В.В. Семенюк

# ЭКОНОМНОЕ КОДИРОВАНИЕ ДИСКРЕТНОЙ ИНФОРМАЦИИ

**Санкт-Петербург**

**2001**

УДК 621.391:519.726

**Семенюк В. В.** Экономное кодирование дискретной информации. – СПб.: СПбГИТМО (ТУ), 2001. – 115 с.

Монография посвящена систематическому обзору методов и алгоритмов экономного кодирования без потерь последовательной дискретной информации. Подробно рассматриваются идеи, лежащие в основе большинства существующих алгоритмических решений, а также наиболее важные детали их практических реализаций. Проводится качественное и количественное сравнение различных подходов.

Книга адресована программистам-практикам, работающим в сфере информационных технологий, и людям, интересующимся прикладными аспектами теории информации. Она может быть рекомендована студентам и преподавателям в качестве дополнительного материала по курсам «Технологии программирования», «Теория информации», «Теория кодирования».

ISBN 5–7577–0076–9

# Оглавление

## Предисловие

Организованная в 1991 году кафедра компьютерных технологий была создана для проведения специального образовательного проекта. Целью этого проекта было формирование нового поколения исследователей, имеющих гармоничную синтетическую подготовку в области математики, физики, информатики и компьютерных технологий. По мнению инициаторов проекта, именно такие специалисты будут наиболее востребованными при развитии высоких технологий в наступившем столетии. Десятилетняя история проведения проекта представила много фактов, свидетельствующих о правильности выбранного пути. В частности хорошей традицией кафедры стало раннее приобщение студентов к научной работе и, что является достаточно редко встречающимся явлением, к написанию оригинальных книг и подготовке объемистых переводов по вопросам современных направлений развития компьютерных технологий.

Предлагаемая читателю книга студента шестого курса кафедры компьютерных технологий Владимира Семенюка занимает достойное место в ряду этих научных трудов молодых ученых. Автор начал над ней работать при написании бакалаврской работы, получившей высокие оценки ряда ведущих российских специалистов, которые предложили развить ее. Тема увлекла Владимира и он продолжил работать над ней при написании магистерской диссертации.

Итогом этой трехлетней работы стала предлагаемая вниманию читателя книга. Актуальность освещаемой автором темы экономного кодирования дискретной информации не требует дополнительных доказательств. Хотелось бы отметить, что в отечественной литературе практически отсутствуют книги, посвященные систематическим исследованиям в данной области.

Монография, по нашему мнению, может быть весьма полезной студентам вузов, обучающихся по компьютерным специальностям.

Хотелось бы надеяться на то, что первый уверенный шаг Владимира Семенюка в науку окажется не последним, и пожелать молодому исследователю успехов в его дальнейших исследованиях в области теоретической информатики.

Ректор СПбГИТМО(ТУ),
профессор В.Н. Васильев

Декан факультета
информационных технологий и программирования,
профессор В.Г. Парфенов

## Выражение благодарности

Данная работа не могла быть написана без помощи и поддержки многих людей, чье участие оказалось для меня по-настоящему неоценимым.

Это, прежде всего, профессор, ректор СПбГИТМО (ТУ) В. Н. Васильев и профессор кафедры компьютерных технологий СПбГИТМО (ТУ) В. Г. Парфенов, которые поддержали меня в желании написать этот труд и всячески способствовали его публикации. Именно благодаря им монография наконец увидела свет.

Я глубоко признателен руководителю моей предыдущей научной работы А. О. Сергееву и ее рецензенту С. Е. Столяру за особое отношение к моим исследованиями в области прикладной теории информации. Это отношение во многом повлияло на мое решение продолжить научную работу, что в конечном итоге привело в написанию данной монографии.

Искренне благодарю профессора СПбГУ И. В. Романовского и профессора СПбГУАП Н. А. Шехунову за интерес, проявленный к моей работе. Их советы и замечания заставили меня критически пересмотреть ряд глав, что, на мой взгляд, позволило в значительной степени улучшить содержание и стиль изложения.

Особую благодарность выражаю доктору техн. наук В. П. Дворковичу и кандидату физ.-мат. наук А. В. Дворковичу, взявших на себя труд ознакомиться с предварительной версией монографии. Их положительный отзыв явился дополнительным стимулом к ее доработке и подготовке к печати.

Еще раз выражаю благодарность и глубокую признательность всем этим замечательным людям.

В. В. Семенюк

## Введение

В связи с непрерывным увеличением объема информации, накапливаемой во всех областях человеческой деятельности, все большее значение приобретают технологии ее компактного представления. Под компактным представлением информации следует понимать некоторую форму хранения информации, экономную с точки зрения занимаемого объема информационного носителя. Информационным носителем может являться любая физическая субстанция, будь то человеческий мозг, память вычислительной машины или простой лист бумаги. Мы привыкли говорить, что носитель информации содержит информацию как некий реальный объект. В действительности же информация есть не более чем атрибут информационного носителя. Дело в том, что информация сама по себе не существует отдельно от носителя. Информация есть порождение информационного носителя: получить информацию – означает «прочитать» состояние ее носителя. Таким образом, информация является ничем иным, как абстракцией, и надо четко осознавать, что, когда речь идет о методах компактного представления информации, на самом деле подразумеваются способы изменения состояния не самой информации, а, скорее, ее носителя. Способ получения компактного представления информации носит название *экономное кодирование*.

Хотя первый алгоритм экономного кодирования был предложен С. Морзе в 1838 году (имеется в виду знаменитая азбука Морзе), отправной точкой развития технологий компактного представления информации все же принято считать сороковые годы двадцатого века – период зарождения теории информации. Долгое время экономное кодирование как таковое не являлось самоцелью. Основные результаты носили в большинстве своем чисто теоретический характер и почти не имели практической ценности (в качестве примера можно привести всем известные теоремы Шеннона об энтропии). С появлением в конце 70-х годов эффективных практических схем экономного кодирования интерес к данному разделу теории информации стал заметно расти, однако невысокий уровень развития вычислительной техники часто серьезно препятствовал разработке многих перспективных направлений. Некоторые из этих направлений были в полной мере исследованы только во второй половине 80-х годов.

