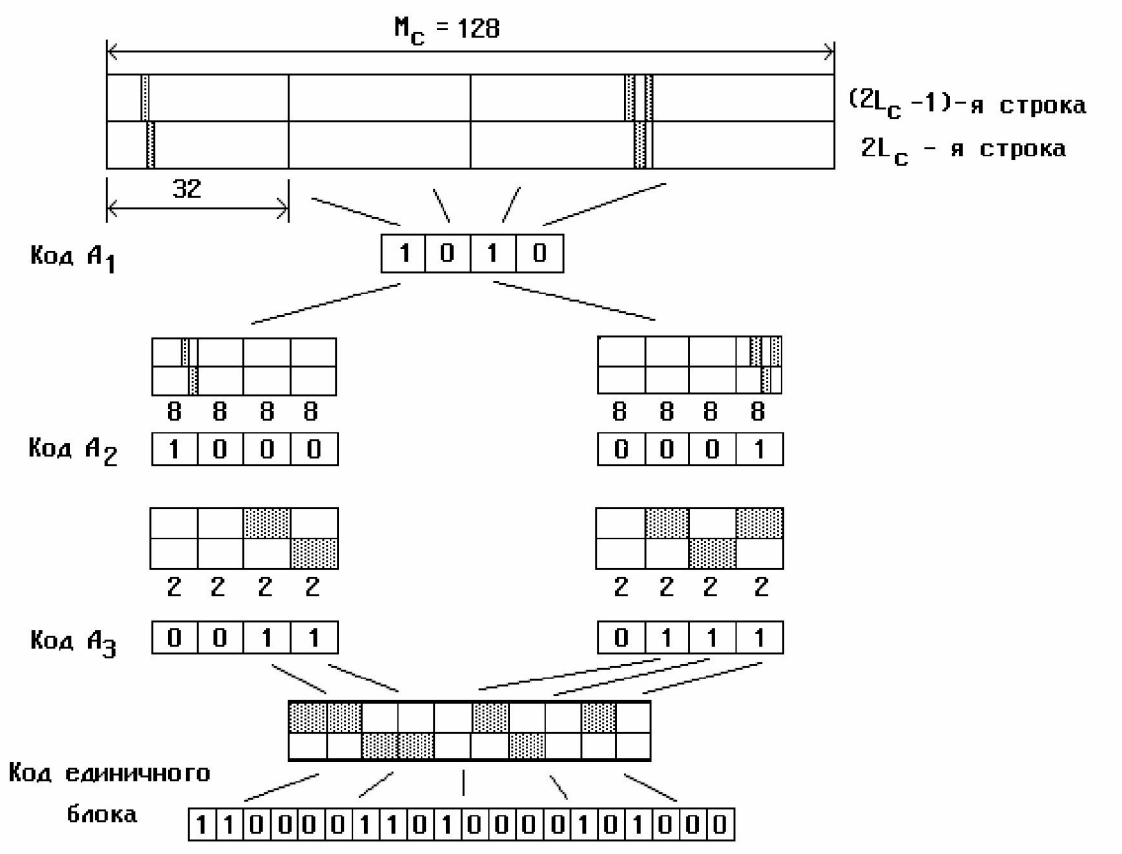


Рис.6.5. Схема каскадного кодирования

Каждый единичный узел становится внутренним узлом и имеет продолжение на следующем шаге, а нулевой узел является внешним.

Для образования кода соответствующих строк изображения осуществляется обход дерева по уровням слева направо и считывание значений узлов.

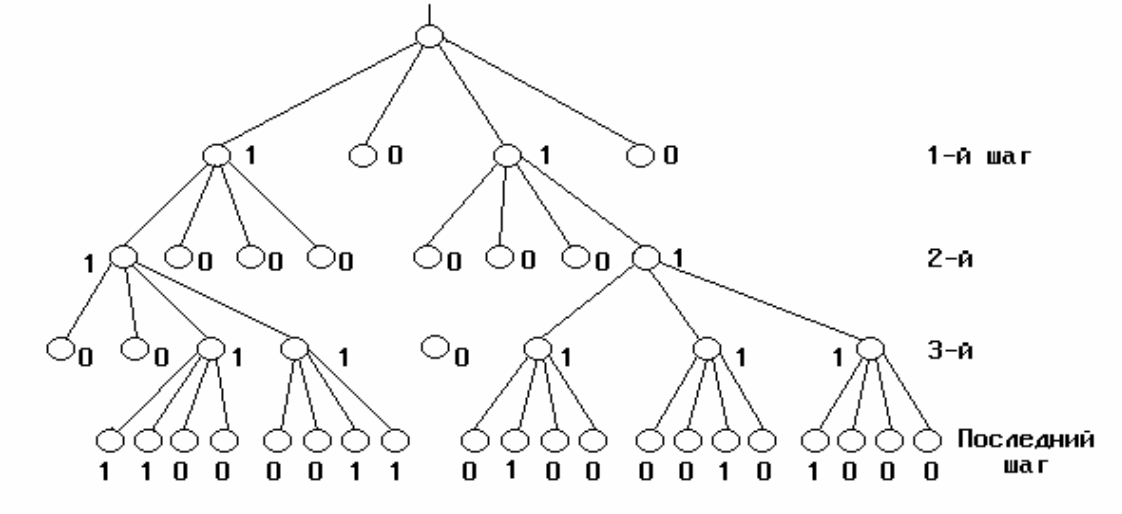


Рис.6.6. Дерево каскадного кодирования

6.6. Модифицированный код READ

Этот код относится к классу двумерных кодов. Он получил название "Код выбора относительного адреса элемента" - *READ* (*Relative Element Address Designate*). *READ* является расширением одномерного кодирования, в котором объединены некоторые приемы, применявшиеся в двух более ранних схемах сжатия *RAC* и *EDIC* [20]. В коде *READ* реализуется построчная схема анализа, при которой позиция каждого *переходного* элемента изображения (цвет которого отличается от цвета предыдущего элемента той же строки) сжимаемой строки кодируется с учетом позиции предыдущего переходного элемента этой строки, либо позиции переходного элемента опорной строки, расположенной непосредственно над кодируемой. После того как текущая строка закодирована, ее используют в качестве опорной для следующей сжимаемой строки. Чтобы предотвратить вертикальное распространение искажений в случае появления ошибок при передаче, одновременно кодируется не более K_c-1 последовательных строк. Обычно K_c выбирается равным 2 при нормальной разрешающей способности и 4 при высокой. Для описания процедуры кодирования по алгоритму READ определим ряд используемых обозначений (рис.6.7):

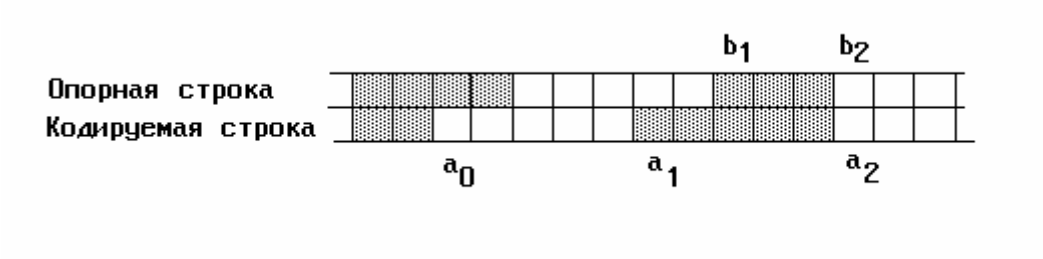


Рис.6.7. Схема обозначения элементов строки при кодировании по алгоритму READ

a_o: опорный или начальный переходный элемент кодируемой строки. В начале кодируемой строки a_o устанавливается на воображаемом белом переходном элементе, расположенном непосредственно перед первым действительным элементом кодируемой строки;

a_1: следующий переходный элемент, расположенный на кодируемой строке справа от элемента a_o, он имеет цвет, противоположный цвету элемента a_o, и является следующим, подлежащим, кодированию переходным элементом;

a_2: ближайший переходный элемент, расположенный в кодируемой строке справа от a_1;

b_1: переходный элемент, расположенный на опорной строке справа от элемента a_o и имеет тот же цвет, что и a_1;

b_2: следующий переходный элемент, расположенный на опорной строке справа от b_1.

Если какой либо из элементов a_1, a_2, b_1, b_2 не обнаружен на протяжении кодирования строки, то он устанавливается на воображаемом элементе, расположенном сразу же после последнего действительного элемента строки сканирования.

Режимы кодирования. В зависимости от взаимного расположения опорных элементов различают 3 режима кодирования.

1. *Переходный*. Устанавливается в случае, если элемент b_2 расположен слева от a_1. Этот режим позволяет обнаруживать белые и черные серии опорной строки, которые не являются смежными к соответствующим белым и черным сериям кодируемой строки (рис.6.8,*а*). В кодовой таблице (табл.6.2) переходный режим указывается одним кодовым словом "0001".

2. *Вертикальный*. При задании этого режима позициия элемента a_1 кодируется относительно b_1 (рис.6.8,*б*). Относительное расстояние a_1b_1 может принимать одно из семи значений: $V(0)$, $V_R(1)$, $V_R(2)$, $V_R(3)$, кодовым словом. Индексы R и L указывают на то, что элемент a_1 находится соответственно справа или слева от b_1, а числа в скобках указывают $V_L(1)$, $V_L(2)$, $V_L(3)$, каждое из которых представляется отдельным величины расстояний a_1b_1.

3. *Горизонтальный*. При невозможности закодировать позиции элементов в вертикальном режиме его кодируют в горизонтальном (рис.6.8, *б*). В этом случае длины серий a_oa_1 и a_1a_2 кодируются кодовыми словами H + $M(a_oa_1)$ + $M(a_1a_2)$, где H - кодовое слово метки "001", а $M(a_oa_1)$ и $M(a_1a_2)$ - кодовые слова модифицированного кода Хаффмена для заданного цвета и длины серии.

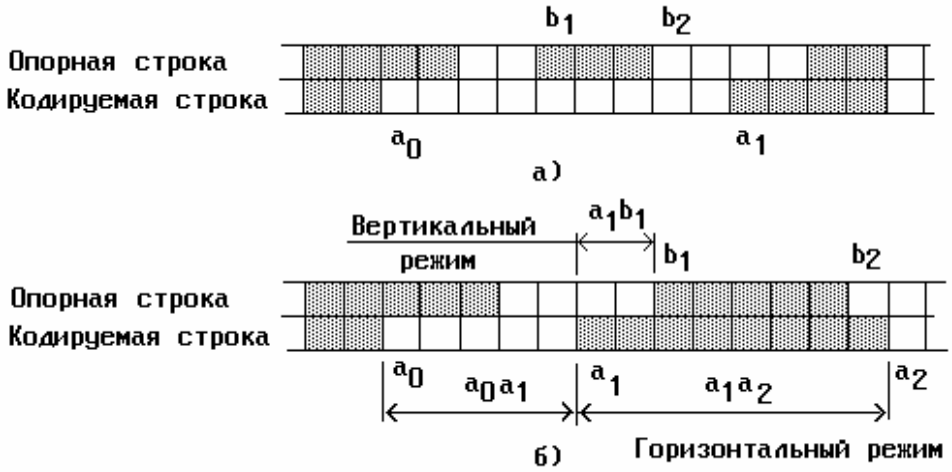


Рис.6.8. Схема режимов кодирования по алгоритму READ

Алгоритм кодирования. После инициализации очередной совокупности переходных элементов a_1, a_2, b_1, b_2 в процедуре кодирования задается соответствующий положениям элементов режим кодирования, из табл. 6.2 выбирается подходящее слово, а затем снова устанавливается новый опорный элемент a_o по способу, определенному ниже. Процедура кодирования описывается следующим алгоритмом.

Шаг 1: 1) Если b_2 обнаруживается раньше a_1, то задать переходный режим и выдать кодовое слово "0001". При подготовке к следующему шагу кодирования элемент a_o поместить на позицию, расположенную под b_2.

2) Если переходный режим не нужен, то перейти к шагу 2.

Шаг 2: Определить число элементов, разделяющих a_1 и b_1.

1) Если $|a_1b_1| < 3$, то относительное расстояние a_1b_1 закодировать в соответствии с вертикальным режимом. При подготовке к следующему этапу кодирования a_o установить на позицию элемента a_1.

2) Если $|a_1b_1| > 3$, то позиции a_1 и a_2 кодировать в соответствии с горизонтальным режимом, то есть передать кодовые слова H +$M(a_oa_1)$ + $M(a_1a_2)$. После кодирования новой позицией эталонного элемента a_0 считать позицию a_2.

Если кодирование первого элемента сжимаемой строки ведется в горизонтальном режиме, то для гарантии правильной передачи значения длины серии величина a_0a_1 заменяется на a_0a_1 - 1 . Поэтому, если первый

элемент строки - черный, то первое кодовое слово $M(a_0a_1)$ будет словом, представляющим белую серию нулевой длины.

3) Сжатие строки продолжать до тех пор, пока не будет закодирован воображаемый переходный элемент, расположенный сразу же за последним элементом кодируемой строки. Таким образом, в каждой строке кодируется ровно 1728 элементов, и приемник может проверить правильность декодирования каждой строки.

Таблица 6.2

Режим	Кодируемые элементы		Обозначение	Кодовое слово
Промежуточный	b_1, b_2		P	0001
Горизонтальный	$a_0\,a_1$, $a_1\,a_2$		$H(a_0\,a_1\,,a_1\,a_2)$	$001+ M(a_0\,a_1) + + M(a_1\,a_2)$
Вертикальный	a_1 под b_1	$a_1b_1 = 0$	V(0)	1
	a_1 справа от b_1	$a_1b_1 = 1$	$V_R(1)$	011
		$a_1b_1 = 2$	$V_R(2)$	000011
		$a_1b_1 = 3$	$V_R(3)$	0000011
	a_1 слева от b_1	$a_1b_1 = 1$	$V_L(1)$	010
		$a_1b_1 = 2$	$V_L(2)$	000010
		$a_1b_1 = 3$	$V_L(3)$	0000010

Значения кодовых слов для различных режимов приведены в табл.6.2. Сигналам данных одномерно или двумерно кодируемых строк предшествует один из двух байтов синхронизации строки *LSS1* или *LSS2*. *LSS1* передается в начале одномерно кодируемой строки и имеет вид 01111111. Байт *LSS2* предшествует двумерно кодируемой строке и представлен комбинацией 01111110. Для того, чтобы сигналы *LSS1* и *LSS2* отличались от остальных, в поток кодируемых данных после любых пяти последовательных "1" принудительно вводится "0". При необходимости в конце кодируемой строки и символом *LSS1(2)* вводится нулевая последовательность переменной длины. Конец передачи документа обозначается посылкой подряд шести сигналов *LSS1*, каждая из которых

является признаком окончания передачи данных и перехода аппаратуры в режим управления.

Как показали экспериментальные исследования алгоритма сжатия факсимильных сообщений READ, средний коэффициент сжатия равен 12,3 при обычной разрешающей способности и 15,8 при повышенной. По сравнению с одномерным модифицированным кодом Хаффмена алгоритм READ сокращает время передачи на 17% и 35% соответственно для обычного и высокого стандартов разрешающей способности.

Контрольные вопросы к главе 6

1. Проведите сравнительную характеристику факсимильного и кодового методов передачи текстовых сообщений.
2. Дайте общую характеристику методов сжатия факсимильных сообщений.
3. Чем вызвана вертикальная и горизонтальная корреляция факсимильных сообщений?
4. Как аналитически можно представить факсимильное изображение ?
4. Какое влияние оказывает выбор модели источника на коэффициент сжатия факсимильных изображений методом ДИКМ?
5. В чем состоит суть одномерного кодирования ?
6. Чем отличается модифицированный код Хаффмена от обычного и каковы его преимущества?
7. Расскажите об особенностях алгоритмов двумерного кодирования изображений.
8. С какой целью вводится зигзагообразное сканирование ?
9. В каких случаях целесообразно использовать каскадное кодирование ?
10. В чем состоит суть кодирования изображений методом READ?

Глава 7

СЖАТИЕ ПОЛУТОНОВЫХ ЧЕРНО-БЕЛЫХ И ЦВЕТНЫХ ИЗОБРАЖЕНИЙ

7.1. Кодирование изображений с частичной потерей информации

Изображения, в которых яркость изменяется непрерывно от уровня белого к черному, называются *полутоновыми* или *многоградационными*. Сигналы полутоновых изображений, в отличие от черо-белых факсимиле, являются аналоговыми. Поэтому для последующей их обработки с помощью ЭВМ они подвергаются дискретизации и квантованию. Сигналы полутонового неподвижного изображения представляют собой двухмерную функцию распределения яркости $B(x\ y)$ на плоскости с координатами x и y. Для подвижных изображений яркость элементарной точки изображения *(пикселя)* зависит не только от ее координаты $(x_i\ ,\ y_j)$ но и от времени t_k, т. е., сигнал яркости является трехмерной функцией $B(x_i\ ,\ y_j\ ,\ t_k)$.

Сжимаемые изображения предназначаются для восприятия их человеком, либо для обработки автоматическими устройствами. Если изображение кодируется для передачи зрителю, то уменьшить объем передаваемой информации можно, используя особенности восприятия зрительного анализатора [12]. Поскольку точность восприятия зрительного анализатора человека ограничена, то это позволяет считать некоторые искажения изображения незаметными или незначительными. Эта особенность дает возможность сжимать исходное изображение за счет потери части малозначительной информации, т. е., вносить определенные искажения. При декодировании, естественно, исключенная информация не может быть восстановлена и изображение воспроизводится с некоторой погрешностью. Различные методы кодирования вносят искажения разной степени. Поэтому при разработке системы компрессии изображений необходимо выбрать такой метод преобразования, который вносит наименее заметные искажения. В настоящее время большинство систем сжатия черно-белых и цветных

неподвижных и подвижных изображений являются системами с потерей части информации. В то же время имеются области применения обработки изображений с использованием автоматических анализаторов, где потери какой-либо части изображений не допускается.

При проектировании и оценке эффективности методов сжатия изображений необходимо иметь достоверную количественную меру качества изображения. К сожалению, не существует аналитической объективно адекватной меры качества изображения для различных систем сжатия изображений. Поэтому для характеристики качества применяются шкалы *субъективной оценки* качества изображения, испытательные изображения, численные зависимости качества от искажений [6,12]. Любая “хорошая” мера качества изображения должна быть коррелирована с субъективными оценками качества изображения, представленными в виде шкалы. Существуют две шкалы субъективной оценки: *шкала качества* и *шкала ухудшения изображения*. Обычно используется пятибалльная система оценок. Каждая ступень качества шкалы характеризует качество рассматриваемого изображения с учетом некоторого множества испытательных изображений. По шкале ухудшения можно оценить степень искажения кодированного изображения по отношению к некоторому исходному изображению. В табл.7.1 приведены шкалы качества и ухудшения, принятые в технике передачи изображений.

Таблица 7.1

Шкала качества и ухудшения изображения		
Качество	Оценка в баллах	Ухудшение
Отличное	5	Незаметно
Хорошее	4	Заметно, но не мешает
Удовлетворительное	3	Слегка мешает
Плохое	2	Мешает
Очень плохое	1	Очень мешает

Процедура оценки качества изображения осуществляется методом экспертной оценки. До эксперимента экспертам предъявляется неискаженное испытательное изображение. В течение эксперимента

периодически показывают неискаженное изображение, сменяющееся с оцениваемым или рядом с ним.

Числовые оценки качества изображений делятся на два класса: одномерные и двумерные. *Одномерные* используются только для одного изображения и измерительная шкала калибруется с помощью сравнения оцениваемого изображения с исходным. *Двумерные* являются дифференциальными показателями качества изображения до обработки и после. Одномерные методы, как правило, основываются на пространственном спектре изображения. Типичные меры качества изображения включают общую или среднюю энергию сигнала на всех частотах. Другой класс одномерных числовых мер основан на использовании статистических характеристик изображения, в частности, математического ожидания и дисперсии отсчетов дискретного изображения или гистограммы полутонового изображения. Одномерная гистограмма показывает распределение в изображении пикселей с определенным уровнем яркости. Такую гистограмму можно рассматривать как аппроксимацию одномерной плотности распределения вероятностей изображения. По ее форме можно судить о контрастности изображения. Аналогично двумерная гистограмма - это аппроксимация двумерной плотности распределения вероятностей изображения. Ширина такой гистограммы в диагональном направлении указывает на пространственную корреляцию изображения.

Двумерные меры качества изображения применяют наиболее часто при оценке качества сжатого изображения, поскольку они указывают на относительные искажения закодированного изображения по сравнению с исходным. Самой распространенной мерой является среднеквадратическая ошибка, представляющая собой разность между значениями соответствующих пикселей исходного и искаженного изображения. К сожалению, среднеквадратическая ошибка часто слабо коррелирована с субъективными оценками качества изображения [7].