В последнее время раздел теории информации, касающийся методов экономного кодирования, претерпевает серьезные изменения. В первую очередь это связано со значительным увеличением объемов теоретических и прикладных исследований. Стоит особо выделить тот факт, что, наряду с усовершенствованием существующих методик, предлагаются и абсолютно новые решения. Одним из таких решений стал алгоритм Барроуза-Уилера, публикация которого в 1994 году ознаменовала своеобразную революцию в

отдельно взятой области экономного кодирования дискретной информации, долгое время считавшейся в достаточной мере изученной. В свою очередь, нельзя не отметить и явно наметившуюся тенденцию к переоценке некоторых «старых» подходов, по ряду причин полностью или частично отвергнутых ранее.

Теория экономного кодирования объединяет в себе несколько различных направлений. В рамках данной теории методы принято разделять на методы экономного кодирования информации *без потерь* (*lossless*) и методы экономного кодирования информации *с потерями* (*lossy*)[1]. Как следует из названий, обработка информации методами первой группы не ведет к информационным потерям, тогда как использование методов второй группы сопряжено с такими потерями.

Методы последней группы применяются к информации, содержащей отдельные несущественные составляющие (при определенных условиях эти составляющие могут быть частично или полностью удалены из информации). Одним из примеров такой информации является звуковая информация, предназначенная для прослушивания. Человеческий слух, как известно, воспринимает весьма ограниченный спектр частот, поэтому уменьшение числа хранимых звуковых гармоник не всегда приводит к ухудшению качества звука. То же самое касается и воспринимаемых зрительных образов. Пренебрежение неразличимыми человеческим зрением особенностями пространственных и временных изменений изображений, позволяет существенно сократить объем представления видеоинформации. Принципиально иной пример использования методов экономного кодирования с потерями – передача телеграфных сообщений. С целью увеличения скорости передачи информации из сообщений удаляются слова (предлоги, знаки препинания и т. д.), не несущие значимой смысловой нагрузки.

Как видно, в основе методов экономного кодирования информации с потерями лежит учет специфических особенностей восприятия информации. Следует отметить, что среди всех методов экономного кодирования эти методы являются наиболее эффективными.

Данная монография посвящена систематическому изложению методов другой группы – методов экономного кодирования информации без потерь. Если методы экономного кодирования с потерями могут быть ориентированы на произвольный тип информации, методы экономного кодирования без потерь используются исключительно для обработки дискретной информации. Под дискретностью информации следует понимать возможность ее представления в виде совокупности конечного или счетного числа нату-

[1] В научной литературе можно также встретить термины *кодирование без искажений* и *кодирование с искажениями*.

ральных чисел. Достаточно наглядно такое определение иллюстрирует корзина, содержащая пронумерованные бильярдные шары. Совокупность номеров можно рассматривать как некоторое дискретное информационное сообщение. Другим ярким примером дискретной информации является текстовая информация. Как нетрудно понять, любое текстовое сообщение отвечает условию дискретности, так как буквам алфавита всегда можно поставить в соответствие некоторые натуральные числа.

Приведенные примеры достаточно показательны, ибо они весьма отчетливо отражают два принципиально различных вида дискретной информации. Сравнивая эти примеры, несложно заметить, что текстовая информация имеет одну важную особенность, которая отсутствует у информации, описывающей содержимое корзины с шарами: текстовое сообщение представимо не просто как совокупность натуральных чисел, а как их последовательность. В данном случае мы имеем дело с особым видом информации. О такой информации говорят как об обладающей *последовательной структурой*.

Информация, обладающая последовательной структурой, является одним из наиболее распространенных, если не самым распространенным видом информации. Последовательная структура есть следствие природы самой информации – процесс рождения информации с последовательной структурой имеет естественную или условную хронологию. Естественная хронология свойственна информации, рождение которой последовательно во времени. Любые две выборки такой информации являются естественно упорядоченными, поскольку они могут быть четко разделены по времени своего появления. В отличие от естественной хронологии, условная хронология подразумевает некий искусственный выбор последовательности рождения информации. В данном случае речь идет не о реальной структуре информации, а лишь об информационной модели. Выбор последовательности рождения информации чаще всего обусловливается удобством нашего восприятия этой информации.

Теория информации не различает естественную и условную хронологию рождения; для теории информации важен только сам факт наличия последовательности в появлении информации. Любое дискретное информационное сообщение, обладающее последовательной структурой, на языке теории информации представляется как некоторая последовательность натуральных чисел, причем выбор конкретных значений этих чисел в большинстве случаев является произвольным[1]. Таким образом, проблема экономного кодировании дискретной информации с последовательной структурой в

[1] Для текстовой информации, например, это означает отсутствие четкого правила нумерации букв алфавита.

рамках теории информации фактически сводится к проблеме экономного кодирования натуральных последовательностей.

Как вы, вероятно, заметили, автор пытается несколько сузить круг вопросов, связанных с экономным кодированием дискретной информации, ограничиваясь проблемой обработки последовательной информации. И это не случайно. Современная теория экономного кодирования информации без потерь почти не выходит за рамки указанной проблематики. С одной стороны, это по-прежнему связано с недостаточным интересом к данному разделу теории информации, с другой стороны, на сегодня отсутствует практическая необходимости в расширении круга затрагиваемых вопросов – информация, с которой приходится сталкиваться на практике, в большинстве своем имеет явно выраженную последовательную структуру.

Итак, монографию следовало бы скорее озаглавить так: «Экономное кодирование последовательной дискретной информации». Если учесть тот факт, что наиболее распространенным типом информации с последовательной структурой является текстовая информация, название можно было бы еще более конкретизировать: «Экономное кодирование текстовой информации». Однако последнее означало бы отказ от универсальности при рассмотрении методов экономного кодирования, безотносительности к конкретному типу информации. Текстовая информация является весьма специфическим типом информации, обладающим рядом особенностей, не свойственных последовательной дискретной информации в ее первоначальном определении. Автор сознательно не идет на такое ограничение, оставаясь приверженцем более универсального подхода.

# 1. Информационные закономерности

Как получить компактное представление той или иной информации? Чтобы ответить на этот вопрос, следует понять, почему информация вообще может быть представлена более или менее компактно. Причина, как ни странно, кроется в самой информации, а точнее – в информационных закономерностях.

Рассмотрим отрывок из известного стихотворения Сергея Есенина:

*Лицом к лицу*
*Лица не увидать.*
*Большое видится на расстоянье.*
*Когда кипит морская гладь,*
*Корабль в плачевном состоянье.*

Преобразуем его в несколько более строгую форму:

*Лицом_к_лицу↓Лица_не_увидать.↓Большое_видится_на_расстоянье.*
*↓Когда_кипит_морская_гладь,↓Корабль_в_плачевном_состоянье.*

(В данном случае символ «↓» обозначает смену строки. Пробел заменяется символом «_».) В результате отрывок стихотворения превращается в информационное сообщение, обладающее некоторыми заметными особенностями. Во-первых, символ «_» встречается чаще других символов. Во-вторых, после символа «↓» всегда следует заглавная буква. В-третьих, в сообщении присутствуют одинаковые символьные сочетания, например, «*Лиц*», «*иц*», «*стоянье*».

С точки зрения теории информации между этими тремя особенностями (закономерностями) нет существенного различия. Информационный источник, для которого указанное сообщение является типичным, обладает свойствами, легко описываемыми на вероятностном языке. Тот факт, что пробел встречается чаще других символов, математически можно выразить следующим образом: вероятность появления символа «_» на выходе информационного источника больше вероятности появления любого другого символа. Неизбежное появление заглавной буквы после символа «↓» означает равенство единице условной вероятности появления заглавной буквы вслед за символом «↓». Третья особенность может быть представлена на вероятностном языке по-разному. С одной стороны, мы можем говорить о более вероятном появлении вышеперечисленных сочетаний символов в сравнении с другими символьными сочетаниями. С другой стороны, очевидно, сущест-

вует зависимость между символами, входящими в указанные сочетания: так, за символом «и» очень часто следует символ «ц».

Некоторые информационные закономерности выражены в менее явной форме. Предположим, что приведенный отрывок стихотворения хранится в файле в 256-символьном представлении. Очевидно, что в данном случае алфавит используется только частично. Действительно, представление отрывка стихотворения содержит 29 различных символов. Для его записи в файле дополнительно требуется символ конца строки «EOL». Таким образом, из 256 символов алфавита реально используются только 30 символов! Данная особенность также описывается с привлечением понятия вероятности: неиспользуемые лексемы информационного алфавита появляются на выходе источника информации с вероятностью, равной нулю, тогда как используемые – с ненулевой вероятностью.

Информационные закономерности свойственны самым различным типам информации и наиболее отчетливо проявляются в информации, обладающей ярко выраженной смысловой нагрузкой, каковой, например, является текстовая информация, большей частью состоящая из слов некоторого языка[1]. Любой метод экономного кодирования так или иначе учитывает информационные закономерности. Различие между методами состоит только в способе учета этих закономерностей.

Как получить компактное представление выбранного отрывка стихотворения, используя его особенности? Самый простой метод в нашей ситуации – это ограничение информационного алфавита. Для хранения одного символа 256-символьного алфавита в памяти компьютера требуется один байт (8 битов). Несложно понять, что алфавит русского языка с учетом встречающихся в тексте[2] заглавных букв и специальных символов, таких как пробел или запятая, допускает и 6-битовое представление информации (возможно использование $2^6 = 64$ различных символов). Однако, как оказывается, и такое представление является избыточным. Действительно, так как в записи отрывка стихотворения реально используются только 30 символов алфавита, для представления одного символа можно ограничиться всего 5 битами (5 битов позволяют хранить один из $2^5 = 32$ различных символов). Для 5-битового представления отрывка требуется $5 \cdot 122 = 610$ битов (122 символа $= 118$ обычных символов + 4 символа конца строки), тогда как, например, 8-битовое представление того же отрывка требует $8 \cdot 122 = 976$ битов.

Выигрыш от использования более компактного представления составляет 366 битов, то есть чуть менее 46 байтов. Много это или мало? Как во-

[1] Правильнее было бы называть такую информацию *языковой*.

[2] К примеру, в тексте вряд ли может встретиться заглавная буква «Ъ».

обще можно оценить эффективность того или иного информационного представления? Существует два способа оценки: относительный и абсолютный.

Первый способ имеет прямое отношение к понятию *сжатие информации*. Сжатием информации называется уменьшение объема ее представления путем преобразования исходного представления в новое, более компактное представление в метрике информационного носителя. Отношение объема нового представления информации к объему ее исходного представления носит название *коэффициент сжатия*. Коэффициент сжатия в нашем случае, если считать исходное представление информации 8-битовым, равен $5/8 = 0.625$ или $62.5\%$.

Для абсолютной оценки эффективности информационного представления используется величина, равная количеству информации данного представления, приходящемуся в среднем на один информационный символ. В рассматриваемом примере эффективность представления (в дальнейшем будет также использоваться термин *качество представления*) составляет 5 бит/сим.

Мы рассмотрели простейший метод экономного кодирования – *метод ограниченного алфавита*. Применим ли он на практике? Оказывается, что в том виде, в котором метод приводится выше, это невозможно. Дело в том, что информационное представление, полученное с использованием данного метода, не может быть правильно интерпретировано (декодировано) без дополнительной информации. Корректная интерпретация представления требует знания параметров кодирования. Пытаясь восстановить информационное сообщение в случае, когда эти параметры неизвестны, мы почти наверняка обрекаем себя на неудачу. Предположив, к примеру, что используется 6-битовое представление информации, мы не только не восстановим исходное информационное сообщение, но и, по-видимому, не сможем установить сам факт ошибочного декодирования. Таким образом, вместе с информационным представлением необходимо хранить исчерпывающие данные, описывающие способ, которым это представление было получено. В методе ограниченного алфавита эти данные должны содержать количество различных символов, задействованных в представлении информации, и сами эти символы. В рассмотренном примере описание способа кодирования занимает 31 байт при условии, что число символов в ограниченном алфавите может быть представлено одним байтом (30 байтов занимает перечисление символов ограниченного алфавита). Объем представления с учетом описания системы кодирования равен 858 битам $(610 + 8 \cdot 31)$.