7.2. Кодирование изображений методом ИКМ и ДИКМ

При *импульсно-кодовой модуляции* (ИКМ) сигнал изображения дискретизируется, каждый отсчет квантуется и кодируется двоичным кодом. В отличие от факсимильной передачи, где сигналы квантуются на два уровня: белый и черный и кодируются с точностью 1 бит на отсчет (нулем или единицей), количество уровней квантования полутоновых изображений устанавливается от 64 до 256, для кодирования которых затрачивается 6 ... 8 бит. Цветное изображение обычно представляет

собой комбинацию трех цветов: красного *(R),* зеленого *(G)* и синего *(B)*. Каждый из этих цветов обрабатывается независимо от других и для кодирования каждого из них выделяется от 6 до 8 битов на отсчет. Кодирование методом ИКМ является своего рода эталоном кодирования. Поэтому степень снижения скорости передачи за счет сжатия исходного сообщения одним из предлагаемых методов обычно оценивается по отношению к ИКМ. Снижение числа уровней квантования при кодировании методом ИКМ не целесообразно, так как это вызывает появление ложных контуров, которые возникают за счет скачкообразных изменений уровней на участках изображения с плавными изменениями яркости или цвета. В то же время энтропия полутонового изображения составляет 4 ... 6 бит на пиксел, что свидетельствует об избыточности кодирования сигналов изображения методом ИКМ.

Для уменьшения избыточности сигналов изображения применяется *дифференциальная импульсно - кодовая модуляция* (ДИКМ). При таком способе модуляции кодируется не полное значение отсчетов, представляющих уровень яркости элементов изображения (пикселей), а разность между текущим и предыдущим отсчетами сигнала. Еще в большей степени уменьшается избыточность в системах с ДИКМ, в которых кодированию подлежит разность между предыдущим отсчетом и предсказанным значением яркости последующего пикселя. Такой вид обработки видеосигналов является разновидностью кодирования с предсказанием (см. п.2.5), который наиболее широко используется в современных системах передачи изображений. При этой модуляции используется то обстоятельство, что непрерывно изменяющаяся яркость изображения *B(x,y)* может быть предсказана для различных *x*. Это используется для сжатия сообщения за счет передачи только той информации, которая требуется для коррекции предсказания. Поскольку предсказанное значение каждого отсчета вычисляется на основе только предшествующих значений, то оно имеется как в кодере, так и в декодере. Восстановление отсчетов в декодере производится путем суммирования ошибки предсказания с предсказанным значением этого отсчета. Структурная схема цифровой ДИКМ изображена на рис.7.1. Входным сигналом цифровой ДИКМ является ИКМ сигнал неподвижного изображения или последовательности телевизионных кадров. Отсчеты исходного сигнала изображения кодируются методом 8 - разрядной равномерной ИКМ. Для каждого входного отсчета яркости B_X точки X предсказатель по N-1 предыдущим отсчетам определяет предсказанное значение $\overline{B}_X$. Наибольшее распространение получили методы линейного предсказания. В этом случае предсказанное значение яркости X - го пикселя определяется по формуле

$$\overline{B}_X = \sum_{i=1}^{N-1} a_i B_{X-i}, \qquad (7.1)$$

где a_i - соответствующие весовые коэффициенты.

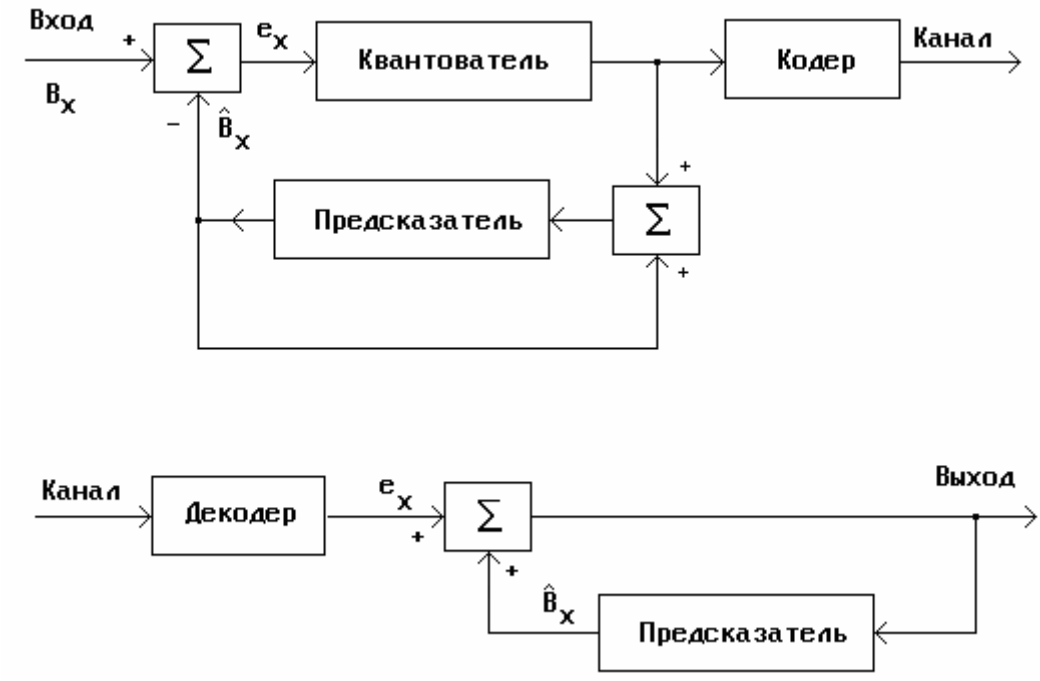


Рис.7.1. Структурная схема цифровой ДИКМ

Разность между предсказанным и истинным значениями M-го отсчета

$$e_M = \overline{B}_M - B_M \qquad (7.2)$$

называют *ошибкой предсказания* , а соответствующий ей сигнал ошибки - разностным сигналом. Весовые коэффициенты a_i выбираются таким образом, чтобы минимизировать дисперсию ошибки предсказания. Ошибка предсказания квантуется неравномерным оптимальным квантователем, который обеспечивает либо минимальную среднеквадратическую ошибку квантования, либо согласуется со свойствами восприятия шумов квантования зрительным анализатором человека. Затем квантованные значения кодируются одним из способов, позволяющим дополнительно сжать передаваемое сообщение.

Кодирование с предсказанием базируется на использовании как статистических свойств сигнала изображения, так и свойств зрительного анализатора. В общем случае отсчеты значений яркости соседних в пространстве пикселей коррелированы. В телевизионном сигнале, кроме того, существует временная корреляция между элементами изображений

в последовательных кадрах. Корреляция, или линейная статистическая зависимость, указывает на то, что при линейном предсказании отсчета на основе соседних пикселей ошибка предсказания будет иметь меньшую дисперсию, чем исходный отсчет. Сигнал ошибки имеет больший динамический диапазон, чем исходный: он может иметь не только положительные но и отрицательные значения. Однако распределение вероятностей для сигнала ошибки выгодно отличается от распределения вероятностей исходного сигнала. Условные распределения вероятностей, характеризующие связи между близкими элементами изображения, а вместе с тем и распределения вероятностей сигнала ошибки, весьма неравномерны. Возникает резкий пик вокруг нуля в распределении для разностного сигнала, так как ошибка предсказания, благодаря сильным связям между соседними элементами, как правило, мала. Хотя и в разностном сигнале остаются корреляционные связи (например, контурные линии объектов), в целом эти связи значительно ослабевают, что и дает возможность утверждать о декорреляции изображений при замене исходного сигнала разностным, полученным при линейном предсказании.

В качестве критерия декорреляции обычно используют *минимум среднего квадрата ошибки предсказания*. Весовые коэффициенты a_i, минимизирующие средний квадрат ошибки предсказания, могут быть вычислены, если известна корреляционная функция изображения. Чтобы осуществить оптимизацию линейного предсказания для ансамбля изображений, нужно знать усредненную по ансамблю функцию корреляции. При этом телевизионные сообщения упрощенно полагают стационарным случайным процессом.

В зависимости от того, как выбираются элементы изображения, по которым ведется предсказание, все системы с ДИКМ делят на три группы: одномерные, двумерные и трехмерные. В алгоритмах одномерного предсказания используется корреляция соседних элементов изображения вдоль строки развертки *(строчная корреляция)*. Такие алгоритмы часто применяют на практике, так как они просты в реализации и достаточно хорошо согласуются со статистическими характеристиками сигнала изображения.

Алгоритмы двумерного предсказания предполагают учет значений яркости пикселов, расположенных в текуще) и предыдущей строках. В этом случае уменьшается ошибка предсказания не только по координате *x* но и по *y*, что приводит к субъективному улучшению качества изображения. Алгоритмы одномерного и двумерного кодирования относят к *внутрикадровому* кодированию, так как при этом учитываются элементы изображения только текущего кадра.

В алгоритмах третьей группы предсказание осуществляется сразу по трем осям, то есть, кроме пикселов, используемых в двумерном предсказании, учитываются значения отсчетов этих пикселов в предыдущем кадре. Такое предсказание получило название *межкадрового* кодирования. Как правило, межкадровая избыточность достигает большой величины и кодирование с межкадровым предсказанием может быть весьма эффективным. Однако трудность его применения связана с необходимостью хранения отсчетов предыдущих кадров, что требует огромных затрат памяти.

На практике используют различные формулы расчета предсказанного значения яркости X-го пикселя $\overline{B}_X$. При сжатии неподвижных изображений в соответствии с рекомендациями международного стандарта JPEG [37] для прогнозирования отсчета текущего пикселя может быть использовано значение отсчета одного из соседних элементов изображения, расположенного в текущей или предыдущей строке, либо линейная комбинация отсчетов этих элементов. При учете одного пикселя яркость текущего элемента может быть определена по одной из следующих формул:

$$\overline{B}_X = B_A; \qquad \overline{B}_X = B_B \qquad \text{или} \quad \overline{B}_X = B_C \ .$$

Здесь буквенными индексами A, B и C обозначены пиксели, схема расположения которых показана на рис.7.2. Для более точного предсказания значения $\overline{B}_X$ стандартом рекомендуется использовать одну из формул [43]:

$$\overline{B}_X = B_A + B_B - B_C \ ; \qquad (7.3)$$

$$\overline{B}_X = B_A + (B_B - B_C)/2 \ ; \qquad (7.4)$$

$$\overline{B}_X = B_B + (B_A - B_C)/2 \ ; \qquad (7.5)$$

$$\overline{B}_X = (B_A + B_B)/2 \ . \qquad (7.6)$$

В подвижных (телевизионных) изображениях применяется чересстрочная развертка, при которой кадр изображения состоит из двух полей, строки которых перемежаются, то есть строки второго поля кадра (четные) располагаются между строками первого (нечетными). В таких системах рекомендуется определять предсказанное значение X-го пиксела по формулам [6,18]:

$$\overline{B}_X = B_A \ ; \qquad (7.7)$$

$$\overline{B}_X = (B_A + B_D)/2 \ ; \qquad (7.8)$$

$$\overline{B}_X = 3(B_A + B_C)/4 - B_B/2 \ ; \qquad (7.9)$$

$$\overline{B}_X = (B_E + B_F)/2 \ ; \qquad (7.10)$$

$$\overline{B}_X = B_A + (B_E + B_F)/2 - (B_C + B_H)/2 \ ; \qquad (7.11)$$

$$\overline{B}_X = B_{X©} \ . \qquad (7.12)$$

Схема размещения пикселов для текущего и предыдущего кадров показана на рис. 7.3. Штриховой линией обозначены перемежающиеся строки чересстрочной развертки.

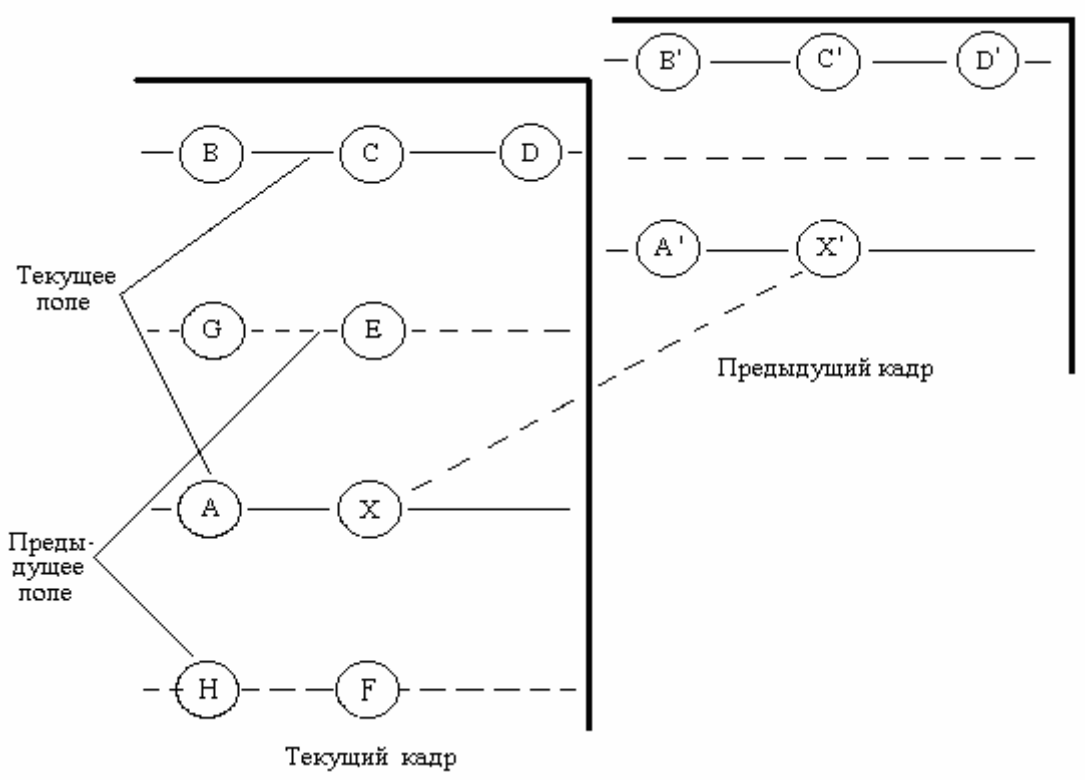


Рис.7.3. Схема размещения пикселей телевизионных изображений

Выбор формулы (7.3 — 7.12) для расчета $\overline{B}_X$ определяется характером изображения и его статистическими характеристиками, а также требуемым качеством восстанавливаемого изображения. Зачастую выбор той или иной формулы осуществляется экспериментальным путем. Во многих случаях целесообразно использовать различные формулы предсказания в зависимости от характеристик сжимаемых изображений.

7.3. Кодирование изображений с преобразованием

Кодированием с преобразованием называется процедура, при которой получаемый с помощью ИКМ видеосигнал до передачи подвергается некоторому обратимому преобразованию с последующим квантованием и кодированием. Целью преобразования является *превращение статистически зависимых элементов* изображения *в независимые коэффициенты*. Примечательным свойством преобразования является то, что все “важные” коэффициенты сосредоточиваются в определенной зоне. Менее значимые коэффициенты преобразованного изображения отбрасываются. Тем самым уменьшается объем исходного изображения. Коэффициент сжатия зависит от числа сохраняемых коэффициентов и может достигать 10. Входной сигнал $S_{и}$, который представляет собой оцифрованные отсчеты элементов изображения, вначале разбивается на блоки (фрагменты) размером $M \times N \times K$. Здесь M и N — количество элементов изображения в строке и столбце соответственно, а K — количество кадров изображений. Для неподвижных изображений $K = 1$.

Структурная схема кодирования с преобразованием показана на рис.7.4. Двумерному преобразованию подвергается все изображение или его области, называемые фрагментами. Типичный фрагмент изображения состоит из 8 × 8, 16 × 16 или 32 × 32 отсчетов, хотя для некоторых изображений он может быть увеличен до 256 × 256. Выбор такого размера фрагмента обусловлен тем, что интервал корреляции в изображениях не превышает 8 ... 32 отсчетов. В то же время, увеличение размера фрагмента резко увеличивает необходимый объем памяти, и затраты времени вычислителя на его обработку.

При сжатии изображений широко применяется преобразование Фурье, косинусное и синусное преобразования, преобразования Адамара, Хаара и Карунена-Лоэва [12]. В результате преобразования создается некоррелированный ряд чисел, называемый *трансформантами*. При этом число относительно больших трансформант невелико, в связи с чем трансформанты могут быть закодированы гораздо эффективнее, чем информация о самом изображении.

Полученные трансформанты разделяются для последующей обработки. Разделение базируется на зональном или пороговом принципе. При зональном разделении отбираются только те трансформанты, которые находятся в заранее определенной зоне (обычно в области нижних пространственных частот) и оказывают существенное влияние на субъективное качество изображения.

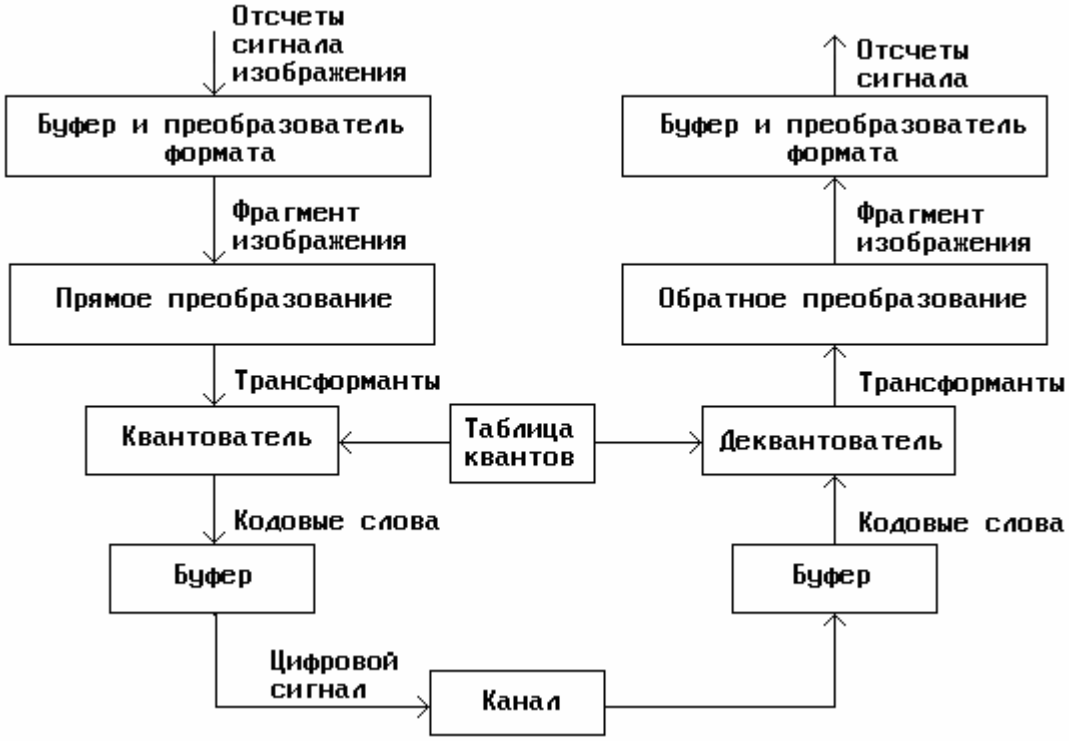


Рис.7.4. Структурная схема кодирования с преобразованием

В случае порогового разделения отбираются трансформанты, превышающие некоторый пороговый уровень. Затем производится квантование и кодирование отобранных трансформант, а остальные приравниваются к нулю. За счет *отбрасывания* части трансформант осуществляется сжатие исходного изображения. Очевидно, что при этом происходит искажение передаваемого изображения. Поэтому очень важным является исключение только тех трансформант, которые мало сказываются на качестве воспроизводимого изображения. Квантование является отображением области непрерывных коэффициентов в область их целочисленных значений, которые далее преобразуются в кодовые слова. Следует заметить, что квантование является вторым источником искажений при сжатии изображений, так как это процесс необратимого преобразования аналогового источника в его квантованный эквивалент. В некоторых схемах сжатия изображений отбрасывание несущественных трансформант производится после процесса квантования.

Квантование трансформант осуществляется в два этапа:

1) трансформанты нормируются измеренной дисперсией, которая определяется путем оценки большого числа фрагментов;

2) нормированные трансформанты обрабатываются квантователем, оптимальным для данной модели сигнала; число битов,

необходимых для представления квантованных трансформант, определяется исходя из предполагаемой дисперсии трансформант до квантования и допустимого уровня искажений; выходная целочисленная последовательность квантователя поступает в канал и затем на вход приемника, где происходит обратное преобразование последовательности проквантованных трансформант и восстановление исходного изображения.

Наилучшим способом преобразования сигналов изображения, обеспечивающим минимальную среднеквадратическую ошибку, является преобразование Карунена-Лоэва (К-Л). На практике это преобразование не нашло применения в связи с тем, что для его реализации требуется знание статистических характеристик ансамбля обрабатываемых изображений. Кроме того, для этого преобразования нет алгоритма быстрого вычисления. Наиболее близко по эффективности к преобразованию К-Л приближается дискретное косинусное преобразование, имеющее быстрый алгоритм вычислений.

Дискретное косинусное преобразование (ДКП) является родственным дискретному преобразованию Фурье. Однако ДКП работает не с двумерным сигналом (яркость B, время t), а с трехмерным (координаты изображения x, y и яркость B). В соответствии с алгоритмом ДКП последовательность отсчетов яркости пикселей преобразуется из трехмерного пространства в идентичное представление в частотной области. Другими словами, посредством косинусного преобразования осуществляется преобразование *пространственной* информации в *частотную* (спектральную). Оси x и y представляют собой пространственные частоты сигнала преобразования в двух различных измерениях. При этом пространственные частоты изображения по координатам x и y определяются числом черных штрихов (разделенных белыми промежутками такой же ширины, как черный штрих) в изображении, укладывающиеся в отрезке длиной 1 мм, перпендикулярной к этим штрихам прямой, совпадающей с направлениями x и y соответственно. Отсюда следует, что пространственные частоты имеют размерность $мм^{-1}$.

Дискретное косинусное преобразование является обратимым, то есть, посредством обратного косинусного преобразования осуществляется перенос сигнала из *частотной* области в *пространственное* представление. Косинусное преобразование оперирует с квадратной матрицей отсчетов яркости элементов изображения $B(x,y)$ размером $N \times N$ пикселов. Результатом преобразования является квадратная матрица $N \times N$ частотных коэффициентов *(трансформант)* $F(i, j)$. Формулы для прямого и обратного ДКП представлены соответственно выражениями:

$$F(i,j)=\frac{1}{\sqrt{2N}}C(i)C(j)\sum_{x=0}^{N-1}\sum_{y=0}^{N-1}B(x,y)\cos\frac{(2x+1)i\pi}{2N}\cos\frac{(2y+1)j\pi}{2N}\ ; \qquad (7.13)$$

$$B(x,y)=\frac{1}{\sqrt{2N}}\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}C(i)C(j)F(i,j)\cos\frac{(2x+1)i\pi}{2N}\cos\frac{(2y+1)j\pi}{2N}\ . \qquad (7.14)$$

Здесь *C(i)* и *C(j)* равны $1/\sqrt{2}$ для *i, j* = 0 и *C(i), C(j) = 1* при i,j>0; *B(x,y)* — значение отсчета яркости пиксела фрагмента изображения с координатами *x* и *y*.

На первый взгляд формулы представляются довольно громоздкими, однако вычисления по ним могут быть запрограммированы с помощью несложных процедур. Отдельные выражения этих формул могут быть заменены простыми табличными операциями. Например, произведение двух косинусов целесообразно вычислить заранее и сохранить результаты в памяти. Аналогично может быть составлена таблица значений коэффициентов *C(i)* и *C(j).*

7.4. Стандартная процедура кодирования изображений JPEG

Для обобщения опыта разработки и использования методов сжатия неподвижных полутоновых изображений и разработки международного стандарта МККТТ и МОС в 1991 году была создана организация, состоящая из группы экспертов, которая получила название *JPEG (Joint Photographic Expert Group).* Разработанный комиссией стандарт алгоритма обработки изображений получил название *JPEG*, который определяет правила сжатия многоградационных как черно-белых, так и цветных изображений. Стандарт состоит из ряда частей, включающих как сжатие данных без потерь, так и с частичным искажением преобразуемой информации. Компрессия без потерь базируется на основе ДИКМ с предсказанием, адаптивных алгоритмов Хаффмена или арифметического кодирования. Компрессия изображений с потерями использует метод косинусного преобразования с последующим квантованием.

Вносимые искажения информации при ее компрессии в соответствии с алгоритмом *JPEG* не должны приводить к заметному

ухудшению качества восстанавливаемого изображения, в частности, качество изображения по сравнению с оригиналом должно оцениваться на “отлично” или “хорошо”. Кроме этого метод должен быть применим для любых многоградационных изображений, а также быть достаточно простым в реализации [39,43].

Структура компрессора и декомпрессора видеоинформации по стандарту *JPEG* показана на рис.7.5. Обратите внимание на то, что она мало отличается от схемы сжатия с преобразованием, показанной на рис.7.4.

Рис.7.5. Структурная схема компрессора и декомпрессора по JPEG

Рассмотрим подробнее некоторые особенности процедуры обработки изображений *JPEG*. Кодируемое изображение разбивается на *блоки* размером 8 × 8 элементов (пикселов). Каждый блок представляет собой 64-точечный дискретный сигнал, состоящий из последовательности целых чисел в диапазоне $[0, 2^k - 1]$, которые затем преобразовываются в знаковые числа диапазона $[-2^{k-1}, 2^{k-1} - 1]$. Так, при 256 градациях яркости количество разрядов для кодирования отсчета изображения $k = 8$. Яркость пиксела путем масштабирования переносится из интервала 0 – 255 в диапазон от – 127 до 127. В блоке прямого дискретного косинусного преобразования *FDCT* осуществляется вычисление в соответствии с выражением (7.13) 64-х коэффициентов преобразования (трансформант), состоящих из последовательности целых знаковых чисел в диапазоне $[-2^{10}, 2^{10} - 1]$.

Выходной сигнал блока *FDCT* представляет собой 64-элементный массив, организованный в матрицу 8 × 8. Амплитуды трансформант

однозначно определяются исходным блоком отсчетов видеосигнала и представляет собой коэффициенты при дискретных частотах. Коэффициент при нулевой частоте определяет амплитуду постоянной составляющей *(DC)*, а остальные коэффициенты - амплитуды переменных составляющих *(AC)*.

В связи с тем, что элементы изображения во входном блоке изменяются слабо, то за счет косинусного преобразования удается сгруппировать трансформанты в области нижних пространственных частот. Следует еще раз подчеркнуть, что косинусное преобразование является обратимым и не приводит к сжатию сообщения. Оно осуществляет только подготовку данных к процедуре сжатия, которая осуществляется в квантователе.

Целью квантования является компрессия изображения путем задания точности квантования не большей, чем это необходимо для получения желаемого качества воспроизведения изображения. При сжатии трансформант можно снижать их точность квантования, причем тем больше, чем дальше расположена трансформанта от постоянной составляющей *DC*, находящейся в матрице с индексами (0,0). Снижение точности отображения трансформант уменьшает количество требуемых для их представления битов. Элементы, которые расположены ближе к постоянной составляющей, кодируются большим числом битов, а более удаленные — меньшим.

В алгоритме *JPEG* операция квантования реализуется с помощью матрицы квантования. Для каждого элемента матрицы трансформант имеется соответствующее ему значение кванта *Q(i, j),* расположенное в матрице квантования. Квантование осуществляется делением каждой трансформанты *F(i, j)* на соответствующий ей квант *Q(i, j)* и выделением целой части числа

$$F_Q(i,j) = \lceil F(i, j) / Q(i, j) \rceil . \qquad (7.15)$$

Значение *Q(i, j)* находится в диапазоне от 1 до 255. Величина *Q(i,j)*=1 обеспечивает наибольшую точность. По мере удаления от верхнего левого угла матрицы значения квантов увеличиваются. Нетрудно заметить, что, начиная с некоторых значений, когда *Q(i,j)>F(i,j)* кантованное значение F_Q *(i,j)* обращается в нуль, т. е. происходит невозвратимая потеря части информации. Выбор величины изменения квантов *Q(i,j)* от пикселя к пикселю определяет степень сжатия и качество воспроизведения информации. Несмотря на наличие стандартной таблицы квантов, *JPEG* предоставляет пользователям определенную свободу выбора элементов матрицы квантов в зависимости

от желаемого качества воспроизведения. В [37] предложено определять значения квантов по формуле

$$Q[i, j] = 1 + (1 + i + j) \times \gamma \ , \qquad (7.16)$$

где i и j — индексы элементов матрицы квантов, при $i, j=0,1,2, \ldots ,N–1$; γ — коэффициент качества воспроизведения изображения, задаваемый пользователем. Величину этого коэффициента рекомендуется выбирать в диапазоне от 1 до 25. Большие значения коэффициента качества также возможны, однако при этом качество воспроизводимого изображения резко ухудшается. В табл.7.2 представлена матрица квантов, рассчитанная по (7.16) при коэффициенте качества $\gamma = 2$.

При деквантовании производится операция умножения, т. е.

$$F'(i, j) = F'_{Q}(i, j) \times Q(i, j \ . \qquad (7.17)$$

Величина $F'(i, j)$ является входной для обратного косинусного преобразования. В табл.7.3, в качестве примера, приведены значения трансформант на выходе косинусного преобразователя произвольного фрагмента изображения, а в табл.7.4 — значения на выходе деквантизатора.

Таблица 7.2

Коэффициенты квантования $Q(i,j)$							
3	5	7	9	11	13	15	17
5	7	9	11	13	15	17	19
7	9	11	13	15	17	19	21
9	11	13	15	17	19	21	23
11	13	15	17	19	21	23	25
13	15	17	19	21	23	25	27
15	17	19	21	23	25	27	29
17	19	21	23	25	27	29	31

В связи с тем, что многие трансформанты приобрели нулевое значение, объем передаваемой информации существенно уменьшится.

Таблица 7.3

Значение трансформант перед квантованием							
92	3	–9	–7	3	–1	0	2
–39	–58	12	17	–2	2	4	2
–84	62	1	–18	3	4	–5	5
–52	–36	–10	14	–10	4	–2	0
–86	–40	49	–7	17	–6	–2	5
–62	65	–12	–2	3	–8	–2	0
–17	14	–36	17	–11	3	3	–1
–54	32	–9	9	22	0	1	3

Дальнейшим шагом *JPEG*-процедуры является кодирование квантованного изображения. Сначала разделяются трансформанты постоянной (*DC*) и переменной (*AC*) составляющих. Трансформанта постоянной составляющей является мерой среднего значения 63-х отсчетов изображения. Так как соседние блоки изображения обычно имеют сильную корреляционную связь, то постоянная составляющая последующего блока в большинстве случаев мало отличается от *DC* - составляющей предыдущего блока. Она преобразуется из абсолютного значения в относительное, и затем кодируется приращение текущей DC-составляющей по отношению к предыдущей (ДИКМ).

Таблица 7.4

Значение трансформант после деквантования							
90	0	–7	0	0	0	0	0
–35	–56	9	11	0	0	0	0
–84	54	0	–13	0	0	0	0
–45	–33	0	0	0	0	0	0
–77	–39	45	0	0	0	0	0
–52	60	0	0	0	0	0	0
–15	0	–19	0	0	0	0	0
–51	19	0	0	0	0	0	0

Трансформанты переменных составляющих преобразуются в последовательность способом “Зигзаг” (рис.7.6). Последовательность трансформант AC_{01} ... AC_{77} можно сжать методом кодирования длин повторяющихся символов (так как в образованной последовательности

имеется большое число нулей), либо хаффменовским или арифметическим кодированием.

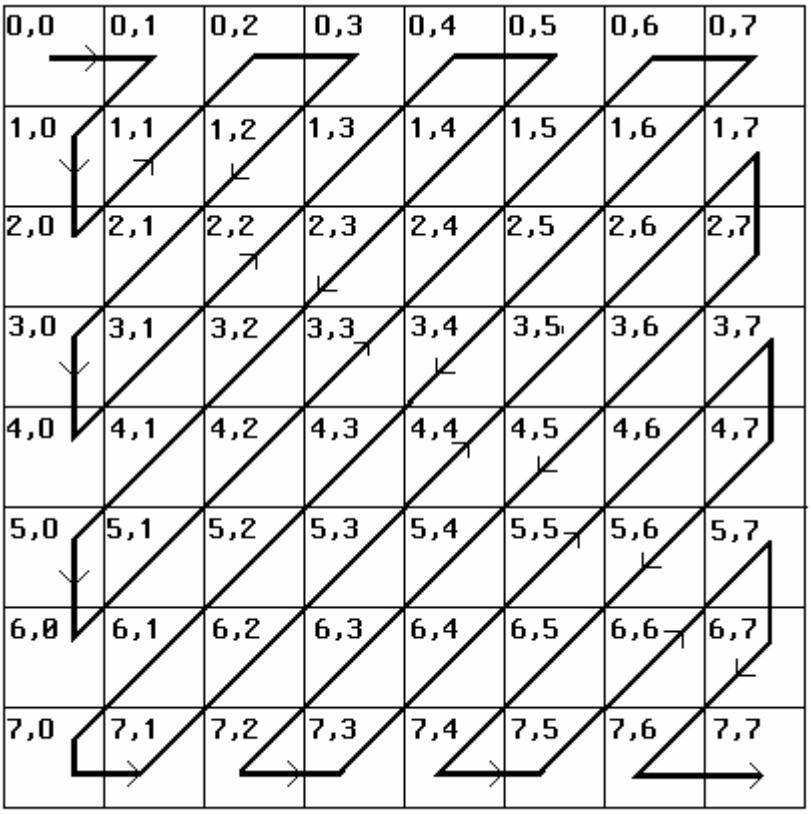


Рис.7.6. Схема считывания отсчетов фрагмента изображения

Для сжатия последовательности *AC*-трансформант фрагмента изображения рекомендуется использовать так называемое *энтропийное кодирование*. При этом способе амплитуды ненулевых *AC*-составляющих отображаются неравномерным кодом, не обладающим свойством разделимости кодовых комбинаций. Поэтому для их разделения перед каждой из комбинаций ставится индикатор, указывающий длину текущей кодовой комбинации.

Длинные группы нулей, расположенные между ненулевыми трансформантами, сжимаются методом кодирования длин последовательности одинаковых символов. В соответствии с рекомендацией *JPEG* отрезки зигзаг-последовательности, состоящие из группы нулевых и одной ненулевой трансформант, кодируются двумя символами: *SYM1* и *SYM2*. *SYM1* представлен одним байтом, старший полубайт которого указывает длину ряда нулевых трансформант кодируемого отрезка, а младший - размер (количество битов) второго символа *SYM2*, отображающего амплитуду ненулевой трансформанты, завершающую отрезок последовательности нулевых трансформант.

Очевидно, что полубайт может закодировать длину отрезка, состоящего из 1 ... 15 нулевых трансформант. На практике длина отрезка может быть больше 15. В этом случае длинная последовательность нулей представляется *SYM1* с кодом (15,0), отображающим группу, состоящую из 16 нулей. Таких символов при величине кодируемого фрагмента изображения 8 × 8 может быть до 3-х. Затем следует *SYM1* с кодом длины, дополняющей последовательность до действительного числа нулевых трансформант. Значение *SYM1* с кодом (0,0) используется для индикации конца кодирования текущего фрагмента отсчетов размером 8 × 8. Для кодирования амплитуды ненулевых трансформант используются целые двоичные знаковые числа, содержащие различное число битов (табл.7.5).

Таблица 7.5

Количество битов	Значение амплитуды
1	-1 , 1
2	от –3 до –2, от 2 до 3
3	от –7 до –4, от 4 до 7
4	от –15 до –8, от 8 до 15
5	от –31 до –16, от 16 до 31
6	от –63 до –32, от 32 до 63
7	от –127 до –64, от 64 до 127
8	от –255 до –128, от 128 до 255
9	от –511 до –256, от 256 до 511
10	от –1023 до –512, от 512 до 1023

Каждая группа битов кодирует симметричный диапазон амплитуд, состоящий из положительных и отрицательных значений. Старший бит этих чисел отображает знак, а остальные — значение амплитуды. Постоянная составляющая трансформант *DC* также кодируется неравномерным кодом и представляется посредством двух символов. Первый символ *SYM1* указывает длину, а второй *SYM2* - амплитуду *DC*-составляющей. В связи с тем, что постоянные составляющие кодируются дифференциальным способом, диапазон их представления увеличивается вдвое и изменяется от -2^{-11} до $2^{11} - 1$. Поэтому в табл.7.5 добавляется дополнительная строка, а *SYM1* принимает значение от 1 до 11. Такой неравномерный код по степени сжатия несколько уступает хаффменовскому или арифметическому кодам. Однако он значительно проще в реализации и является достаточно эффективным, когда

большинство трансформант состоит из малых значений, что чаще всего наблюдается на практике.

В процессе исследований процедуры сжатия установлено, что кодовые комбинации *SYM1*, отображающие длину нулевых последовательностей и амплитуду трансформант, характеризуются большой неравномерностью вероятности появления. Поэтому *JPEG* рекомендует производить дополнительное сжатие информации путем хаффменовского кодирования символов *SYM1*.

Пример 7.1. Закодировать способом энтропийного кодирования проквантованые трансформанты блока, приведенного в табл.7.6.

Таблица 7.6

Значение трансформант после квантования							
86	-5	3	0	0	2	0	-1
12	-3	0	0	0	0	0	0
-1	17	0	0	0	-2	0	0
0	0	-1	0	0	0	0	0
0	21	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	-7	0	0	0	0	0
0	0	0	0	0	0	0	0

На основе зигзагообразной развертки блока (рис.7.6) запишем последовательность значений трансформант:

86 -5 12 -1 -3 3 0 0 17 0 0 0 0 0 0 2 0 0 -1 21 0 0 0 0 0 0 0 0 -1 0 -2 0 0 0 0 0 0 -7 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 .

Используя описанный выше способ кодирования и данные табл.7.5, получаем последовательность символов *SYM1* и *SYM2*:

(7)(86) (0,3)(-5) (0,4)(12) (0,1)(-1) (0,2)(-3) (0,2)(3) (2,5)(17)
(6,2)(2) (2,1)(-1) (0,5)(21) (8,1)(-1) (1,2)(-2) (6,3)(-7) (0,0) .

Рассмотренный *JPEG*-алгоритм относится к монохромным изображениям. Цветные изображения обычно состоят из трех компонент: красного, зеленого и синего цветов (*RGB* - формат сигнала). В этом случае *JPEG*-алгоритм рассматривает изображение так, как будто оно состоит из трех отдельных изображений. При *RGB*-изображении сначала сжимается красная, затем зеленая и в конце синяя компонента. Для

каждой компоненты могут использоваться различные таблицы квантования и энтропийного кодирования, которые определяются статистическими характеристиками составляющих изображения. В процессе компрессии и декомпрессии осуществляется синхронное переключение таблиц в соответствии с обрабатываемой компонентой.

Контрольные вопросы к главе 7

1. В чем состоит отличие полутоновых изображений от факсимильных ?
2. На основании чего допускается частичная потеря информации при сжатии многоградационных изображений ?
3. Как оценивается качество восстановленных изображений при использовании алгоритмов сжатия с частичной потерей информации ?
4. Каким образом посредством предсказания яркости последующих пикселей можно осуществить сжатие полутоновых изображений ?
5. Приведите примеры формул расчета значений последующих пикселей.
6. За счет чего осуществляется сжатие изображений при кодировании с преобразованием ?
7. Какие факторы влияют на коэффициент сжатия изображений при кодировании с преобразованием ?
8. На какой стадии кодирования изображений с преобразованием происходит потеря информации и почему ?
9. Какое влияние на степень сжатия изображений по алгоритму JPEG оказывают значения коэффициентов таблицы квантов ?
10. В чем состоит суть энтропийного кодирования, используемого в процедуре JPEG ?

Глава 8

СЖАТИЕ ЦВЕТНЫХ ПОДВИЖНЫХ ИЗОБРАЖЕНИЙ

8.1. Проблемы передачи и хранения цветных подвижных изображений

Развитие и совершенствование методов цифровой обработки информации сделали возможным применение их для сжатия цветных подвижных изображений, характерных для множества телекоммуникационных приложений: компьютерные мультимедийные системы, системы хранения видеоданных на компакт-дисках *(CD-ROM),* видеотелефонная связь, телевидение.

Подвижные изображения можно разделить на две группы: *малоподвижные* и *быстроизменяющиеся*. Малоподвижные характерны для систем видеотелефона или телеконференций, когда с помощью телевидения передается изображение участников совещания и представляемые ими материалы во время конференции. В таких системах, используемых обычно для диалога, телевизионная камера, как правило, неподвижна, а сцены в основном содержат небольшие области, перемещающиеся по стационарному фону. Быстрое изменение происходит относительно редко, обычно только при переключении камер, смене передаваемых таблиц, рисунков, диаграмм, внезапном изменении освещения.