Главным недостатком метода ограниченного алфавита является его малая применимость. Предположим, что в кодируемом информационном сообщении количество различных символов исходного 256-символьного ал-

фавита равно 129, причем символ с номером 129 встречается только один раз. Наличие уже одного такого символа делает рассмотренный метод неэффективным, так как для представления 129 символов требуется столько же битов информации, сколько требуется для представления 256 символов исходного алфавита. Можно придумать множество способов, позволяющих расширить область применения метода на подобные особые случаи, однако во многих ситуациях метод так и останется неприменимым. Ясно, что необходим подход, более универсальный по сравнению с описанным. Очевидный недостаток метода ограниченного алфавита состоит в том, что для каждого символа метод допускает только два состояния: символ либо участвует в описании информации, либо не участвует, и никак не учитывается то, что одни символы могут встречаться чаще других. Более эффективный метод кодирования должен базироваться на учете статистических особенностей кодируемой информации. Интуитивно понятно, что при использовании такого подхода «129-й символ» уже не будет иметь решающего значения.

Чтобы пояснить эту идею, давайте попытаемся представить себе некоторую гипотетическую систему кодирования, в которой разным символам соответствуют коды разной длины. В методе ограниченного алфавита длина кода для всех символов была одинаковой, то есть мы имели дело с так называемым *равномерным кодом*. Новая система кодирования должна включать в себя совершенно иной тип кодов – *коды переменной длины*[1].

В чем же состоит преимущество новой системы кодирования? Оказывается, что использование кодов переменной длины позволяет учесть статистические особенности появления символов в информационном сообщении. Часто встречающимся символам целесообразно ставить в соответствие короткие коды, а символам, встречающимся реже, – длинные коды. Полученное таким образом информационное представление может оказаться существенно более эффективным по сравнению с представлением, полученным с применением равномерного кода.

Итак, предположим, что существует способ кодирования, позволяющий использовать кодовые комбинации переменной длины. Зададимся вопросом: как должна выглядеть зависимость длины кода от вероятности появления кодируемого символа в информационном сообщении? Из теории, основы которой были заложены К. Шенноном в работе [8], следует, что в системе представления информации[2] с основанием $m$ (при $m = 2$ это двоичная или бинарная система) символу, вероятность появления которого равна $p$, оптимально и, что особенно важно, теоретически допустимо ставить в соответст-

[1] Данный термин применим только к совокупности кодов, но не к отдельным кодам.

[2] Более привычное название – система счисления.

вие код длины $-\log_m p$[1]. Это означает, например, что символы 128-символьного набора ASCII, появляющиеся в потоке информации с равной вероятностью, в двоичной системе представления целесообразно кодировать $-\log_2(1/128) = 7$ битами.

Приведенная выше формула лежит в основе теории экономного кодирования. Ее следствием является ранее высказанное положение о том, что эффективность представления информации целиком и полностью зависит от свойств самой информации. Для количественной оценки этих свойств Шеннон вводит две тесно связанные между собой характеристики: *энтропия* (*entropy*) и *избыточность* (*redundancy*).

Энтропия характеризует среднюю неопределенность (случайность) информационного сообщения. Пусть $a_1, \ldots, a_N$ – символы некоторого информационного алфавита $A$ мощности $N$, а $p(a_1), \ldots, p(a_N)$ – априорные вероятности их появления в некоторый момент времени на выходе информационного источника $S$ (при определенных оговорках в качестве источника информации может рассматриваться файл). Величина энтропии появления различных символов информационного алфавита на выходе источника $S$ при заданном вероятностном распределении (в данном случае также применим термин *величина энтропии вероятностного распределения*) вычисляется по формуле:

$$H(S) = -\sum_{i=1}^{N} p(a_i) \cdot \log_m p(a_i).$$

Единица измерения энтропии зависит от основания системы представления информации $m$. Например, при $m = 2$ энтропия измеряется в битах, а при $m = 10$ – в дитах[2]. Величина энтропии соответствует количеству информации, содержащемуся в среднем в одном информационном символе. Чем меньше вероятности появления различных символов на выходе информационного источника отличаются от вероятности их появления при равномерном распределении, тем больше величина энтропии. В случае, когда символы появляются равновероятно, энтропия максимальна и равна

$$H(S)_{\max} = \log_m N.$$

[1] Приведенный результат выглядит достаточно странно, если учесть тот факт, что код в дискретной системе представления всегда имеет целую длину. Тем не менее, как будет показано в п. 3.2, этот результат является практически достижимым.

[2] Дит - единица десятичного представления информации.

Избыточность есть понятие, обратное понятию энтропии. Численно избыточность определяется как

$$R(S) = H(S)_{\max} - H(S)^{1}.$$

По аналогии с качественным определением энтропии избыточность можно рассматривать как меру определенности (неслучайности) информационного сообщения.

Как непосредственно следует из приведенных формул, энтропия и избыточность появления символов на выходе источника информации в каждый конкретный момент времени позволяют оценить среднюю эффективность информационного представления символа, порождаемого источником в данный момент. Безусловно, более содержательной была бы оценка эффективности представления символа, поступающего с выхода источника информации в произвольный момент времени. В данном случае принято говорить об энтропии и избыточности информационного источника. Попытаемся понять, в чем же состоит различие между этими парами понятий. Пусть имеется двоичный источник информации, который порождает символ «1» вслед за символом «0» с вероятностью 1/2, а символ «0» вслед за символом «1» – с вероятностью 1/3. Какова средняя неопределенность появления символа на выходе такого источника? Энтропия появления символов при условии, что предыдущим символом, порожденным источником, был символ «0», согласно приведенной выше формуле, равна $-(1/2 \cdot \log_2(1/2) + 1/2 \cdot \log_2(1/2)) = 1$ бит, а энтропия появления символов при условии, что предыдущим символом, порожденным источником, был символ «1», равна $-(1/3 \cdot \log_2(1/3) + 2/3 \cdot \log_2(2/3)) \approx 0.92$ бита. Как видно, источник может находится в одном из двух состояний, каждому из которых соответствует определенная величина энтропии. Для нахождения энтропии источника нам, очевидно, следует усреднить эти энтропии с учетом вероятностей нахождения источника в каждом из состояний[2].

В теории вероятностей доказывается теорема[3] о том, что ансамбль условных вероятностей переходов системы из состояния в состояние при определенных условиях однозначно определяет вероятность нахождения системы в том или ином состоянии, бесконечно удаленном по числу переходов

---

[1] Если быть точнее, данная формула определяет абсолютную избыточность. Помимо абсолютной, вводится также относительная избыточность, которая вычисляется по формуле $R(S) = 1 - \frac{H(S)}{H(S)_{\max}}$.

[2] В общем случае такое усреднение возможно далеко не всегда.

[3] Имеется в виду эргодическая теорема А. А. Маркова-старшего.

от некоторого известного начального состояния. Применяя эту теорему к рассматриваемому примеру, можно заключить, что вероятность появления каждого бинарного символа на выходе указанного информационного источника в произвольный момент времени определена и не зависит от того, какой символ был порожден источником в далеком прошлом. Вероятность появления символа «0» в произвольный, достаточно удаленный от начального момент времени равна 2/5, а «1» – 3/5 (расчет данных величин здесь сознательно опущен). Теперь мы можем легко вычислить энтропию источника информации, произведя соответствующее усреднение:

$$H(S) = 1 \cdot 2/5 + 0.92 \cdot 3/5 \approx 0.95 \text{ бита.}$$

Рассмотрим более общую ситуацию. Предположим, что информационный источник *S* может находиться в одном из *T* состояний. Обозначим вероятность перехода из состояния $i$ в состояние $j$ как $p_{i,j}$. Если существует предел $P^{\infty} = \prod_{\infty} P$, где $P$ – матрица с компонентами $p_{i,j}$, величина энтропии источника *S* может быть вычислена по формуле:

$$H(S) = -\sum_{i=1}^{T} p_i \sum_{j=1}^{T} p_{i,j} \cdot \log_m p_{i,j} .$$

Здесь $p_i$ – вероятность нахождения источника *S* в $i$-ом состоянии ($p_i$ есть значение $i$-го элемента любой строки матрицы $P^{\infty}$), а $m$ – основание системы представления информации.

Множество состояний информационного источника в совокупности с ансамблем переходных вероятностей принято называть *моделью состояний* или *марковской моделью*. Для определения реальной энтропии и избыточности источника информации в рамках этой модели необходимо правильно подобрать ее структуру и параметры, которые должны в полной мере соответствовать данному источнику. К сожалению, на практике это далеко не всегда возможно. Во-первых, параметры информационных источников, как правило, заранее неизвестны, и их определение подчас требует приложения немалых усилий. Во-вторых, далеко не все информационные источники могут быть описаны с использованием модели состояний.

В случае, когда модель состояний оказывается непригодной для описания источника информации, целесообразно прибегнуть к помощи других моделей. (К примеру, можно воспользоваться моделью из метода ограниченного алфавита, представляющей собой сокращенный набор символов, задействованных в информационном представлении.) Существование раз-

личных информационных моделей значительно расширяет наши возможности в описании информационных источников, что делает эти источники хорошо предсказуемыми. Почему, спрашивается, так важно уметь правильно предсказывать поведение информационных источников? Ответ очевиден: чем лучше мы можем предсказать поведение источника, тем эффективнее мы можем его обработать. Кодирование всегда сопряжено с построением модели кодируемого источника. Любой метод экономного кодирования из тех, с которыми нам предстоит познакомиться в дальнейшем, имеет в своей основе такую модель. Процедура экономного кодирования, таким образом, включает в себя как бы два этапа: этап моделирования и этап кодирования. Первый этап – описание поведения источника в рамках выбранной модели. Второй этап – генерация кода на основе данного описания.

Моделирование и кодирование в рамках некоторого отдельно взятого метода или алгоритма неразрывно связаны между собой. В методе ограниченного алфавита, имеющем в своей основе достаточно тривиальную модель, используется и наиболее простой метод кодирования – генерация равномерного кода. Применение более сложных информационных моделей требует соответственно реализации более сложных механизмов кодирования. Речь, как правило, идет об использовании кодов переменной длины.

При создании метода экономного кодирования мы обязаны решить три основные проблемы: проблему построения информационной модели, проблему генерации кода и проблему хранения описания информационной модели, то есть фактически уже упоминавшуюся выше проблему хранения описания способа кодирования. Все эти проблемы заслуживают детального рассмотрения, чему будет посвящена большая часть последующего изложения. В главах 2, 3, 4 приведены некоторые общие замечания касательно отдельных аспектов этих проблем. Глава 5 посвящена рассмотрению возможных путей решения данных проблем на примере существующих методов экономного кодирования.

## 2. Марковские источники

В предыдущей главе мы рассмотрели понятия энтропии и избыточности информационного источника. Как мы выяснили, для получения объективной оценки степени неопределенности информации, поступающей с выхода того или иного информационного источника, необходимо учитывать вероятностные зависимости между его состояниями. Применительно к символьной информации наличие таких зависимостей тождественно существованию межсимвольной корреляции, сущность которой заключается во взаимосвязи между символами в информационном сообщении. Термин

«межсимвольная корреляция» тесно связан с понятиями *марковская цепь* и *марковский источник*.

Пусть имеется информационный источник *S*, порождающий различные символы с вероятностями, зависящими от символов, порожденных источником в предыдущие моменты времени. Такой информационный источник называется *источником с памятью*, так как его текущее состояние как бы «помнит» о его предыдущих состояниях. Символы, поступающие с выхода источника с памятью, часто образуют так называемые *марковские цепи*. Если в информационной последовательности вероятность появления символа определяется только одним предыдущим символом, про такую последовательность говорят, что она образует *марковскую цепь первого порядка* (*простая* или *односвязная марковская цепь*). При наличии вероятностной зависимости появления символа от *K* предыдущих символов говорят, что информационная последовательность образует *марковскую цепь K-го порядка* (*K-связная марковская цепь*). Информационный источник, порождающий марковскую цепь *K*-го порядка, принято называть *марковским источником K-го порядка*. Данное определение можно распространить и на случай источника, не обладающего памятью, назвав такой источник *марковским источником нулевого порядка*.