Быстроизменяющиеся изображения характерны для систем телевизионного вещания, видеозаписи и мультимедийных компьютерных систем. Телевизионный сигнал представляет собой последовательность отдельных кадров изображения, которые непрерывно сменяют друг друга. Количество кадров, представляемых зрителю в секунду, определяется постоянной времени сетчатки глаза. Приемлемой частотой, при которой мелькание кадров становится незаметным, является 24 кадра в секунду. Телевизионный кадр формируют из двух полукадров *(полей)* передаваемых последовательно во времени. Одна полукадр содержит нечетные строки, а второй - четные, т.е. осуществляется так называемая *чересстрочная*

развертка изображения. Поэтому частота мелькания полей оказывается в два раза выше частоты смены кадров. Чтобы наводки питающей сети не создавали дополнительных помех при формировании кадра, частота кадровой развертки выбрана равной частоте питающей сети (для европейских систем 50 Гц, а для американской - 60 Гц).

Для формирования цветовой палитры на экране монитора применяются три основных цвета: красный *R (Red),* зеленый *G (Green)* и синий *B (Blue).* Формирование таких цветов происходит в трехлучевых кинескопах, управление лучами которых осуществляется по соответствующим каналам *RGB*. Меняя интенсивность цветов и смешивая их, можно получить любой цвет или его оттенок.

Современные графические цветные дисплеи имеют разрешающую способность 1024 × 768 пикселов. Для управления одним пикселом требуется 24 бита (по одному байту на луч). Тогда для воспроизведения одного кадра небходимо 1024 × 768 × 3 ≈ 2,4 Мбайта информации. При выводе изображения на дисплей с частотой 30 кадр/с понадобился бы канал с пропускной способностью около 72 Мбайт/с, а для хранения десятиминутного фильма необходимо 2,4 × 10 × 60 = 1,4 Гбайта памяти. Очевидно, что без использования эффективных методов сжатия воспроизведение подвижных изображений на экране персонального компьютера не представляется возможным. Еще острее стоит проблема сжатия при необходимости передачи изображений по каналам связи. При этом потребовался бы канал связи, обеспечивающий передачу цифровых данных со скоростью 72 × 8 = 576 Мбит/с.

Передача цифровых сигналов изображения может осуществляться по каналам цифровой интегральной сети связи, созданной на базе систем уплотнения с ИКМ. За рубежом эти сети сокращенно называют *ISDN (Integrated Servis Digital Network).* Цифровая интегральная сеть построена по иерархическому принципу. В настоящее время наибольшее распространение получили два типа иерархий цифровых систем передачи (ЦСП): европейская и североамериканская. Незначительно отличается от североамериканской на высших уровнях иерархии цифровая сеть Японии. Основным каналом всех трех сетей является цифровой канал со скоростью передачи 64 кбит/с, который образуется путем уплотнения кабельных линий связи аппаратурой с временным разделением каналов и импульсно-кодовой модуляцией (ИКМ). В европейской иерархии нижний уровень образует первичная аппаратура уплотнения типа ИКМ-30, объединяющая 30 стандартных телефонных каналов тональной частоты и два служебных, которые используются для передачи сигналов синхронизации и управления. Скорость передачи в каждом из цифровых каналов составляет 64 кбит/с, а скорость передачи группового сигнала - 32×64=2048 кбит/с. Скорость 2048 кбит/с установлена в качестве стандартной для всех

европейских стран. Каналы для передачи таких потоков образуют первый уровень иерархии. В США первичная аппаратура уплотнения нижнего уровня формирует 24 канала основной цифровой группы (ИКМ-24), поэтому первый уровень иерархии имеет пропускную способность 24×64=1544 кбит/с. Четыре первичных потока объединяются в один вторичный с суммарной пропускной способностью 8448 кбит/с (6312 кбит/с для США). Каналы вторичной группы образуют второй уровень иерархии цифровой интегральной сети связи. В европейской иерархии на каждом уровне производится объединение четырех каналов предыдущих уровней (рис.8.1,а), а в североамериканской на верхних уровнях сети

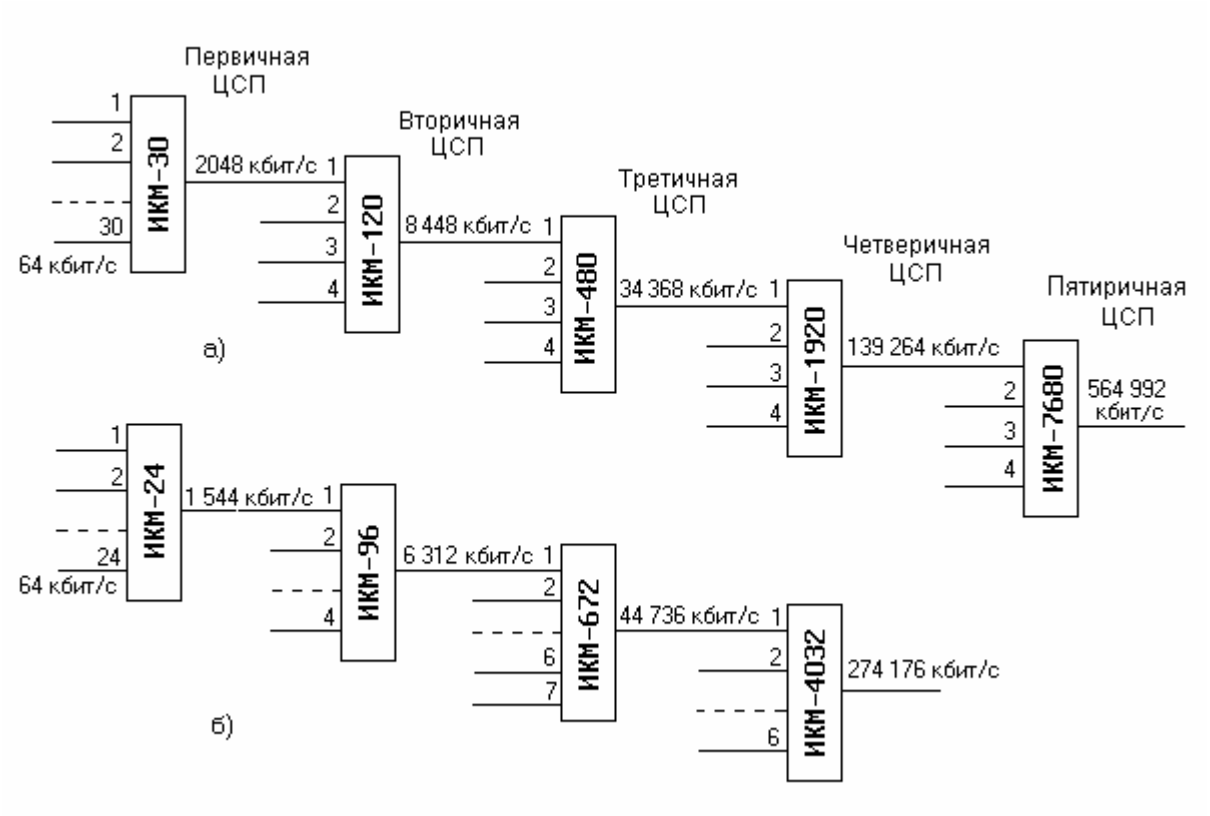


Рис.8.1. Иерархия цифровых сетей передачи:
а) европейская; б) североамериканская

объединяется различное количество каналов (рис.8.1,б).

Таким образом, цифровая интегральная сеть связи может обеспечить передачу данных со скоростями, не превышающими скорости соответствующих уровней иерархии. Поэтому при разработке компьютерных сетей и систем сжатия изображений необходимо учитывать ограничения на возможность передачи сигналов изображений по существующим каналам цифровой связи.

8.2. Формирование и кодирование сигналов цветных изображений

В настоящее время применяются три стандарта вещательного телевидения: системы *NTSC* (Национального комитета по телевизионным системам США), *PAL* (система с построчной коммутацией фазы) и *SECAM* (с последовательной передачей цветов и запоминанием). В Северной Америке и Японии применяется система *NTSC*, в странах Восточной Европы, в том числе, в странах СНГ, принята французская система *SECAM*, остальные европейские, южноамериканские и африканские страны используют систему *PAL*. Во всех трех стандартах применяется чересстрочная развертка. Основные параметры этих стандартов приведены в табл.8.1.

Таблица 8.1

Параметры	NTSC	PAL	SECAM
Число кадров в секунду	30	25	25
Число полей в секунду	60	50	50
Количество строк на кадр	525	625	625
Ширина полосы канала связи, МГц	6	8	8
Ширина полосы видеосигнала, МГц	4,2	5	6

Источником цветного подвижного изображения является телевизионная камера. Сигналы с камеры после соответствующей обработки передаются по каналу связи, воспринимаются приемником телевизионных сигналов и затем воспроизводятся на цветном или черно-белом мониторе. Для более полного понимания технологии сжатия сигналов цветных подвижных изображений рассмотрим кратко особенности формирования цифровых телевизионных сигналов.

В цветной телевизионной системе на выходе камеры формируются три сигнала, соответствующие красной *R*, зеленой *G* и синей *B* цветовых составляющих передаваемого изображения. Для более эффективной передачи цветных изображений по каналу связи и обеспечения их совместимости с черно-белыми телевизионными приемниками, трехкомпонентный *RGB*-сигнал преобразуется в монохромный сигнал

яркости *(luminance) Y* и две цветонесущие компоненты *(chrominance) U* и *V*. В соответствии с рекомендациями международного стандарта *CCIR* 601, разработанного международным консультативным комитетом по радиосвязи (МККР), яркостная компонента *Y* формируется путем линейного комбинирования *RGB*-сигналов

$$Y = 0{,}299R + 0{,}587G + 0{,}114B, \qquad (8.1)$$

а сигналы цветности *U* и *V* представляют собой линейные функции разности компонент цвета и яркости:

$$\begin{aligned} U &= 0{,}564\ (R - Y) \\ V &= 0{,}713\ (B - Y)\ , \end{aligned} \qquad (8.2)$$

в связи с чем они получили название *“цветоразностные сигналы”*.

Другим распространенным форматом видеосигналов является трехкомпонентный YC_BC_R -формат, в котором яркостная составляющая *Y* определяется аналогично (8.1), а цветонесущие компоненты C_B и C_R формируются путем масштабирования и сдвига сигналов *U* и *V* следующим образом:

$$\begin{aligned} C_B &= U/2 + 0{,}5; \\ C_R &= V/1{,}6 + 0{,}5. \end{aligned} \qquad (8.3)$$

На приемной стороне путем обработки *YUV*- или YC_BC_R-компонент осуществляется восстановление *RGB*-сигналов, которые используются для управления видеомонитором. В зависимости от области применения (компьютерные, мультимедийные или телевизионные системы) в качестве входных сигналов компрессоров изображений могут использоваться *RGB-, YUV*- или YC_BC_R - компоненты.

Все эти сигналы по своей природе являются непрерывными, и для обработки их цифровыми методами осуществляется квантование и преобразование в цифровую форму. Следует заметить, что при передаче изображений по каналам связи более целесообразным является цифровое кодирование *YUV-,* YC_BC_R -компонент, чем *RGB* в связи с тем, что они в значительной степени декоррелированы и основная энергия сосредоточена в яркостной компоненте.

Высокое качество передаваемых в цифровой форме подвижных изображений достигается рациональным выбором структуры и частоты дискретизации непрерывных видеосигналов. В настоящее время

применяются несколько типов структур дискретизации, к которым относятся:

ортогональная - отсчеты изображений расположены на одинаковых позициях в вертикальном направлении (рис.8.2,а);

со *строчным чередованием* точек отсчета; образуется в результате сдвига на половину интервала дискретизации отсчетов соседних строк данного поля (рис.8.2,б);

с *чередованием точек* по полям; образуется двумя ортогональными структурами в результате сдвига соседних полей на половины интервала дискретизации (рис.8.2,в).

На рис.8.2 крестиками условно показаны отсчеты сигналов нечетного поля кадра изображения, а закрашенными кружочками - четного поля.

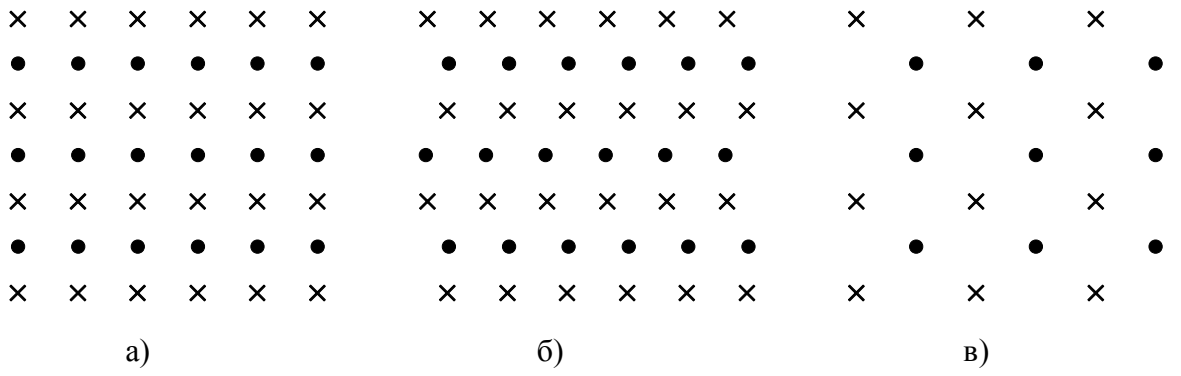


Рис.8.2. Примеры структур дискретизации внутри телевизионного поля:
а) ортогональная; б) со строчным чередованием; в) с чередованием по полям

Частота дискретизации изображения выбирается на основе теоремы отсчетов и жестко связана с частотами строк и кадров, так как при этом менее заметны искажения, возникающие в процессе дискретизации. В качестве стандартной частоты дискретизации для отечественной и американской телевизионных систем установлена $f_Д$=13,5 МГц. В зависимости от используемых компонент и требуемого качества обработки цифровых сигналов изображения, количество отсчетов на каждую компоненту может быть одинаково (для *RGB*) и одинаковым, или различным для сигналов *YUV* или YC_BC_R. Количество отсчетов на компоненту характеризуется соотношением частот дискретизации, которое представляется в виде отношения частоты дискретизации первой компоненты (яркостной составляющей, Y- или R - сигнала) к частотам других цветонесущих: f_1:f_2:f_3. Так отношение 2:1:1 показывает, что на два отсчета яркостного сигнала передается по одному отсчету цветонесущих компонент. В настоящее время наиболее часто применяется компонентное

кодирование с соотношением частот 4:4:4, 4:2:2, 4:1:1, 2:1:1 и др. Возможные области применения различных соотношений частот дискретизации приведены в табл.8.2.

Таблица 8.2

Соотношение частот $f_1 : f_2 : f_3$	Область применения
8 : 8 : 8	Обработка *RGB*-сигнала в системах мультимедиа
4 : 4 : 4	Высококачественная обработка в аппаратно-студийных телевизионных комплексах (АСК). Специальное применение
4 : 2 : 2	В АСК и цифровой магнитной видеозаписи. Международный обмен ТВ программами
4 : 1 ; 1	Передача данных по каналам связи. Системы мультимедиа
2 : 1 : 1	Передача по каналам связи. Спутниковая видеожурналистика

8.3. Кодирование изображений с предсказанием, компенсацией движения и интерполяцией

При сжатии подвижных изображений, обладающих значительной межкадровой избыточностью, широко используются принципы кодирования с предсказанием, которые подробно описаны в предыдущих разделах. Методы кодирования с предсказанием по своей сути являются локальными и поэтому весьма чувствительны к изменению характера данных. Нестационарность межкадровой статистики возникает в результате движения, охватывающего последовательные кадры. Для повышения степени сжатия подвижных изображений и качества их воспроизведения межкадровое кодирование с предсказанием осуществляют с учетом движения объектов. В процессе реализации алгоритма межкадрового кодирования с предсказанием производится

разделение *(сегментация)* изображения на подвижные зоны и стационарный фон, а затем осуществляется предсказание пикселов в данной зоне по элементам, принадлежащим одному или нескольким предыдущим кадрам.

Для сжатия видеоданных в случае подвижных изображений в принципе требуется кодировать только первичные (опорные) кадры и информацию о траектории движения каждого элемента. Для восстановления изображения достаточно перемещать элементы по их траекториям. На практике изображение разбивается на блоки фиксированных размеров $M \times N$ и измеряются параметры движения отдельных блоков. При этом предполагается, что каждый блок перемещается линейно и кодируется только вектор его перемещения. Движение в телевизионных и компьютерных изображениях можно моделировать кусочно-линейным перемещением движущихся объектов в сочетании с измерением величины и направления этого перемещения. Получив оценку параметров движения, сжатие можно осуществлять путем пропуска кадров изображения вплоть до поступления нового опорного кадра. Пропуск кадров широко используется при сжатии видеоданных для подвижных изображений даже в тех случаях, когда движение не измеряется.

Восстановление кадров при отсутствии компенсации движения может осуществляться повторением или интерполяцией кадров. При этом информацию о траектории движения используют как для предсказания, так и для интерполяции. В случае простейшей линейной интерполяции значения яркости пикселов блока определяются как среднее арифметическое соответствующих пикселов предыдущего и последующего опорных кадров

$$B_k(i,j) = [B_{k-1}(i,j) + B_{k+1}(i,j)]/2,$$

где B_{k-1}, B_k, B_{k+1} - элементы изображений блоков предыдущего, текущего и последующего кадров соответственно.

Простая линейная интерполяция применима только к телевизионным изображением с медленным движением. При быстроизменяющихся изображениях линейная интерполяция вызывает при воспроизведении скачкообразность перемещения и “смазывание” в областях движения. Процедура сопровождается смазыванием, поскольку при интерполяции происходит усреднение амплитуд элементов, принадлежащих движущемуся объекту и неподвижному фону. Чтобы предотвратить смазывание при умеренных и быстрых перемещениях, алгоритм интерполяции должен “компенсировать” движение объектов. Информацию о траектории движения можно использовать как для

предсказания, так и для интерполяции. Так, при предсказании с учетом движения пропущенный блок определяется как

$$B_k(i,j) = B_{k-1}(i+dx,j+dy),$$

а при интерполяции - следующим образом:

$$B_k(i,j) = [B_{k-1}(i,j) + B_{k+1}(i+dx,j+dy)]/2,$$

где *(dx,dy)* - векторы перемещения блока по координатам *x* и *y*.

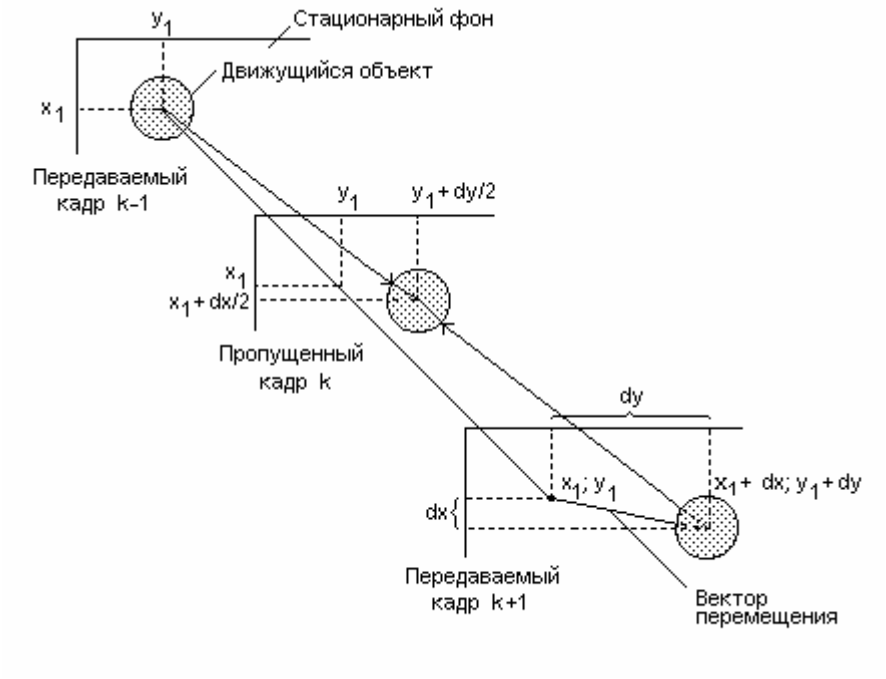


Рис.8.3. Иллюстрация процесса адаптивной интерполяции с учетом движения объекта

На рис.8.3 показан процесс восстановления элемента изображения с координатами (x_l, y_l) с помощью линейной интерполяции по одноименным элементам (x_l, y_l) переданных *k*-1 и (*k*+1) - го кадров.