Особенности марковских источников описываются на языке условных вероятностей. Рассмотрим марковский источник *K*-го порядка, с выхода которого поступают символы информационного алфавита *A*. Вероятность появления символа $s_i$ после цепочки символов $s_{i-K}, \ldots, s_{i-1}$ на выходе такого источника определяется как условная вероятность $p(s_i \mid s_{i-K}, \ldots, s_{i-1})$. Величина энтропии марковского источника вычисляется по формуле:

$$H(S) = -\sum_{i_1=1}^{N} \ldots \sum_{i_{K+1}=1}^{N} p(a_{i_1}, \ldots, a_{i_{K+1}}) \cdot \log_m p(a_{i_{K+1}} \mid a_{i_1}, \ldots, a_{i_K}).$$

Здесь по-прежнему $a_1, \ldots, a_N$ – символы алфавита $A$ мощности $N$, а $m$ – основание системы представления информации. Заметим, что данная формула является частным случаем приведенной в предыдущей главе формулы для энтропии информационного источника с конечным числом состояний.

Марковский источник есть не что иное, как модель рождения информации. В связи с этим возникает естественный вопрос – насколько данная модель отвечает особенностям реальных информационных источников? Попытаемся ответить на этот вопрос применительно к одному из наиболее распространенных типов информации – текстовой информации.

Вряд ли вызовет сомнения тот факт, что соседние символы в текстовой информации в некоторой степени взаимосвязаны между собой. Обосновы-

вается это хотя бы тем, что по началу какого-либо слова мы часто способны угадать само слово, то есть конец слова в вероятностном смысле зависит от его начала. Таким образом, источник, порождающий текстовую информацию, определенно является источником с памятью. Остается определить порядок этой памяти. Безусловно, этот порядок не фиксирован и меняется в весьма широких пределах. Тем не менее можно утверждать, что в среднем для каждой разновидности текстовой информации порядок памяти остается примерно постоянным. Из сказанного следует, что источник текстовой информации не является марковским с фиксированным порядком памяти, хотя при достаточно грубых допущениях его все же можно считать таковым.

Сделанный вывод касается не только текстовой информации, но и многих других типов информации. Модель марковского источника, как правило, в целом не соответствует свойствам реальных информационных источников. Несмотря на это, модель марковского источника весьма активно привлекается на практике для описания всевозможных информационных процессов. Методы экономного кодирования ориентируются в основном именно на источники с конечной памятью, причем нередко предполагается, что порядок памяти остается неизменным. Такие ограничения, с одной стороны, позволяют упростить практические реализации этих методов, но, с другой стороны, вполне могут стать причиной потерь в их эффективности. Выбор оптимального решения в каждом конкретном случае должен опираться на соотношение между выигрышем в скорости получения информационного представления и возможным снижением качества этого представления. Чаще всего выбор делается в пользу первого фактора, так как за счет незначительных потерь в качестве информационного представления, как правило, можно достигнуть многократного увеличения производительности кодирования. По-видимому, именно в силу последнего рассмотренная в данной главе модель рождения информации получила в технологиях экономного кодирования достаточно широкое распространение.

## 3. Проблема генерации кода

Согласно приведенной в гл. 1 формуле, оптимальная длина кода символа определяется вероятностью его появления на выходе источника информации. В реальных задачах мы обычно не знаем эту вероятность, однако можем получить ее оценку на основе некоторой информационной модели. В данной главе нас не будут интересовать способы вычисления вероятностных оценок; мы займемся изучением методик генерации систем кодов переменной длины по уже заданным вероятностным распределениям. Такие системы обычно получают либо с использованием *префиксного кодирования*, либо с использованием *арифметического кодирования*.

Префиксное кодирование является простейшим методом генерации кодов переменной длины. Коды, получаемые с использованием префиксного кодирования, имеют целую длину в единицах представления информации. Преимущество префиксного кодирования состоит в его простоте и высокой производительности, а недостаток – в невозможности создавать коды нецелой длины, что, естественно, отрицательно сказывается на эффективности кодирования. По понятным причинам[1] наибольшие потери в эффективности при использовании префиксного кодирования наблюдаются в случае частой генерации кодов, теоретически оптимальная длина которых близка к нулю.

В отличие от префиксного кодирования, арифметическое кодирование позволяет генерировать коды[2] как целой, так и нецелой длины. Являясь теоретически оптимальным методом, арифметическое кодирование превосходит в эффективности префиксное кодирование. Оно также опережает префиксное кодирование в скорости построения системы кодов[3], однако из-за повышенной сложности нередко заметно уступает в скорости самого кодирования (имеется в виду процесс генерации кодовой последовательности).

Несмотря на некоторые существенные преимущества арифметического кодирования, данный метод до последнего времени был недостаточно распространен на практике. На сегодняшний день в большинстве коммерческих приложений для построения системы кодов переменной длины используется кодирование Хаффмана, являющееся лучшей с точки зрения эффективности реализацией префиксного кодирования. Арифметическое кодирование применяется лишь в тех случаях, когда требуется добиться максимально возможного качества информационного представления и когда скорость работы не имеет решающего значения[4].

## 3.1. Префиксное кодирование

Префиксное кодирование основано на выработке систем кодов переменной длины, которые принято называть *системами префиксных кодов* (или *кодов префикса*). Особенность этих систем состоит в том, что в пределах каждой из них более короткие по длине коды не совпадают с началом (префиксом) более длинных кодов – так называемое *свойство префикса*.

---

[1] Минимальная длина префиксного кода равна информационной единице.

[2] Употребление термина «код» в данном случае не совсем корректно. Подробнее об этом речь пойдет в п. 3.2.

[3] На самом деле этот этап в арифметическом кодировании фактически отсутствует (см. п. 3.2).

[4] В связи с повсеместным практическим внедрением новых высокоэффективных методов экономного кодирования в самое ближайшее время арифметическое кодирование может стать доминирующим способом генерации кода.