При наличии в изображении нескольких движущихся объектов интерполяция должна локально адаптироваться к перемещениям каждого из них. Для этого необходимо обеспечить обнаружение и выделение *(сегментацию)* подвижных объектов, а также “открываемых” фоновых областей в текущем кадре и “закрываемых” фоновых областей в следующем кадре. При этом часть сегментов, относящихся к

“открываемым” областям текущего и “закрываемым” областям следующего кадров, приходится восстанавливать с помощью экстраполяции, а не интерполяции, поскольку необходимая информация для “открываемой” области содержится только в последующем кадре, а для “закрываемой” - в предшествующем. Стационарный фон можно восстанавливать с помощью линейной интерполяции.

8.4. Оценивание перемещения объектов в кадре изображения

Один из наиболее распространенных методов оценки движения основывается на процедуре согласования (сопряжения) блоков [24], суть которой сводится к следующему. Все изображение разбивается на блоки размером $M \times N$ с центром в текущем пикселе (i,j). При этом предполагается, что движение элементов блока до появления следующего кадра направлено в одну сторону и происходит с одинаковой скоростью. В связи с этим движение блока оценивается величиной смещения блока k-го кадра по отношению к расположению этого же блока в предыдущем (k-1)-м кадре (или в последующем (k+1)-м кадре). Прежде чем определить смещение определенного блока k-го кадра, необходимо найти рассматриваемый блок в последующем (k+1)-м кадре, а затем определить его координаты. Для этого в (k+1)-м кадре выделяется прямоугольный район *(зона)* поиска, центром которого является рассматриваемый блок. Внешние границы зоны поиска располагаются на расстоянии p от сторон блока (рис.8.4). Параметр p выбирается таким образом, чтобы рассматриваемый блок, передвигаясь по экрану с максимальной скоростью, не вышел за пределы поискового района, расположенного в последующем (или в предыдущем) кадре. В предположении, что максимальное горизонтальное или вертикальное перемещение блока составляет p элементов изображения, зона поиска S_R определяется выражением

$$S_R = (M+2p) \times (N+2p).$$

Затем, перемещая блок в районе поиска, покрывают им группу пикселов и выбирают ту из них, которая максимально согласуется с покрывающим блоком. В связи с тем, что за время между анализируемыми кадрами яркость (или цвет) ряда пикселов может измениться, нельзя

добиться полного совпадения. Координаты этой группы являются новым месторасположением блока в рассматриваемом кадре.

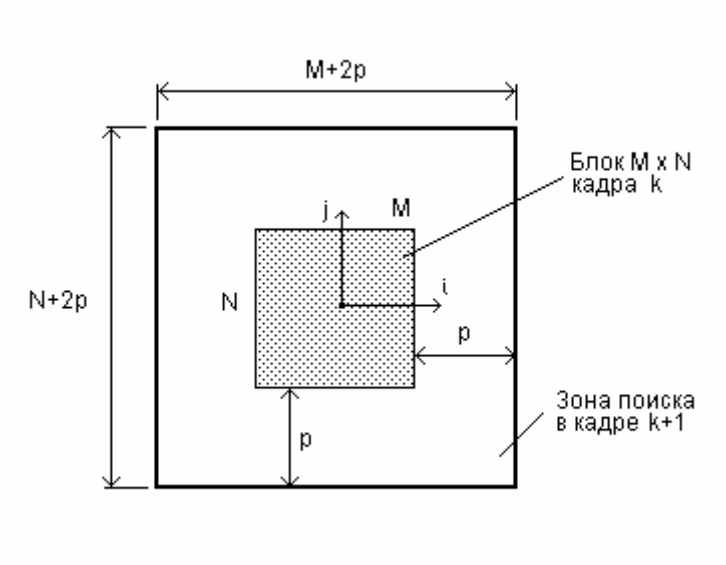


Рис.8.4. Схема зоны поиска ее параметры

Существует несколько критериев согласования группы пикселов с покрывающим ее блоком [24,39]. Одним из самых простых является критерий минимального абсолютного различия *MAD (Mean-Absolute Difference)*

$$MAD(dx,dy) = \frac{1}{MN}\sum_{i=-M/2}^{M/2}\sum_{j=-N/2}^{N/2}\left|B(i,j) - G(i+dx, j+dy)\right|, \quad (8.4)$$

где *B(i,j)* - блок изображения размером *M×N* текущего кадра; *G(i,j)* - такой же блок предыдущего *(k-1)*-го или последующего *(k+1)*-го кадров; *(dx,dy)*-вектор, представляющий местоположение в зоне поиска.

Поисковая зона задается следующим образом: *dx={-p,+p}* и *dy={-p,+p}*. В процессе поиска согласованного блока для каждой точки с координатами *(dx,dy)* вычисляется значение критерия *MAD(dx,dy)* по всем пикселам блока размером *M×N*. Варьируя координаты *(dx,dy)* в пределах поисковой зоны, находят точку *(dx,dy),* в которой различие группы покрываемых пикселов и данного блока минимально, т.е. найденная группа пикселов является искомым блоком. Координаты *(dx,dy)* определяют вектор перемещения блока.

Более точное согласование блоков может быть достигнуто при использовании критерия среднеквадратического различия *MSD (Mean-Squared Difference)*

$$MSD(dx,dy)=\frac{1}{MN}\sum_{i=-M/2}^{M/2}\sum_{j=-N/2}^{N/2}[B(i,j)-G(i+dx,j+dy)]^2. \qquad (8.5)$$

Высокая степень согласования блоков обеспечивается при использовании в качестве критерия функции взаимной корреляции *CCF(dx,dy),* по максимальному значению которой находят координаты вектора движения

$$CCF(dx,dy)=\frac{\sum_i\sum_j B(i,j)G(i+dx,j+dy)}{\sqrt{\sum_i\sum_j B^2(i,j)}\sqrt{\sum_i\sum_j G^2(i+dx,j+dy)}}. \qquad (8.6)$$

Несмотря на то, что критерии (8.5) и (8.6) обеспечивают более точное определение координат вектора движения, на практике чаще всего применяют критерий *MAD(dx,dy),* так как для его вычисления требуется значительно меньшее количество вычислительных операций при приемлемой точности оценки вектора движения.

Для сокращения количества вычислительных действий может быть использован упрощенный критерий классификации различия пикселов *PDC (Pixel Difference Classification)*[27]

$$PDC(dx,dy)=\sum_i\sum_j T(dx,dy,i,j), \qquad (8.7)$$

где *T(dx,dy,i,j)* - функция различия пикселов, которая определяется следующим образом:

T(dx,dy,i,j) = 1 если $|B(i,j) - G(i+dx, j+dy)| \le t_п$, и
T(dx,dy,i,j) = 0 в противном случае,

здесь $t_п$ - предварительно заданный порог.

В этом критерии каждый пиксел в блоке классифицируется как совпадающий (*T*=1) или не совпадающий (*T*=0) элемент изображения. Группа пикселов исследуемой зоны поиска, для которой *PDC* максимально, определяются как искомый блок.

Для нахождения вектора движения необходимо вычислить значения одного из критериев (8.4) - (8.7) во всех точках зоны поиска. Причем количество сдвигов блока $Q_С$ в зоне поиска шириной *p* равно

$$Q_C = (2p+1)^2 , \qquad (8.8)$$

что приводит к большому объему вычислений. С целью уменьшения числа сдвигов Q_C, требуемых для нахождения наилучшего согласования, разработан ряд методов поиска. Наиболее распространенными являются двумерно-логарифмический метод поиска, трехступенчатый поиск и метод поиска сопряженного направления [24].

Двумерно-логарифмический алгоритм основан на предположении, что при отклонении от направления максимального согласования в процессе поиска значение критерия *MSD(dx,dy)* начинает монотонно возрастать. Направление минимального отклонения определяется индексами *(dx,dy),* при которых минимизируется *MSD*. Алгоритм отслеживает направление минимального отклонения. Как показано на рис.8.5, на каждом шаге проверяются 5 точек поиска, причем если минимум оказывается в центре ячеек поиска или на границе зоны, то расстояние между точками поиска уменьшается. В приведенном примере для нахождения вектора перемещения в точке (*i*+2, *j*+6) потребовалось 5 итераций.

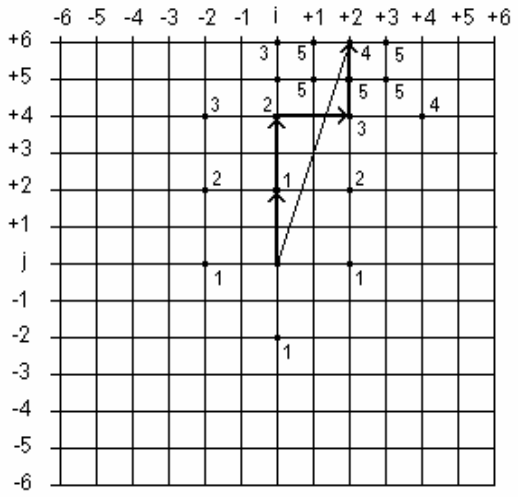


Рис.8.5. Процедура двумерно-логарифмического поиска

Трехступенчатый алгоритм тесно связан с двумерно-логарифмическим, однако на первом шаге исследуется восемь точек поиска (рис.8.6), не считая начальной точки с координатами *(i, j),* которые расположены вокруг начальной точки (центра выделенного блока). Первое приближение согласно критерия *MAD* соответствует точке (*i*+3, *j*+3). На второй итерации восемь точек поиска располагаются вокруг точки первого приближения и затем находится точка (*i*+3, *j*+5). На второй итерации восемь точек поиска располагаются вокруг точки первого приближения и

затем находится точка (i+3, j+5). Эта операция повторяется до получения требуемой точности. В данном примере для зоны поиска $p \le 6$ окончательное значение вектора перемещения (i+2, j+6) найдено за три шага.

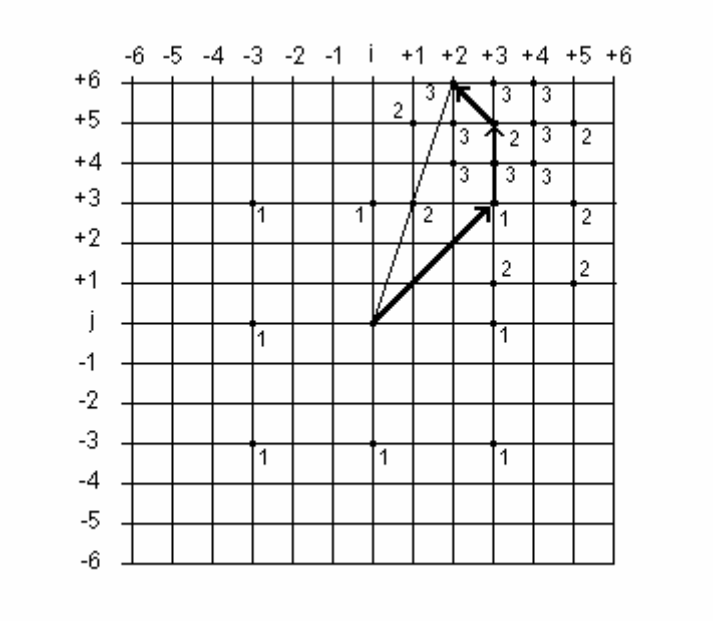


Рис.8.6. Процедура трехступенчатого поиска

Наиболее эффективным является алгоритм, получивший название *“поиск сопряженного направления*”. Алгоритм заключается в поиске направления минимального отклонения, которое определяется согласно критерию *MAD(dx,dy)*. Вначале определяется минимум в направлении *dx*, для чего вычисляются разности *MAD(dx*-1,*dy), MAD(dx,dy)* и *MAD(dx*+1,*dy)*.

В случае, если разность *MAD(dx*+1,*dy)* окажется наименьшей, вычисляется разность для следующей точки в этом же направлении, т.е. в точке *(dx*+2,*dy)* и определяется наименьшее из значений *MAD(dx,dy), MAD(dx*+1,*dy), MAD(dx*+2,*dy)*. Эта процедура продолжается до тех пор, пока разность с наименьшим значением не окажется между двумя разностями с большими значениями и таким образом определяется минимум в направлении *dx*.

После этого, начиная от найденного минимума, с помощью такой же процедуры ищется минимум в направлении *dy*. В примере, приведенном на рис.8.7, поиск в направлении *dx* дает точку *(dx*+2*, dy),* а в направлении *dy* - точку *(dx*+2*, dy*+6*)*.

Вычислительная сложность алгоритмов оценивается количеством требуемых для поиска точек. Если перемещение ограничивается величиной p=6, то прямой метод нахождения сопряженного блока требует Q_C=169 точек поиска.

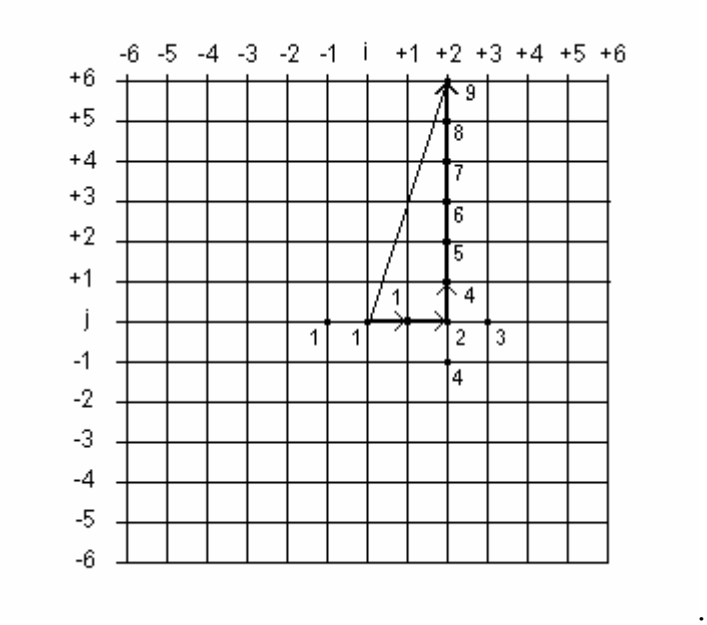


Рис. 8.7. Процедура поиска сопряженного направления

В табл.8.3 приведено необходимое количество точек поиска для приведенных алгоритмов в случае вектора перемещения *(dx*+2, *dy*+6*)* (колонка I) и наиболее неблагоприятного расположения вектора (колонка II).

Таблица 8.3

Алгоритм поиска	Требуемое число точек поиска		Требуемое число итераций	
	I	II	I	II
Двумерно-логарифмический	18	21	5	7
Трехступенчатый	25	25	3	3
Сопряженного направления	12	15	9	12

8.5. Сжатие данных видеосвязи по алгоритму Px64

При разработке компрессоров данных для видеотелефонной связи и телеконференций следует учитывать, что передача информации будет осуществляться по одному или группе каналов интегральной цифровой сети связи (*ISDN*), структура которой рассматривалась в п.8.1. Это означает, что скорость передачи должна быть кратна 64 кбит/с. Поэтому стандартная процедура сжатия видеотелефонной связи по рекомендации МККТТ *H*.261 получила название Px64. Здесь *P* - целочисленный коэффициент кратности скорости передачи, который изменяется от 1 до 30, т.е. возможно объединение 30 каналов основной цифровой группы с общей пропускной способностью 2048 кбит/с. Для видеотелефонной связи используют один или два (P=1,2) цифровых канала, а для видеоконференций, для которых сцены характеризуются большей подвижностью, значение коэффициента должно быть не меньше 6 ($P \geq 6$).

Процедура *H*.261 обеспечивает максимальный коэффициент сжатия полноцветных малоподвижных изображений, передаваемых в реальном времени. В алгоритме Px64 осуществляется внутрикадровое *(intraframe)* и междукадровое *(interframe)* кодирование. При внутрикадровом кодировании учитываются только фрагменты текущего кадра, в то время как при междукадровом используют ссылки на предыдущий и последующий кадры изображения. Кроме этого, алгоритмом предусматривается оценка скорости перемещения объектов в кадре, которая используется для повышения степени сжатия.

В соответствии с рекомендацией МККТТ предусмотрено использования двух форматов изображений: общий формат *CIF (Common Intermediate Format)* и укороченный *QCIF (Quarter CIF)*. Параметры этих форматов приведены в табл.8.4. Выбор типа используемого формата зависит от требуемого качества передаваемого изображения и пропускной способности имеющегося в наличии канала связи.

Пример 8.1.

Определить требуемый коэффициент компрессии K_C для передачи сигналов видеотелефонной связи в *QCIF* формате по основному цифровому каналу, если частота кадров видеотелефона равна 10 кадр/с.

Определим общее количество битов $V_и$ на один кадр изображения. Nк = (144×176 + 72×88 + 72×88) × 8 = 300 кбит. Объем информации, передаваемой камерой в секунду, равен $V_и$ = 300 × 10 = 3 Мбит/с. Тогда при скорости передачи в цифровом канале $V_{ЦК}$=64 кбит/с требуемый коэффициент компрессии $K_C = V_И / V_{ЦК} = 47$.

Таблица 8.4

Компоненты сигнала	CIF		QCIF	
	Лин/кадр	Пиксел/лин	Лин/кадр	Пиксел/лин
Яркостная	288	352	144	176
Цветонесущая (C_B)	144	176	72	88
Цветонесущая (C_R)	144	176	72	88

Пример 8.2.

Вычислить необходимый коэффициент компрессии K_C для передачи сигналов телеконференций в *CIF* формате по каналу, образованному объединением 8 основных цифровых каналов, при частоте кадров телевизионной системы 30 кадр/с.

Определим количество битов, генерируемых источником изображения в сек. $V_И$ = (288×352 + 144×176 + 144×176)×8×30 = 36,5 Мбит/с. Тогда требуемый коэффициент компрессии K_C= 36,5 / (8×64) = 74,8.

Из приведенных примеров видно, что требуемые коэффициенты сжатия могут быть обеспечены только при использовании комбинированных алгоритмов компрессии изображений и сжатием с частичной потерей информации. При этом целесообразно осуществлять комбинирование внутрикадрового и межкадрового кодирования. Стандартом *H*.261 рекомендуется при внутрикадровом кодировании использовать процедуру *JPEG* (см. гл.7), основой которого является дискретное косинусное преобразование (ДКП), квантование и энтропийное кодирование. Межкадровое кодирование рекомендуется реализовать на базе дифференциальной импульсно-кодовой модуляции (ДИКМ) с предсказанием. Структурная схема кодера Px64 показана на рис.8.8. Входным сигналом компрессора является трехкомпонентный YC_BC_R-сигнал с соотношением частот дискретизации 4 : 1 : 1, сгруппированный в блоки размером 8×8 пикселов. Причем, на четыре блока 8×8 яркостной компоненты Y формируется по одному блоку 8×8 цветонесущих составляющих C_B и C_R (рис.8.9). Набор этих шести блоков объединяется в так называемый макроблок (МБ), МБ = $4Y + C_B + C_R$, а последовательность, состоящая из 3×11 макроблоков образует группу блоков (ГБ).

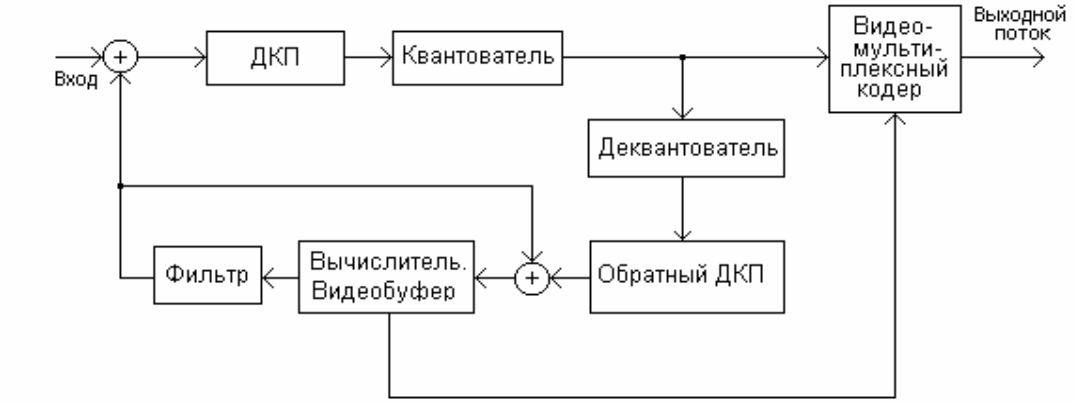


Рис. 8.8. Схема кодера системы сжатия видеоданных по стандарту Px64

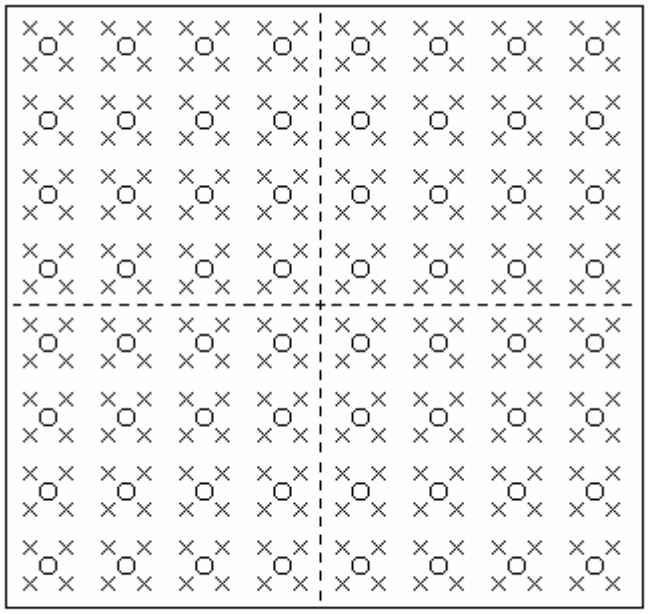


Рис.8.9. Структура трехкомпонентного макроблока (× - яркостная компонента, o - цветонесущие составляющие)

Над матрицей отсчетов производится дискретное косинусное преобразование. Полученные трансформанты квантуются, затем поступают в видеомультиплексный кодер, в котором происходит энтропийное кодирование. Одновременно с помощью деквантователя и обратного ДКП кадр восстанавливается и сохраняется в памяти вычислителя для использования его при межкадровом кодировании.

В блоке вычислителя производится оценка вектора движения и с его учетом осуществляется предсказание на основе ДИКМ отсчетов последующего кадра. Предсказание базируется на оценке движения путем сравнения всех макроблоков (только яркостных) текущего кадра с макроблоками соответствующих фрагментов изображения предыдущего кадра. Затем ошибка предсказания преобразуется в соответствии с процедурой ДКП, квантуется и после энтропийного кодирования включается в выходной поток. На видеомультиплексор поступает также сигнал вектора движения, Кроме этого, мультиплексор включает в выходной поток информацию о виде кодирования (внутри- или междукадровое), коэффициентах квантования, наличии фильтра подавления высокочастотных шумов.

Компрессор изображений видеосвязи может работать в одном из трех режимов:

-ДИКМ кодирование без компенсации движения (нулевой вектор движения);

-ДИКМ кодирование с ненулевым вектором движения;

-фильтрация блоков (по мере необходимости) встроенным фильтром для подавления высокочастотных шумов.

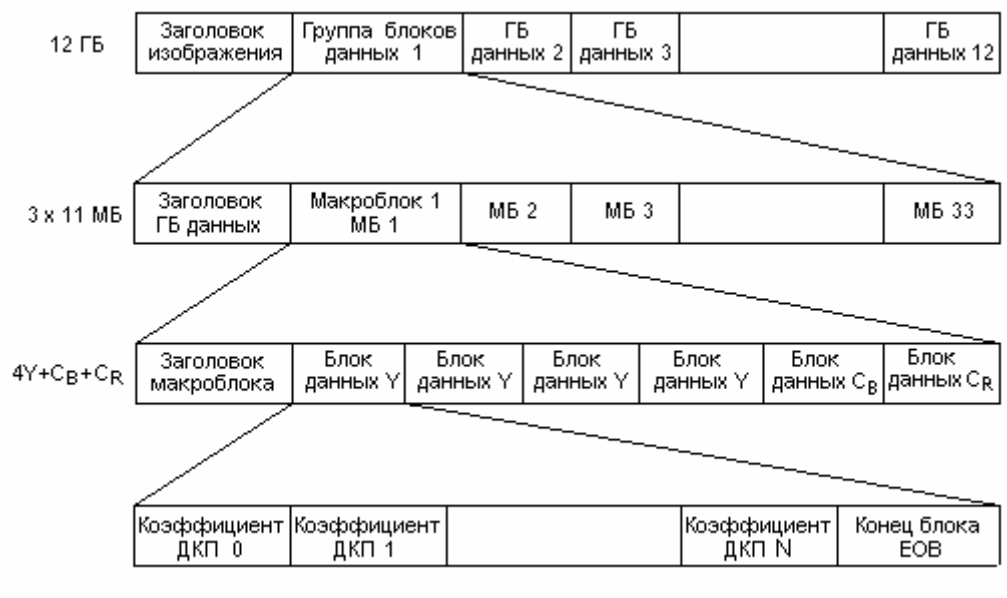


Рис.8.10. Иерархическая структура данных процедуры сжатия по стандарту Px64

Важнейшей характеристикой, определяющей однозначность декодирования сжатого изображения, является структура данных. Все структурные составляющие изображения могут быть представлены в виде четырехслойной иерархической структуры, изображенной на рис.8.10.

Верхним уровнем иерархии является изображение (кадр). Данные для каждого кадра состоят из заголовка изображения *(Picture Header),* за которым следуют группы блоков данных ГБ. Для формата *CIF* изображение включает 12 групповых блоков, а для *QCIF*- четыре. Заголовок содержит 20-битовую кодовую комбинацию начала кадра и другую служебную информацию (тип видеоформата, номер кадра и т.д.). Заголовок имеет фиксированную длину, а группы блоков - переменную.

Следующий уровень образует группа блоков. Каждая из групп в свою очередь содержит заголовок и 3×11=33 макроблоков (МБ). На этом уровне заголовок также состоит из фиксированного числа разрядов и включает 16-битовую стартовую комбинацию начала группы блоков и информацию о позиции ГБ, сведения о используемом квантовании и др.

Третий уровень иерархической структуры образует макроблок. Формат макроблока включает заголовок и информационную часть, состоящую из шести блоков ($4Y+C_B+C_R$). Длина блока является переменной. В заголовке располагается служебная информация, которая указывает тип макроблока, вид кодирования (внутри- или междукадровое), наличие оценки движения объекта в кадре, использование шумоподавляющего фильтра. Каждый блок заканчивается кодовой комбинацией EOB (конец блока).

Операции по восстановлению сжатого изображения производятся в декодере. Типовая структура декодера Px64 показана на рис.8.11.

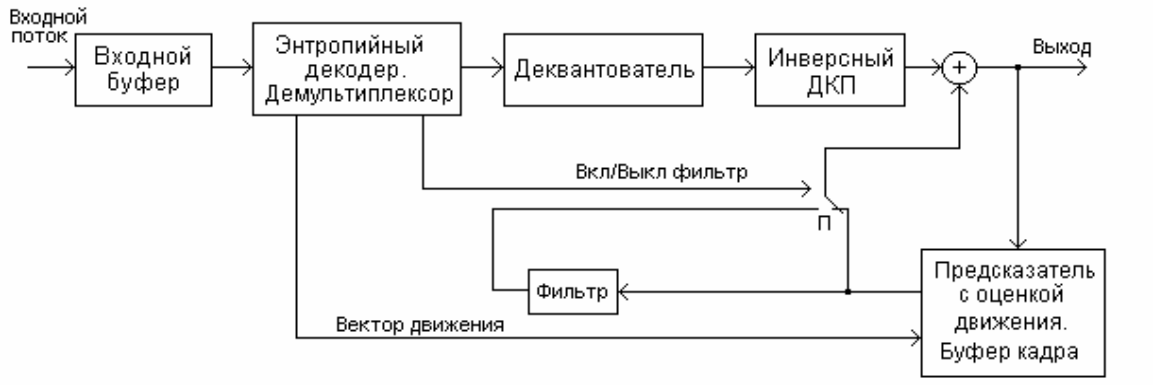


Рис.8.11. Структурная схема декодера Px64

На его вход поступает мультиплексированный цифровой поток данных, содержащий закодированные неравномерным кодом ошибки предсказания отсчетов фрагментов изображения, а также управляющую служебную информацию о параметрах кодирования, в том числе векторы движения объектов в кадре. Разделение кодовых комбинаций ошибок предсказания и служебной информации осуществляется

демультиплексором, который посылает информацию об ошибках предсказания на деквантователь, а служебные сообщения - на соответствующие функциональные блоки. Выделенный из входного потока вектор движения поступает в вычислитель, где происходит предсказание принимаемого кадра. К предсказанным отсчетам прибавляются ошибки предсказания и затем восстановленный кадр выдается потребителю. Предсказанные значения поступают на сумматор через фильтр или напрямую. Это зависит от того, использовался ли фильтр при кодировании изображения. Включение и отключение фильтра осуществляется управляющим сигналом, выделяемым демультиплексором из входного потока.

8.6. Стандартная процедура сжатия изображения MPEG

В связи с внедрением мультимедиа систем подвижные изображения становятся формой компьютерных данных, в которых текст объединяется с графикой и звуковым сопровождением, а по компьютерным сетям все чаще передаются сигналы видеоданных и связанные с ними аудиосигналы. В этих условиях стандартизация технологии сжатия подвижных изображений позволяет уменьшить высокую стоимость кодеков для видеокомпрессии и разрешить проблему совместимости программных и технических средств различных производителей. Поэтому аналогично группе *JPEG* международными организациями МОС и МККТТ была создана группа по разработке стандарта сжатия цветных подвижных изображений *MPEG (Moving Picture Experts Group)*. Деятельность *MPEG* охватывает сжатие не только видеоданных, но и компрессию связанных с ними звуковых сигналов, формирование сигналов аудио- и видеосинхронизации, мультиплексирование многочисленных битовых потоков различных источников.

Стандарты *MPEG* предназначены как для симметричных, так и для несимметричных приложений. Несимметричные характеризуются многократным использованием процесса декомпрессии, в то время как компрессия данных выполняется один раз в процессе производства носителей информации (например, *CD-ROM*). В симметричных приложениях процедура компрессии и декомпрессии применяется одинаково часто. Примером симметричных приложений являются системы телевидения, видеотелефонии, телеконференций и др.

В 1991 году опубликован стандарт *MPEG*-1 [27], ориентированный на использование в системах мультимедиа персональных компьютеров с целью воспроизведения изображений игровых программ и с учетом

использования односкоростных *CD-ROM*. При разработке стандарта также учитывалось, что при передаче данных по первичным каналам цифровой интегральной сети связи со скоростью около 1,5 Мбит/с должно обеспечиваться приемлемое качество восстанавливаемых изображений. Входное изображение для кодера *MPEG* представляется в формате *SIF (Source Input Format)* с разрешающей способностью 352×240 пикселов, который является четвертью стандартного телевизионного формата *CCIR* 601. Затем были опубликованы стандарты *MPEG*-2 и *MPEG*-3 [23]. Разработанные стандарты не отрицают друг друга, каждый из них рассчитан на работу в определенных условиях.

Стандарт *MPEG*-2 ориентирован на формат *CCIR* 601 для высококачественного воспроизведения изображений с разрешающей способностью 704×480 пикселов для *NTSC* телевизионных систем, и 704×576 — для системы *PAL*. Необходимая пропускная способность канала связи составляет 6 Мбит/с.

MPEG-3 разработан для использования в системах телевидения высокой четкости *HDTV*, обеспечивающего разрешающую способность 1920×1080 пикселов. Требуемая при таком качестве скорость передачи данных находится в пределах от 20 до 40 Мбит/с. Пока этот стандарт не применяется, а для *HDTV* используется *MPEG*-2. Работы над созданием *MPEG*-4 начались в 1993 г. Задачей стандарта является обеспечение приемлемого качества изображения с разрешающей способностью 174×144 пикселов и частоте 10 кадр/с при скорости передачи 64 кбит/с. Он предназначен для использования в системах мультимедийной и видеотелефонной связи, электронных газет, почты, игр. Стандарт предполагается принять в 1998 г. В табл. 8.5 приведены основные параметры стандартов *MPEG*.

Таблица 8.5

Формат	Разрешающая способность, пиксел	Частота кадров, Гц	Скорость передачи, Мбит/с	Стандарт
SIF	352 × 240	30	1,2 - 3	MPEG-1
CCIR 601	720 × 486	30	5 - 10	MPEG-2
EDTV	960 × 486	30	7 - 15	MPEG-2
HDTV	1920 × 1080	30	20 - 40	MPEG-2 (-3)
QCIF	174 × 144	10	0,064	MPEG-4

Обеспечение высокого качества восстанавливаемого изображения при ограничении на скорость передачи может быть обеспечено только при устранении временной и пространственной избыточности, характерных для подвижных сцен, в связи с чем целесообразно применять комбинирование внутрикадрового и межкадрового кодирования. Причем, необходимо оптимальное сочетание этих методов, т.к. в процессе обработки видеоданных требуется обеспечить произвольный доступ к элементам изображения, что проще всего реализуется при чисто внутрикадровом кодировании.

Для решения указанной проблемы разработчики стандарта предложили использовать методы предсказания и интерполяции с компенсацией движения объекта в кадре, а для внутрикадрового кодирования - хорошо зарекомендовавшее себя дискретно-косинусное преобразование (ДКП), подобное используемому в процедуре *JPEG* и последующее энтропийное кодирование.

Метод компенсации движения применяется как в причинном (чисто предсказательное кодирование), так и в беспричинном прогнозировании (интерполяционное кодирование). Полученная ошибка предсказания затем сжимается посредством ДКП. Сведения о движении фрагментов изображения мультиплексируются вместе с пространственной информацией, и после дополнительного сжатия путем неравномерного кодирования поступают в канал.

Суть межкадрового кодирования по процедуре MPEG состоит в том, что кодируемое изображение представляется тремя типами кадров:

I-кадры *(Intra),* которые берутся за основу и кодируются без обращения (ссылок) к другим кадрам; *I*-кадры позволяют осуществлять произвольный доступ к элементам изображения; в этих кадрах осуществляется ДКП кодирование, которое обеспечивает относительно низкую степень сжатия;

P-кадры *(Predictive),* закодированные (предсказанные) на основе предыдущих кадров; степень сжатия выше, чем в кадрах типа *I*;

B-кадры *(Bidirectionally predictive),* закодированные на основе предыдущего и последующего кадров, обеспечивают максимальную степень компрессии.

Во всех случаях при кодировании со ссылками используется компенсация движения, позволяющая повысить эффективность сжатия. В процессе кодирования формируется поток кадров (рис.8.12), в который через определенный интервал, в качестве опорных, вставляются *I*-кадры, а на их основе предсказываются изображения *P*- и *B*-кадров. Вначале, путем прямого предсказания, на базе предыдущего *I*-кадра вычисляются отсчеты *P*-кадра. Затем на основе *I*- и *P*-кадров прогнозируются кадры *B* (2-, 3- и 4-й).

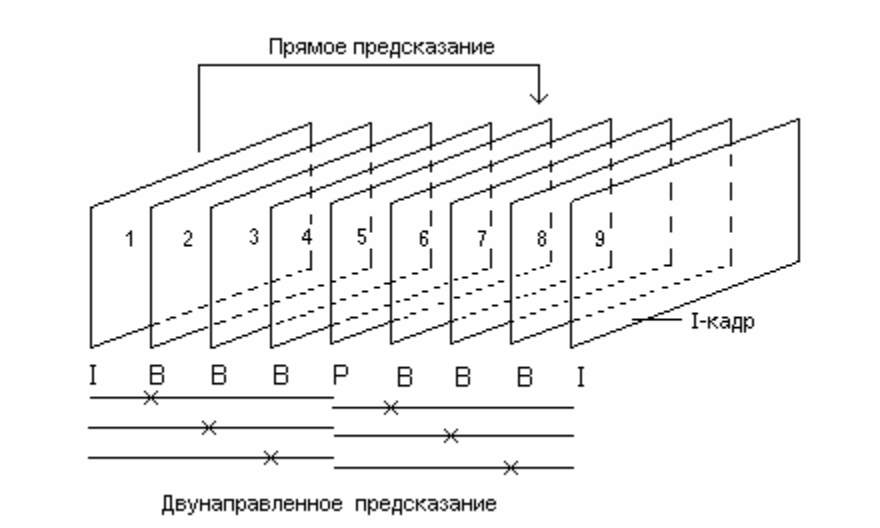


Рис.8.12. Поток кадров в системе сжатия видеоданных
по процедуре MPEG

Интерполяция 6-,7- и 8-го *B*-кадров производится на основе *P*-кадра (6-го) и последующего *I*-кадра. Интерполяция всех шести *B*-кадров проводится с учетом вектора движения объектов в кадре. Очевидно, что информация о параметрах движения должна быть получена заранее.

Организация изображений в *MPEG* очень гибкая и зависит от конкретной области и условий применения. Так, если есть необходимость для быстрого произвольного доступа к элементам изображения, количество *I*-кадров в последовательности может быть увеличено до тех пор, пока вся последовательность не превратится в поток *I*-кадров. В частности, очень эффективной может быть структура потока *(IBBPBBPBB) (IBBPBBPBB) ...* . Процесс кодирования *P*- и *B*-кадров включает оценку движения, в соответствии с которой находится наиболее сопоставимый блок в ссылочных кадрах. *P*-кадры определяются на основе прямого предсказания, в то время как кадры *B* вычисляются путем интерполяции (двунаправленного предсказания) предыдущего и последующего *P*- и *I*-кадров с учетом вектора движения. Предсказание с учетом вектора движения объекта осуществляется для фрагмента изображения размером 16×16 пикселов, называемого макроблоком. В зависимости от того, к какому типу кадра относятся прогнозируемые макроблоки, их аналогично подразделяют на внутренние *(Intra),* прямого предсказания *(Forward Predicted),* обратного предсказания (Backward Predicted) и усредненные *(Average).*Сведения о движении содержат информацию об одном векторе для макроблоков прямого и обратного предсказания и двух - для усредненных блоков (двунаправленного предсказания). Прогнозируемые значения пикселов для данного макроблока зависят от кадров, на которые осуществляется ссылка (предыдущий или последующий). Предсказание

для различных типов кадров в стандарте *MPEG* осуществляется по следующим формулам.

Внутрикадровое предсказание:

$$\overline{B}_1(i,j) = 128 .$$

Прямое предсказание:

$$\overline{B}_1(i,j) = \overline{B}_0(i + dx_{01}, j + dy_{01}) .$$

Обратное предсказание:

$$\overline{B}_1(i,j) = \overline{B}_2(i + dx_{21}, j + dy_{21}) .$$

Усреднение:

$$\overline{B}_1(i,j) = 0{,}5[\overline{B}_{01}(i + dx_{01}, j + dy_{01}) + \overline{B}_2(i + dx_{21}, j + dy_{21})] .$$

Здесь $\overline{B}_1(i,j)$ – предсказанное значение отсчета изображения 1-го кадра в точке с координатами *(i,j)*, [*i,j*] изменяются в пределах [-8,8]; dx_{01}, dy_{01} - компоненты вектора перемещения объекта в кадре 1 по отношению к кадру 0. Ошибка предсказания при этом равна

$$E(i,j) = B_1(i,j) - B_1(i,j)$$

для всех (i,j) = [-8,8].

В *MPEG* процедуре не определяется, каким образом должен быть вычислен вектор движения. Однако из-за блокового представления движения наиболее целесообразной является оценка движения, определяемая методом сопоставления блоков (см. п.8.3).

На рис. 8.13 показана схема типового *MPEG* кодирующего устройства, состоящего из двух ветвей. В обеих ветвях производится обработка *UYV* компонент сигнала, для чего на их входах установлены преобразователи *RGB*-сигнала в *UYV*.

В первой ветви осуществляется внутрикадровое кодирование, основанное на дискретно-косинусном преобразовании (ДКП) и квантовании трансформант. Во второй ветви производится кодирование ошибок предсказания фрагментов (блоков) *P*- и *B*-кадров по отношению к

предыдущему и последующему опорным кадрам и с учетом вектора движения.