Рассмотрим систему из трех двоичных кодов: {«0», «10», «11»}. Как видно, ни один из этих кодов не является началом другого. Таким образом, о данной системе можно говорить как о системе префиксных кодов. Пусть трем выбранным кодам соответствуют символы «a», «b» и «c». Тогда информационному сообщению «abac» будет соответствовать кодовая последовательность «010011». Попытаемся восстановить исходное сообщение по данной кодовой последовательности. Первый нулевой бит можно интерпретировать только как символ «a», так как коды, соответствующие символам «b» и «c», начинаются с единицы. За первым битом следует единичный бит, который может быть началом кодов сразу двух символов: «b» и «c». Определить следующий символ позволяет значение третьего бита, которое однозначно указывает на символ «b». На данном этапе восстановленная часть сообщения приобретает вид «ab». Вслед за кодом символа «b» вновь следует нулевой бит. Как мы выяснили, это код, соответствующий символу «a». Завершают кодовую последовательность два единичных бита. Так же, как и в рассмотренном выше случае, значение последнего из них позволяет однозначно воспроизвести последний символ закодированного сообщения – «c». Итак, нам удалось правильно декодировать всю кодовую последовательность. Как нетрудно понять, это стало возможным благодаря выполнению свойства префикса.

Что дает на практике такая система префиксных кодов? В приведенном примере символ «a» встречается в сообщении «abac» с относительной частотой 1/2, а символы «b» и «c» – с относительной частотой 1/4. Предположим, что информационный источник, породивший данное сообщение, является источником без памяти. За неимением других сведений об источнике логично считать вероятности появления символов его на выходе равными относительным частотам. Тогда оптимальная длина кода для символа «a» должна быть равна $-\log_2(1/2)=1$ бит, а для символов «b» и «c» – $-\log_2(1/4)=2$ бита. Как видно, выбранная система префиксных кодов находится в полном соответствии с предполагаемым вероятностным распределением.

Применение префиксного кодирования в рассмотренном примере позволило получить оптимальное с точки зрения компактности представление информационного сообщения. Однако, если к данному сообщению добавить еще один символ «a», использование префиксного кодирования уже не даст оптимального результата. Символ «a» будет встречаться в новом сообщении с относительной частотой 3/5, а символы «b» и «c» – с относительной частотой 1/5. Оптимальные длины кодов для символов «a», «b» и «c» будут соответственно равны $-\log_2(3/5)\approx 0.74$, $-\log_2(1/5)\approx 2.32$ и $-\log_2(1/5)\approx 2.32$ бита. Таким образом, для представления нового сообщения достаточно приблизительно 6.86 бита. Для сравнения, максимально возможная эффективность префиксного кодирования в данном случае составляет 7 битов.

Системы префиксных кодов обычно получают построением *кодовых деревьев*. Степень ветвления в этих деревьях зависит от основания системы представления информации. Рассмотрим случай двоичной системы представления, которой соответствуют бинарные кодовые деревья (последующее рассуждение легко распространяется и на случай $m$-ичной системы представления). Как известно, каждый внутренний узел бинарного дерева имеет до двух исходящих ребер. Пронумеруем эти ребра так, чтобы одному из них соответствовал двоичный символ «0», а другому – «1». Так как в дереве не может быть циклов, от корневого узла к листовому узлу всегда можно проложить единственный маршрут. Если ребра дерева пронумерованы, то каждому такому маршруту соответствует некоторая уникальная двоичная последовательность. Множество всех таких последовательностей, очевидно, образует систему префиксных кодов.

Легко понять, что любой системе префиксных кодов однозначно соответствует некоторое кодовое дерево (например, рассмотренной системе префиксных кодов {«0», «10», «11»} соответствует кодовое дерево, изображенное на рис. 3.1). Таким образом, выполнение свойства префикса можно интерпретировать как наличие особой древовидной организации внутри системы кодов переменной длины. Помимо такой геометрической трактовки свойства префикса, возможна также и алгебраическая трактовка, которая гласит, что система префиксных кодов с натуральными длинами $l_1, \ldots, l_N$ в системе представления информации с основанием $m$ может существовать тогда и только тогда, когда выполняется *неравенство Крафта*

$$\sum_{i=1}^{N} m^{-l_i} \le 1.$$

Для того чтобы убедиться в тождественности этих, на первый взгляд непохожих утверждений, достаточно заметить, что выполнение неравенства Крафта фактически эквивалентно существованию дерева с $N$ листовыми узлами, находящимися на расстояниях $l_i$ от корневого узла.

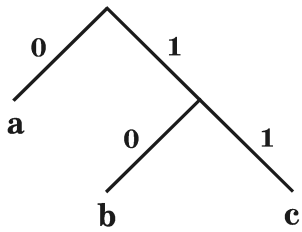



Рис. 3.1. Бинарное кодовое дерево для системы префиксных кодов {«0», «10», «11»} (коды соответствуют символам «a», «b» и «c»)

Кодовые деревья необходимо строить таким образом, чтобы длины маршрутов, ведущих от корневого узла к листовым узлам, были максимально приближены к теоретически оптимальным длинам кодов, соответствую-

щих различным символам информационного алфавита. Заметим, что речь здесь идет лишь о длинах маршрутов, но не о самих маршрутах. Дело в том, что нумерация ребер кодового дерева может быть осуществлена абсолютно произвольным образом, что обусловливает произвольность в выборе маршрута, то есть, в конечном итоге, произвольность в выборе состава префиксного кода. Иначе говоря, конкретная комбинация нулей и единиц, составляющая префиксный код, не играет особой роли, а определяющее значение имеет только его длина. Таким образом, для заданного вероятностного распределения может быть построено сразу несколько кодовых деревьев, соответствующих различным оптимальным системам префиксных кодов. Итак, как же строить эти деревья? Наиболее известными алгоритмами построения кодовых деревьев с конфигурацией, отвечающей заданному вероятностному распределению, являются алгоритмы Шеннона-Фано [8, 27] и Хаффмана [7]. Как было указано выше, система кодов, полученная с применением алгоритма Хаффмана, является лучшей с точки зрения качества информационного представления среди всех возможных систем префиксных кодов. Алгоритм Шеннона-Фано менее эффективен, однако он существенно более прост и нагляден. Помимо алгоритмов Шеннона-Фано и Хаффмана, можно также выделить алгоритмы Голомба [31] и Райса [49]. Рассмотрим наиболее распространенные алгоритмы префиксного кодирования применительно к двоичной системе представления информации.