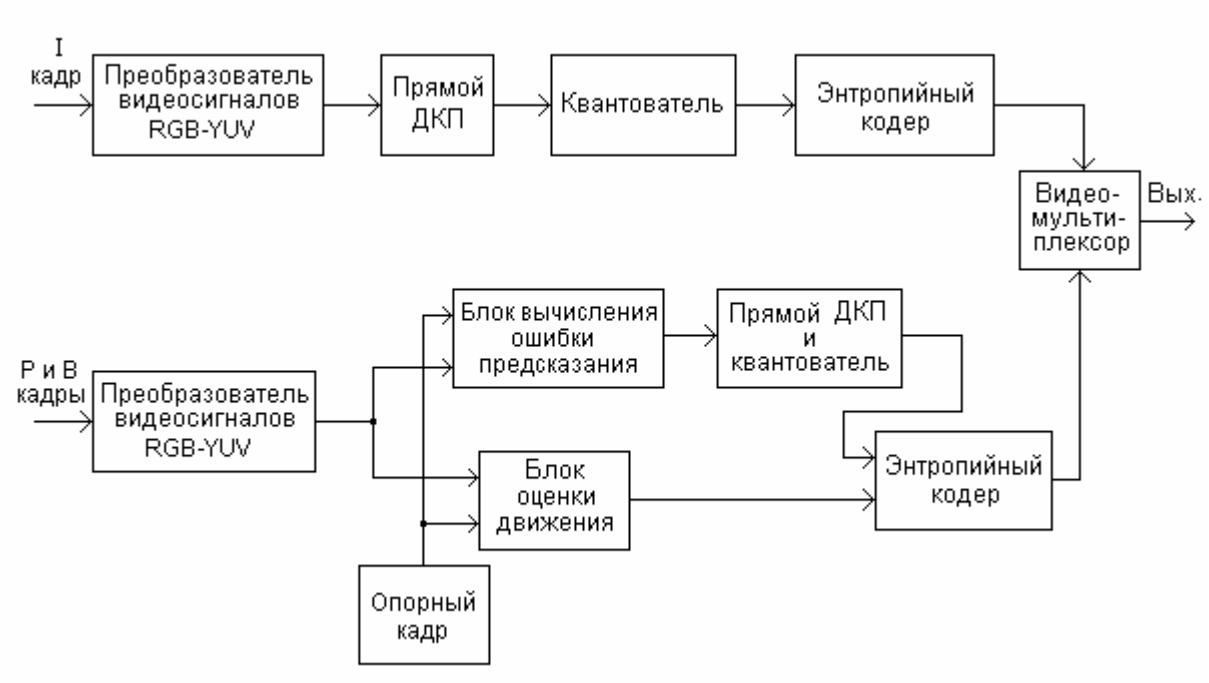


Рис.8.13. Структурная схема типового *MPEG*-кодера

Отсчеты фрагментов *I*-кадра должны быть проквантованы отлично от квантования ошибок предсказания или интерполяции. Это вызвано тем, что трансформанты блоков при внутрикадровом кодировании имеют примерно одинаковую энергию на всех частотах, в то время как блоки величин ошибок предсказания имеют преимущественно высокочастотные компоненты (в связи с достаточно точным предсказанием низкочастотных составляющих) и могут быть проквантованы более грубо. Проквантованные трансформанты ДКП кодируются неравномерным кодом в соответствующих энтропийных кодерах.

Схема типового *MPEG*-декодера изображена на рис.8.14. После восстановления кодовых слов из неравномерных кодовых комбинаций они разделяются демультиплексором на поток квантованных трансформант *I-, P-* и *B*-кадров, и служебную информацию, содержащую сведения о векторе движения, типе кадра и пр. С помощью переключателей П1...П3 осуществляется занесение в соответствующие буферы предыдущего и последующего кадров изображения, а также добавление к декодированным значениям ошибок предсказания отсчетов, предсказанных с учетом движения объектов кадров.

Реализация *MPEG*-кодеров и декодеров является чрезвычайно сложной задачей, которая решается на основе оптимального сочетания

аппаратных и программных средств. Практически каждый блок кодера и декодера (рис.8.13 и 8.14) реализуется на основе сверхбыстродействующего специализированного процессора.

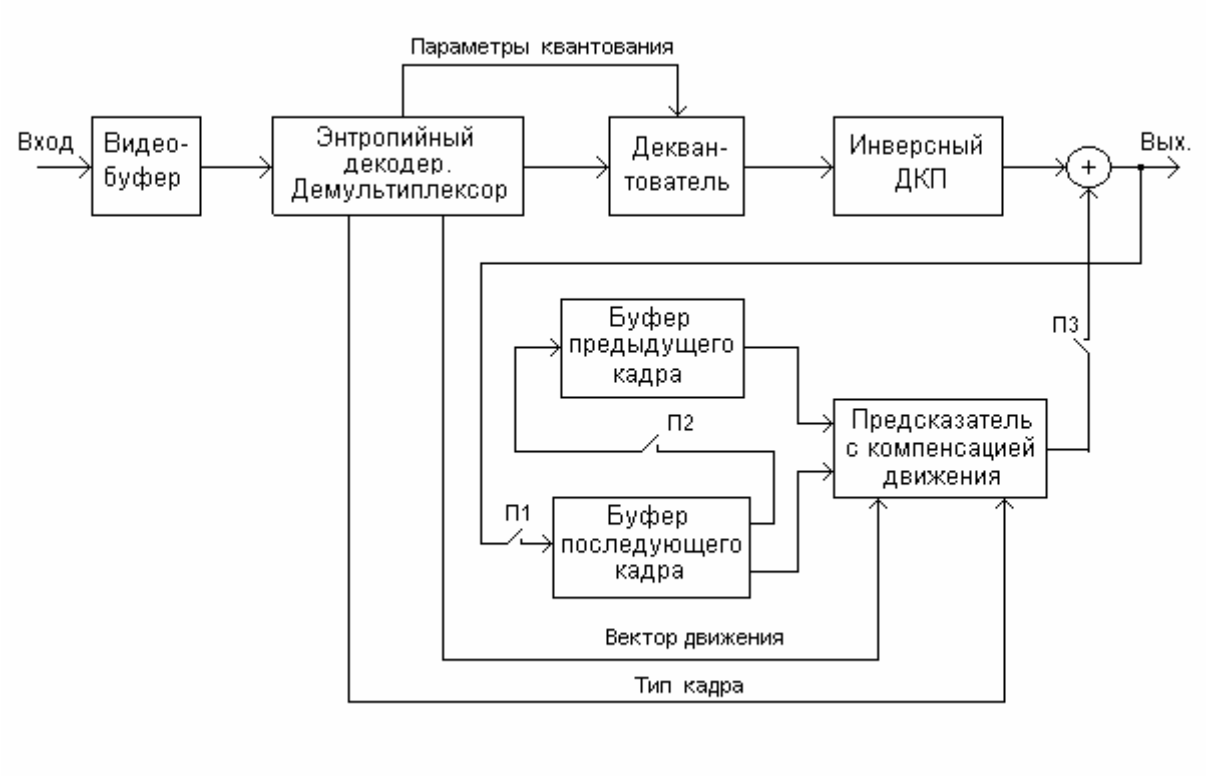


Рис.8.14. Структурная схема типового *MPEG*-декодера

В настоящее время многими фирмами освоен выпуск *MPEG*-карт, используемых в *PC*-мультимедиа системах. На рис.8.15 в качестве примера изображена схема полнофункционального кодера. выполненного на пяти БИС [25], осуществляющего сжатие подвижных изображений по стандарту *MPEG*-2. В состав кодера входит видеоинтерфейсный процессор (ВИП), три высокопроизводительных процессора оценки движения (ПОД), процессор обработки видеосигналов (ПОВС) и блоки видеопамяти. Кодер имеет две внутренние магистрали: шину межпроцессорной связи и шину ввода/вывода видеоданных. Связь *MPEG*-карты с шиной ПЭВМ обеспечивает видеоинтерфейсный процессор. Кроме этого ВИП готовит кадры изображения для сжатия и переформатирует их в последовательность, необходимую для оценки вектора движения.

Для выполнения всех функций *MPEG*-алгоритма в реальном времени требуется обработка данных со скоростью несколько миллиардов операций в секунду. Большинство из этих операций выполняется процессорами оценки движения. В приведенной схеме кодера три процессора оценки движения работают параллельно, сканируя видеокадры, отыскивая сопряженные блоки, формируют вектор,

характеризующий движение от одного кадра к другому. Один ПОД может производить 4,5 млрд. оп/с, обследуя поисковую область ±128 пикселей, при размерах сопоставляемых блоков 16×16 или 16×8 пикселей. Поиск сопряженных блоков занимает менее 10 мс. Каждый из спецпроцессоров кодера снабжен индивидуальной высокоскоростной видеопамятью, которая используется в качестве буфера на несколько кадров. Общий объем ОЗУ кодера равен 14 Мбайт.

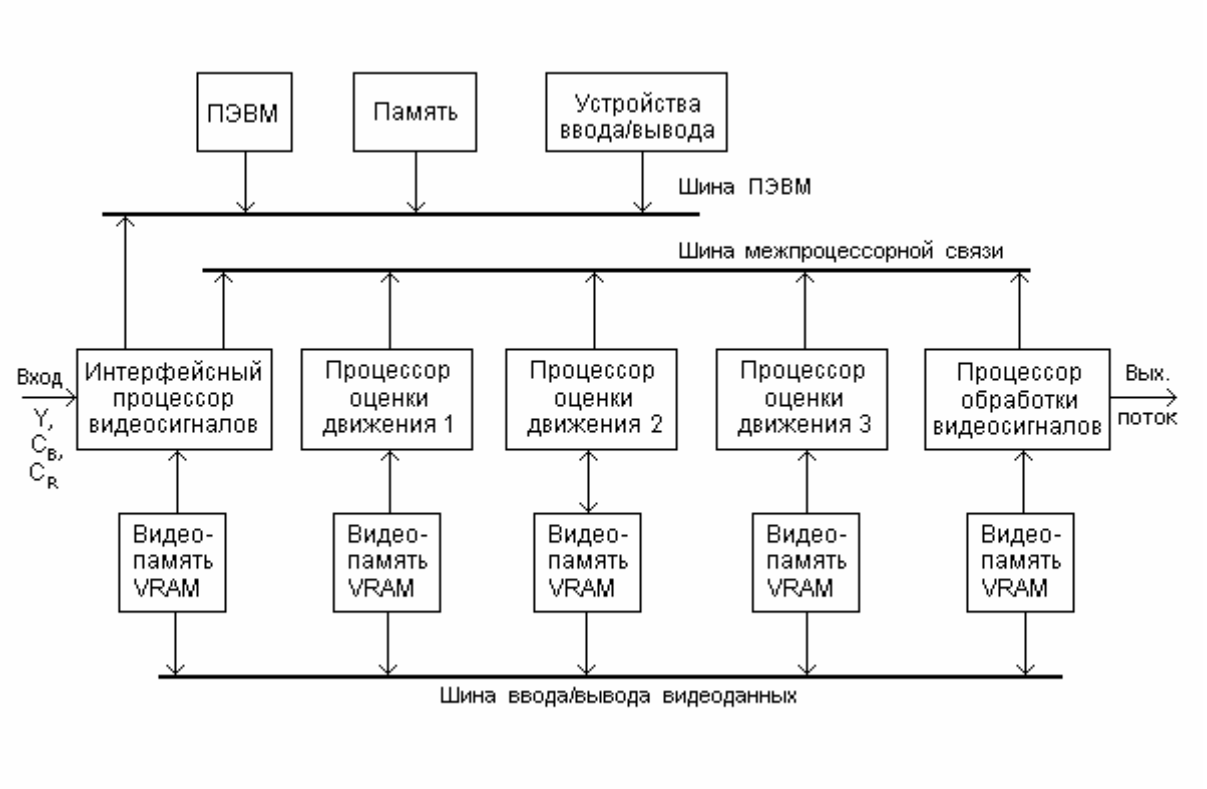


Рис.8.15. Схема полнофункционального кодера *MPEG*-2

Все три типа специализированных процессоров выполнены по *RISC*-архитектуре с 32-разрядной шиной данных. Программирование процессоров может осуществляться на языке высокого уровня С и С++, что существенно упрощает разработку прикладных программ. Большинство выполняемых процессорами функций могут быть запрограммированы пользователем, повышая тем самим гибкость системы и расширяя область применения *MPEG*-карты.

Контрольные вопросы к главе 8

1. С какой целью в телевидении применяют чересстрочную развертку?

2. Каким образом на экране монитора можно получить произвольный цвет или его оттенок?

3. В чем состоит отличие европейской иерархии цифровой интегральной сети связи (*ISDN*) от североамериканской и японской?

4. Какие группы (компоненты) сигналов используются для передачи цветных подвижных изображений и какая между ними существует взаимосвязь?

5. Объясните, что обозначают следующие соотношения: 4:2:2; 4:1:1; 2:1:1?

6. Как используются параметры движения объектов при предсказании и интерполяции кадров?

7. Как осуществляется интерполяция в случае наличия в кадре нескольких движущихся объектов?

8. Каким образом можно определить координаты смещения объекта по отношению к его положению в соседнем кадре?

9. Дайте сравнительную характеристику критериев согласования фрагментов (блоков) изображения?

10. Приведите пример наиболее неблагоприятного расположения вектора перемещения для различных процедур поиска?

11. В результате каких действий достигается максимальный коэффициент сжатия малоподвижных изображений в процедуре Px64?

12. Почему в качестве входного сигнала компрессора Px64 используются компоненты YC_BC_R?

13. Почему длина блока трансформант является переменной?

14. В чем состоит отличие процедуры *MPEG* от *JPEG*?

15. Объясните суть двунаправленного предсказания.

СПИСОК ЛИТЕРАТУРЫ

1. *Боровик А*. Технология сжатия данных // Компьютер.—1991.—N2(5).— С.52-53.
2. *Вирт Н*. Алгоритмы + структуры данных = программы. — М.: Мир, 1985. — 406 с.
3. *Галлагер Р.* Теория информации и надежная связь. — М.: Сов.радио, 1974, — 720 с.
4. *Дмитриев В.И*. Прикладная теория информации. — М.: Высшая школа, 1989. — 320 с.
5. *Ефимов В.М. и др*. Оценка эффективности некоторых алгоритмов сокращения избыточности при абсолютной точности воспроизведения // Автометрия.— 1991.—N6.— С. 93 - 97.
6. *Зубарев Ю.Б., Глориозов Г.Л.* Передача изображений. — М.: Радио и связь, 1989. — 332 с.
7. Иконика. Цифровая обработка видеоинформации. — М.: Наука, 1989. — 186 с.
8. *Кохманюк Д.* Сжатие данных: как это делается. Часть I и часть II// Index PRO.— 1992.— N1—С.18-29.— 1993.- N2— С.30-49.
9. *Криворучко В.О*. Сжатие цветных синтезированных изображений // Автометрия.— 1993.— N5.— С.58-63.
10. *Кричевский Р.Е*. Сжатие и поиск информации. — М.: Радио и связь, 1989. — 168 с.
11. *Лэнгсам Й., Огенстайн М., Тененбаум А*. Структуры данных для персональных ЭВМ. — М.: Мир, 1989. — 568 с.
12. Методы передачи изображений. Сокращение избыточности / Под ред. *У.К.Прэтта.* — М:. Радио и связь, 1983.—264 с.
13. Передача дискретных сообщений: Учебник для вузов / Под ред. *В.П.Шувалова.* — М.: Радио и связь, 1990. — 464 с.
14. *Рябко Б.Я.* Быстрый алгоритм адаптивного кодирования // Проблемы передачи информации.— 1990.— т.26.— Вып.4.— С.24-37.
15. Сжатие видеоинформации (Обзор) // ТИИЭР. — 1981. — Т. 69.—N3.- С.71-117.
16. *Фано Р*. Передача информации. Статистическая теория связи. — М.: Мир, 1965. — 438 с.

17. *Хамелл Р.Л.* Последовательная передача данных: Руководство для программиста. — М.: Мир, 1996. — 752 с.
18. *Харатишвили Н.Г.* Цифровое кодирование с предсказанием непрерывных сигналов. — М.: Радио и связь, 1986. —140 с.
19. *Хэмминг Р.В.* Теория кодирования и теория информации. —М.: Радио и связь, 1983. — 176 с.
20. Цифровое кодирование графики //ТИИЭР.—1980.—Т.68.— N7.—215 с.
21. *Цымбал В.П.* Теория информации и кодирование. — К.: Выща школа, 1992. — 263 с.
22. *Чернега В.С., Василенко В.А., Бондарев В.Н.* Расчет и проектирование технических средств обмена и передачи информации. —М.: Высшая школа, 1990. — 224 с.
23. *Чизбру Э.* MPEG — за и против // Компьютер+программы.— 1996. — N9. — С.22-25.
24. *Чэн Ш.-К.* Принципы кодирования систем визуальной информации.— М.: Мир, 1994. — 408 с.
25. *Bursky Dave*. Full-Feature MPEG-2 Encoder NEEds Just Five Logig Ics. // Electronic Design.— 1996.— N4.— P.137-140.
26. *Dance D.L., Pooch U.W.* An adaptive on line data compression System. // Computer Journal.— 1976. — Vol. 19.—N3.—P. 216-224.
27. *Furht Borko.* Multimedia Systems and Techniques. — Boston: Kluwer Academic Publishers, 1996.— P.325.
28. *Gallager R.G.* Variations on a theme by Huffman // IEEE Trans.on Inform.Theory.— 1978.— Vol. 24. — N6. — P. 668-674.
29. *Gonzalez M., Storer J.A.* Parallel algorithms for data compression // Journal of the ACM.— 1985.— Vol.32.— N2.— P. 344-373.
30. *Hankamer M.* A modified Huffman procedure with reduced memory requirement // IEEE Trans. Commun.— 1979.— Vol. 27.— N6.—P. 930-932.
31. *Held G.* Data compression: Techniques and applications. — New York: John Wiley and Sons, 1991. — P.301.
32. *Knuth D.E.* Dynamic Huffman coding // Journal of algorithms.— 1985.— N6.— P.163-180.
33. *Langton G.G.* An Introduction to Arithmetic Coding // IBM Journal of Research and Development. — 1984.— Vol.28.— N2.— P.135 -149.

34. *Lelewer D.A., Hirschberg D.S.* Data compression // ACM Computing Surveys.— 1987.— Vol.9.—N3.— P.261-296.
35. *McIntyre D.R., Pechura M.A.* Data compression using static Huffman code-decode tables // Journal of the ACM. — 1985.— Vol.28. — N6.— P. 612-616.
36. *Rissanen J. A* universal data compression system // IEEE Trans.on Inform.Theory, 1983.— Vol. 29.— N5. — P. 656 - 664.
37. *Nelson Mark.* Datenkomprimierung. Effiziente Algorithmen in C. — Hannover : Verlag Heinz Heise, 1993. — P. 475 .
38. *Rubin I.* Data compression for communication networks: The Delay-distortion function. // IEEE Trans. on Inform.Theory.— 1976.— Vol. 22.—N6.— P.655 - 665.
39. *Sayood Khalid*. Introduktion to Data Compression.— San Francisco: Morgan Kaufmann Publishers. Inc. — 1995.- P.453.
40. *Storer J.A*. Data compression: Methods and theory. — Computer science press.— 1988. — P.413.
41. *Tschernega V.S.* Methoden der dynamischen Datenkompression. Zuerich: ETH IFE. — 1994. — S.55 .
42. *Vitter J.S.* Design and analysis of dynamic Huffman coding // Journal of ACM.— 1987.— Vol.34.— N4. — P. 825 - 845.
43. *Wallace G.K*. The JPEG Still Picture Compression Standard // Communication of the ACM.— 1991.— Vol. 34.— N4.— P.31 - 44.
44. *Witten I.H., Neal R.M., Cleary J.G.* Arithmetic Coding for Data Compression // Communication of the ACM.— 1987.— Vol. 30.— N6.— P.520 - 540.
45. *Welch T.A*. A technique for high-perfomanse data compression // IEEE Computer.— 1984.— Vol.17.—N6.— P. 8-19.
46. *Ziv J.,Lempel A.* A universal algorithm for sequential data compression // IEEE Trans. on Inform.Theory. — 1977.— Vol. 23.— N3. — P. 337-343.
47. *Ziv J., Lempel A*. Compression of individual sequences via variable-rate coding // IEEE Trans.on Inform.Theory.— 1978.— Vol.24.—N5.— P.530-536.

ПРИЛОЖЕНИЯ

Приложение 1

Вероятности появления символов в английских текстах

Символ	Вероятность	Символ	Вероятность	Символ	Вероятность
Пробел	0,200	H	0,047	W	0,012
E	0,105	D	0,035	G	0,011
T	0,072	L	0,029	B	0,011
O	0,065	C	0,023	V	0,008
A	0,063	F	0,023	K	0,003
N	0,059	U	0,023	X	0,001
I	0,055	M	0,021	J	0,001
R	0,054	P	0,018	Q	0,001
S	0,052	Y	0,012	Z	0,001

Приложение 2

Вероятности появления символов в русских текстах

Символ	Вероятность	Символ	Вероятность	Символ	Вероятность
Пробел	0,175	К	0,028	Ч	0,012
О	0,090	М	0,026	Й	0,010
Е	0,072	Д	0,025	Х	0,009
А	0,062	П	0,023	Ж	0,007
И	0,062	У	0,021	Ю	0,006
Н	0,053	Я	0,018	Ш	0,006
Т	0,053	Ы	0,016	Ц	0,004
С	0,045	З	0,016	Щ	0,003
Р	0,040	Ь	0,014	Э	0,003
В	0,038	Б	0,014	Ф	0,002
Л	0,035	Г	0,013		

Приложение 3

Программа компрессии файлов методом LZW

```
program compress;
  uses crt, dos;
  {константы для hash-функции}
const  BITS  =13;          {разрядность кодов строк}
       TABLE_SIZE  =9029;       {max размер таблицы- больше 2^bits}
       HASHING_SHIFT =BITS-8;
       MAX_VALUE =1 shl BITS;  {2 в степени  BITS }
       MAX_CODE   =MAX_VALUE-2; {код должен быть BITS
                                              разрядным }
type
  compress_type=0..MAX_CODE+1; {описание типа; значения BITS
                                              битовые}
        { Запись из трех элементов:
        поле code_value - равно -1 если ячейка пуста, либо коду
        строки которая сохранена  как prefix_code+append_character в
        соответствующих полях записи.
        В записи хранятся: код строки (или -1), код ее левой части и код
         добавленного символа  }
par=record
      code_value : integer;
      prefix_code : word;
      append_character : byte;
    end;
massiv_par=array [0..TABLE_SIZE-1] of par; {таблица-массив записей}
 var
 {next_code-равен коду, который будет присвоен следующей строке,
  заносимой в таблицу;  character-символ-добавка, string_code-
  код строки; String_code+character соответствует
  prefix_code+append_character в таблице; Index-индекс записи,
  формируется функцией find_match из string_code и character}
   i, string_code : compress_type;
   next_code, index: word;
   character: byte;
   ch   : char;
   inn  : text;
   b,v  : file of char;
   out  : file of compress_type;
   c  : real;
```

```
   vbn : integer;
   ptr_table : ^massiv_par;
   count  : longint;     {счетчик считанных из файла символов}
   input_file, out_file:  pathstr;
    chas1, min1, sek1, sot1:   word; {переменные для хранения теку-}
    chas2, min2, sek2, sot2:   word; {щего времени начала и конца }
                                     { компрессии              }
  {-------------------------------------------}
  function Find_match(hash_prefix: integer;
     hash_character: word):  compress_type;
  var index, offset: integer;
  begin
    index:=hash_character shl HASHING_SHIFT;{расчет индекса index}
    index:=index xor hash_prefix;
    if index=0 then offset:=1                {смещение для случая коллизии}
    else offset:=TABLE_SIZE-index;
    while 1<2 do begin
                  {выход из программы, если в code_value  пусто,
                  или если code_value содержит код строки, а prefix_code и
                  append_character равны hash_prefix и hash_character
                  соответственно, т.е. запись занята аналогичной строкой}
      if ptr_table^[index].code_value=-1 then
                 begin
                    find_match:=index; exit;
                 end;
      if (ptr_table^[index].prefix_code=hash_prefix) and
        (hash_character=ptr_table^[index].append_character) then
               begin find_match:=index;
                exit;
               end;
      { коллизия (поле записи с таким индексом занято строкой, не
         аналогичной с данной)
       расчет другого индекса:}
      index:=index-offset;    {offset - псевдослучайное смещение }
      if index<0 then index:=index+TABLE_SIZE;
             end;
  end;
  {-------------------------------------------}
 procedure Time;          {процедура определения времени процесса}
                { Рассчитывается вычитанием времени начала процесса (chas1,
                min1, sek1, sot1) из времени окончания процесса (chas2, min2,
                sek2, sot2) }
```

```
 begin
     if sot2 < sot1 then
       begin
         sot2:=sot2+100;
         dec(sek2);
       end;
     if sek2 < sek1 then
       begin
         sek2:=sek2+60;
         dec(min2);
       end;
     if min2 < min1 then
       begin
         min2:=min2+60;
         dec(chas2);
       end;
   chas1:=chas2-chas1;
   min1:=  min2-min1;
   sek1 :=  sek2-sek1;
   sot1 :=  sot2-sot1;
   if chas1<>0 then write(chas1, 'час.');
   if min1 <>0 then write(min1, 'мин.');
   write(sek1,'.', sot1, 'сек.');
End;
 {-------------------------------------------}
 Begin
        {ввод имен файлов и открытие их}
   clrscr;
   writeln('-------Программа сжатия файлов----------версия 3.0--------');
   writeln;
 case ParamCount of
    0: begin
       writeln ('compress [ Имя_сжимаемого_файла [ Имя_файла-
                 результата]]');  writeln;
       write('Введите имя сжимаемого файла  (*.*) : ');
       readln(input_file);
       write('Введите имя файла-результата (*.lzw): ');
       readln(out_file); writeln;
       if (length(input_file)=0) or (length(out_file)=0) then halt(0);
       end;
    1: begin
       input_file:=ParamStr(1);
```

```
    write('Введите имя файла-результата (*.lzw):  ');
    readln(out_file); writeln;
    if (length(out_file)=0) then halt(0);
    end;
  2: begin
    input_file:=ParamStr(1);
    out_file:=ParamStr(2);
    end;
else
 begin writeln ('compress [ Имя_сжимаемого_файла [ Имя_файла-
                  результата]]'); halt(0);
 end;
 end;
 assign(inn,input_file);
 {$i-}
  reset(inn);
 {$i+}
 if ioresult<>0 then
   begin
    writeln('Ошибка открытия файла ', input_file);
    halt(0);
   end;
  assign(out, out_file);
 {$i-}
  rewrite(out);
 {$i+}
 if ioresult<>0 then
   begin
    writeln('Ошибка открытия файла ', out_file);
    halt(0);
   end;
   writeln(' Сжимается файл ', input_file);
              {выделение места для массива в "куче", теперь обращение к
              его записям возможно только через указатель}
  New(ptr_table);
              {получить текущее время начала компрессии }
  GetTime(chas1, min1, sek1, sot1);
{-----------------------------------------------------}
              {исходная установка}
next_code:=256; count:=0;
i:=0;
while i<=TABLE_SIZE do
```

```
    begin
        ptr_table^[i].code_value:=-1;
        i:=i+1;
    end;
i:=0;
{-----------------------------------------------------}
read(inn,ch);                    {ввод символа}
string_code:=ord(ch);            {string_code=коду символа}
WHILE not eof(inn) do            {пока входной файл не пуст, делать:}
 begin
         {ввод символа и присвоение его кода переменной character}
  read(inn, ch); character:=ord(ch);
  inc(count);
  if (count mod 1000)=0 then write('*');
          {рассчитать индекс из строки string_code  и символа character}
  index:=find_match(string_code,character);
                  {если поля записи заняты, то string_code=коду строки,
                   которой занята запись}
  if ptr_table^[index].code_value<>-1 then
         string_code:=ptr_table^[index].code_value
  else begin
        {иначе: если таблица строк не заполнена, то занести строку
          и ее код в таблицу}
      if next_code<=MAX_CODE then
    begin
        ptr_table^[index].code_value:=next_code;
        next_code:=next_code+1;
        ptr_table^[index].prefix_code:=string_code;
        ptr_table^[index].append_character:=character;
    end;
      write(out, string_code); {вывести код string_code в вых. файл}
      string_code:=character;
      end;
 end;
 write(out, string_code); {вывести код string_code в вых.файл}
 i:=MAX_VALUE-1;{ BITS двоичных единий - признак конца }
 write(out, i); {занести в вых. файл признак конца сжатых данных}
 i:=0;
 write(out,i);
 writeln;
{-----------------------------------------------------------}
                  {получить текущее время окончания компрессии }
```

```
     GetTime(chas2, min2, sek2, sot2);
     writeln;        {рассчитать время окончания компрессии }
     write('Файл сжат за : '); Time; writeln;
     close(inn);     {закрыть файлы}
     close(out);
   dispose(ptr_table);     {освободить память в "куче"}
                            {расчет коэффициента сжатия}
assign(v, input_file);
assign(b, out_file);
reset(v); reset(b);
 c:=filesize(b)/filesize(v);
 c:=(1-c)*100;
 vbn:=round(c);          {округление до целого}
       {печать размеров файлов до и после сжатия и коэффициент сжатия}
writeln('Размер файла до сжатия   : ', filesize(v),' b');
writeln('Размер файла после сжатия: ', filesize(b),' b');
 if vbn>0 then
       begin
         vbn:=100-vbn;
         writeln('Процент сжатия : ', vbn, ' %');
       end
             {если файл стал больше, чем был- печать сообщения об этом}
 else begin
       writeln;
       writeln('      Невозможно сжать файл ', input_file); writeln;
       writeln('  Размер файла-результата больше, чем размер исходного
                    файла.');
       write('  Его уничтожить? (Y/N)');
       ch:=readkey;
       if (ch='y') or (ch='Y') then erase(out);
     end;
writeln; writeln('Для выхода нажмите любую клавишу...');  writeln;
writeln('------------------------------------------------------------');
repeat until keypressed;
close(v); close(b);
  End.
```

Приложение 4

Программа декомпрессии файлов методом LZW

```
program Expand;
  uses crt, dos;
            {константы для hash-функции}
 const  BITS =13;                    {разрядность кодов строк }
        TABLE_SIZE =9029;            {max размер таблицы- больше 2^bits}
        HASHING_SHIFT=BITS-8;
        MAX_VALUE =1 shl BITS;
        MAX_CODE  =MAX_VALUE-2;
 type
  compress_type=0..MAX_CODE+1;      {коды BITS разрядные }
                { Запись из двух полей:
                каждая строка сохранена как prefix_code+append_character}
  par=record
      prefix_code : word;
      append_character : byte;
    end;
 massiv_par=array [0..TABLE_SIZE] of par;
 var
  b,v  : file of char;
  count, c: word;
     {next_code-равен коду, который будет присвоен следующей строке,
     заносимой в таблицу; new_code-код символа только введенного
     из входного файла; old_code-код строки, сформированный ранее}
  new_code,old_code: compress_type;
                                  {string_ - массив для декодирования строки}
  string_   : array [0..4000] of word;          {массив, имитирующий стек}
  ind_buf,i : integer;                           {индекс массива string_}
  s : char;
  inn : file of compress_type;
  out : text;
  ptr_table : ^massiv_par;                       {указатель на запись par}
  next_code, character_: word;
  input_file, out_file : pathstr;
   chas1, min1, sek1, sot1: word;  {переменные для хранения текущего }
   chas2, min2, sek2, sot2: word;  {времени начала и конца компрессии }
                    {-----------------------------------------}
      {процедура делит Code на prefix_code и append_character,
```

```
        затем append_character выводит в массив string_;
  процесс повторяется до тех пор, пока code не станет <256
  (пока он код строки, а не символа); когда он станет <256,
  он заносится в массив и - выход. }
  procedure decode_string(code: compress_type);
  var i: integer;
begin
    i:=1;
    while code>255 do begin
      string_[ind_buf]:=ptr_table^[code].append_character;
      code:=ptr_table^[code].prefix_code;
      inc(ind_buf);
      if ind_buf>=4000 then
  begin
        writeln('Переполнение буфера стека...');  {переполнение буфера
                                                        стека}
        halt(0);
  end;
end;
     string_[ind_buf]:=code;
  end;
  {-----------------------------------------}
 procedure Time;   {процедура определения времени процесса}
                  { Рассчитывается вычитанием времени начала процесса
                    (chas1, min1, sek1, sot1) из времени окончания
                      процесса (chas2, min2, sek2, sot2) }
  begin
      if sot2 < sot1 then
        begin
          sot2:=sot2+100;
          dec(sek2);
        end;
      if sek2 < sek1 then
        begin
          sek2:=sek2+60;
          dec(min2);
        end;
      if min2 < min1 then
        begin
          min2:=min2+60;
          dec(chas2);
        end;
```

```
    chas1:=chas2-chas1;
    min1 :=  min2-min1;
    sek1 :=  sek2-sek1;
    sot1 :=  sot2-sot1;
    if chas1<>0 then write(chas1, 'час.');
    if min1 <>0 then write(min1, 'мин.');
    write(sek1,'.', sot1, 'сек.');
  End;
   {-----------------------------------------}
   Begin
                        {Ввод имен файлов и открытие их}
   clrscr;
   writeln('-------Программа декомпрессии файлов-----версия 3.0--------');
   writeln;
   case ParamCount of
      0: begin
         writeln ('expand [ Имя_декомпрессируемого_файла [ Имя_файла-
результата]]'); writeln;
         write('Введите имя декомпрессируемого файла  (*.lzw) : ');
         readln(input_file);
         write('Введите имя файла-результата  (*.*): ');
         readln(out_file); writeln;
         if (length(input_file)=0) or (length(out_file)=0) then halt(0);
         end;
      1: begin
         input_file:=ParamStr(1);
         write('Введите имя файла-результата  (*.*): ');
         readln(out_file); writeln;
         if (length(out_file)=0) then halt(0);
         end;
      2: begin
         input_file:=ParamStr(1);
         out_file:=ParamStr(2);
         end;
    else
         begin
    writeln ('expand [ Имя_декомпрессируемого_файла
                          [ Имя_файла-результата]]');
              writeln;
              halt(0);
         end;
    end;
```

```
 count:=1;
   assign(inn, input_file);
   {$i-}
    reset(inn);
   {$i+}
   if ioresult<>0 then
     begin
       writeln('Ошибка открытия файла ', input_file);
       halt(0);
     end;
    assign(out, out_file);
   {$i-}
    rewrite(out);
   {$i+}
   if ioresult<>0 then
     begin
       writeln('Ошибка открытия файла ', out_file);
       halt(0);
     end;
    writeln('   Декомпессия файла ', input_file);
         {выделение места для массива в "куче", теперь обращение к
         его записям возможно только через указатель}
    New(ptr_table);
    {получить  текущее время начала декомпрессии }
    GetTime(chas1, min1, sek1, sot1);
  {-----------------------------------------------------}
  next_code:=256;   {исходная установка next_code}
  read(inn,old_code); {ввод кода old_code}
  character_:=old_code;
  s:=chr(old_code);    {преобразование кода в символ и печать его}
  write(out, s);
repeat
{Выполнять ниже перечисленные действия, пока на входе есть коды
                          из сжатого файла}
 begin
  read(inn, new_code); nc(count);          {ввод кода new_code}
  if (count mod 1000)=0 then write('*');
  IF new_code<>MAX_VALUE-1 THEN
   begin
    ind_buf:=1;
  if new_code>=next_code then
    begin
```

```
      {если new_code не определен в таблице, то декодировать в строку код
                              old_code}
       decode_string(old_code);
       string_[0]:=character_; {по алгоритму: string=string+character
       т.е. добавить character к строке справа, т.к. в массиве  строка
       сохранена в обратном порядке, то мы  ставим его крайним слева}
     end
    else
        begin
                  {если код new_code определен в таблице,
                   то декодировать его в строку}
         decode_string(new_code);
         string_[0]:=0;     {т.к. в этом случае ничего к строке
                             добавлять не надо, пишем туда 0}
        end;
   i:= ind_buf;
                       {вывести содержимое массива string_, содержащего
                        декодированную строку, в обратном порядке}
   while ind_buf>0 do
       begin
    s:=chr(string_[ind_buf]);
    write(out,s);
    dec(ind_buf);
       end;
   if string_[0]<>0 then
       begin       {если=0? то этого элемента нет  и его выводить не надо}
    s:=chr(string_[0]);
    write(out, s);
       end;
   character_:=string_[i];      {character равен первому символу
                  декодированной строки (последнему занятому в массиве)}
   if next_code<MAX_CODE then
       begin       {если в таблице строк еще есть место, то
                    вносим в нее строку в виде old_code+character_}
     ptr_table^[next_code].prefix_code:=old_code;
     ptr_table^[next_code].append_character:=character_;
     inc(next_code);      {расчет кода для строки , которую надо будет
                           вносить в таблицу строки}
       end;
   old_code:=new_code;
END
  ELSE i:=30000; {если new_code=BITS единиц, то конец декомпрессии}
```

```
  end;
  until (i=30000);
   {-------------------------------------------------------------}
    writeln;          {получить текущее время окончания декомпрессии }
    GetTime(chas2, min2, sek2, sot2);
                       {определить время декомпрессии}
    write('Время декомпрессии файла :'); Time; writeln;
    close(inn);       {закрыть файлы}
    close(out); writeln;
    dispose(ptr_table);   {освободить память в куче}
                          {вывод объема файлов до и после декомпрессии}
 assign(v, input_file);
 assign(b, out_file);
  reset(v); reset(b);
writeln('Размер файла до декомпрессии    : ', filesize(v),' b');
writeln('Размер файла после декомпрессии : ', filesize(b),' b');
writeln; writeln('Для выхода нажмите любую клавишу...');
writeln;
writeln('-------------------------------------------------------------');
repeat until keypressed;
close(v); close(b);
  End.
```

Приложение П5

Таблица модифицированного кода Хаффмена

Длина серии	Завершающие кодовые слова	
	Белые серии	Черные серии
0	0011 0101	0000 1110 11
1	0001 11	010
2	0111	11
3	1000	10
4	1011	011
5	1100	0011
6	1110	0010
7	1111	00011
8	10011	000101
9	10100	000100
10	00111	0000100
11	01000	0000101
12	0010 00	0000 111
13	0000 11	0000 0110
14	1101 00	0000 0111
15	1101 01	0000 1100 0
16	1010 10	0000 0101 11
17	1010 11	0000 0110 00
18	0100 111	0000 0010 00
19	0001 100	0000 1100 111
20	0001 000	0000 1101 000
21	0010 111	0000 1101 100
22	0000 011	0000 1101 111
23	0000 100	0000 1101 000
24	0101 000	0000 0010 111
25	0101 011	0000 0011 000
26	0010 011	0000 1100 1010
27	0100 100	0000 1100 1011
28	0011 000	0000 1100 1100
29	0000 0010	0000 1100 1101
30	0000 0011	0000 0110 1000
31	0001 1010	0000 0110 1001
32	0001 1011	0000 0110 1010
33	0001 0010	0000 0110 1011

Приложение П5 (продолжение)

Длина серий	Завершающие кодовые слова	
	Белые серии	Черные серии
34	0001 0011	0000 1101 0010
35	0001 0100	0000 1101 0011
36	0001 0101	0000 1101 0100
37	0001 0110	0000 1101 0101
38	0001 0111	0000 1101 0110
39	0010 1000	0000 1101 0111
40	0010 1001	0000 0110 1100
41	0010 1010	0000 0110 1101
42	0010 1011	0000 1101 1010
43	0010 1100	0000 1101 1011
44	0010 1101	0000 0101 0100
45	0000 0100	0000 0101 0101
46	0000 0101	0000 0101 0110
47	0000 1010	0000 0101 0111
48	0000 1011	0000 0110 0100
49	0101 0010	0000 0110 0101
50	0101 0011	0000 0101 0010
51	0101 0100	0000 0101 0011
52	0101 0101	0000 0010 0100
53	0010 0100	0000 0011 0111
54	0010 0101	0000 0011 1000
55	0101 1000	0000 0010 0111
56	0101 1001	0000 0010 1000
57	0101 1010	0000 0101 1000
58	0101 1011	0000 0101 1001
59	0100 1010	0000 0010 1011
60	0100 1011	0000 0010 1100
61	0011 0010	0000 0101 1010
62	0011 0011	0000 0110 0110
63	0011 0100	0000 0110 0111

Приложение 5 (продолжение)

Длина серий	Основные кодовые слова	
64	1101 1	0000 0011 11
128	1001 0	0000 1100 1000
192	0101 11	0000 1100 1001
256	0110 111	0000 0101 1011
320	0011 0110	0000 0011 0011
384	0011 0111	0000 0011 0100
448	0110 0100	0000 0000 0101
512	0110 0101	0000 0011 0110 0
576	0110 1000	0000 0011 0110 1
640	0110 0111	0000 0010 0101 0
704	0110 0110 0	0000 0010 0101 1
768	0110 0110 1	0000 0010 0110 0
832	0110 1001 0	0000 0110 0110 1
896	0110 1001 1	0000 0011 1001 0
960	0110 1010 0	0000 0011 1001 1
1024	0110 1010 1	0000 0011 1010 0
1088	0110 1011 0	0000 0011 1010 1
1152	0110 1011 1	0000 0011 1011 0
1216	0110 1100 0	0000 0011 1011 1
1280	0110 1100 1	0000 0010 1001 0
1344	0110 1101 0	0000 0010 1001 1
1408	0110 1101 1	0000 0010 1010 0
1472	0100 1100 0	0000 0010 1010 1
1536	0100 1100 1	0000 0010 1101 0
1600	0100 1101 0	0000 0010 1101 1
1664	0110 00	0000 0011 0010 0
1728	0100 1101 1	0000 0011 0010 1
EOL	0000 0000 0001	0000 0000 001

ОГЛАВЛЕНИЕ