### 3.1.1. Алгоритм Шеннона-Фано

Рассматриваемый алгоритм предложили независимо друг от друга Р. Фано [27] и К. Шеннон [8][1]. Алгоритм представляет собой рекурсивную процедуру, позволяющую на основе заданного вероятностного распределения появления символов на выходе источника информации поэтапно формировать отвечающую ему систему префиксных кодов.

Первоначально каждому символу информационного алфавита ставится в соответствие код нулевой длины («пустой» код) и вес, равный вероятности появления символа на выходе информационного источника в рамках выбранной информационной модели. Все символы алфавита сортируются по убыванию или возрастанию их весов, после чего упорядоченный ряд символов в некотором месте делится на две части так, чтобы в каждой из них сумма весов символов была примерно одинакова. К кодам символов, принадлежащих одной из частей, добавляется «0», а к кодам символов, принадлежащим другой части, добавляется «1» (добавляемое значение формирует оче-

[1] В действительности Фано и Шеннон предложили разные формулировки алгоритма. В данном пункте алгоритм будет рассмотрен в формулировке Фано. С вариантом Шеннона можно ознакомиться в [8].

редной крайний правый разряд кода). Как нетрудно понять, каждая из указанных частей сама по себе является упорядоченным рядом символов. Каждый из этих рядов, если он содержит более одного символа, в свою очередь делится на две части в соответствии с описанным выше принципом, и к кодам символов вновь добавляются соответствующие двоичные значения и т.д. Процесс завершается тогда, когда во всех полученных таким образом рядах остается ровно по одному символу.

Как видно, алгоритм Шеннона-Фано в действительности не строит кодовое дерево, однако его работу можно проиллюстрировать таким построением. При разделении группы символов происходит как бы разветвление некоторого узла бинарного дерева (листовой узел становится узлом родителем для двух новообразованных дочерних узлов), а присвоение нулей и единиц равносильно установлению соответствия между кодами и маршрутами движения по дереву от корневого узла к листовому узлу.

Работа алгоритма Шеннона-Фано проиллюстрирована на примере (рис. 3.2). В данном случае используется информационная модель, в которой порождающий информационный источник рассматривается как источник без памяти. Вероятности появления символов в данной модели заменяются частотами их вхождения в информационное сообщение[1]. В дальнейшем изложении такая модель будет именоваться *нулевой моделью*.

Насколько обоснован такой способ построения системы кодов переменной длины? Как оказывается, он не универсален даже в рамках префиксного кодирования. Для доказательства достаточно найти оптимальную систему префиксных кодов для 5-ти символов, вероятности появления которых равны 5/13, 2/13, 2/13, 2/13 и 2/13, и убедиться в том, что система, полученная с использованием алгоритма Шеннона-Фано, в данном случае оказывается менее выгодной.

Тем не менее алгоритм Шеннона-Фано достаточно эффективен. Несложно показать, что длина кода символа с вероятностью появления $p$, полученного с использованием данного алгоритма, превышает теоретически оптимальное значение $-\log_2 p$ менее чем на 1 бит. Эффективность алгоритма находит подтверждение и на практике: как правило, алгоритм Шеннона-Фано очень незначительно уступает в эффективности оптимальному алгоритму префиксного кодирования – алгоритму Хаффмана.

[1] Вероятности могут быть заменены частотами, так как в данном случае отсутствует необходимость в нормировке весов (частоты фактически представляют собой ненормированные вероятности).

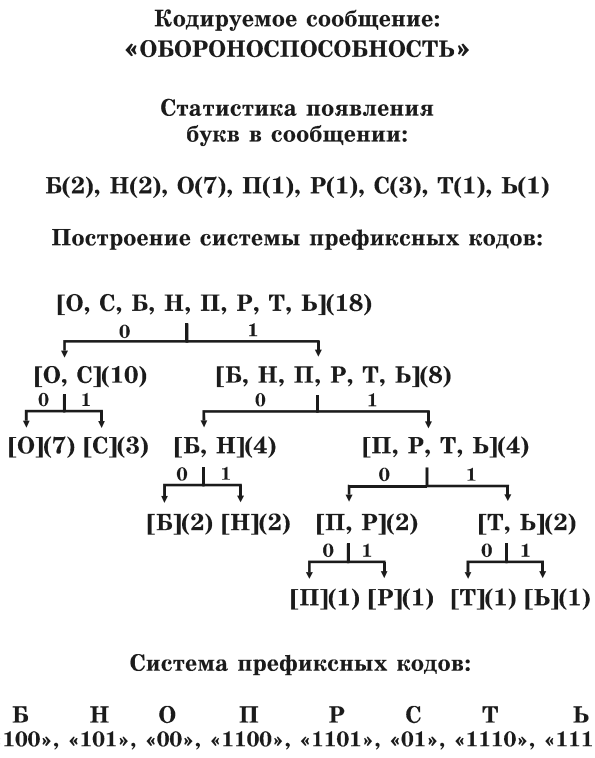



Рис. 3.2. Иллюстрация работы алгоритма Шеннона-Фано

### 3.1.2. Алгоритм Хаффмана

Данный алгоритм был описан Д. Хаффманом в работе [7] Хаффман предложил строить кодовое дерево не сверху вниз, как это неявно делается в алгоритме Шеннона-Фано, – от корневого узла к листовым узлам, а снизу вверх – от листовых узлов к корневому узлу. Такой подход оказался наиболее эффективным и получил весьма широкое распространение.

На начальном этапе работы алгоритма каждому символу информационного алфавита ставится в соответствие вес, равный вероятности (частоте) появления данного символа в информации. Символы помещаются в список, который сортируется по убыванию весов. На каждом шаге (итерации) два последних элемента списка объединяются в новый элемент, который затем помещается в список вместо двух объединяемых элементов. Новому элементу списка ставится в соответствие вес, равный сумме весов замещаемых элементов. Каждая итерация заканчивается упорядочиванием полученного нового списка, который всегда содержит на один элемент меньше, чем старый список.

Параллельно с работой указанной процедуры осуществляется последовательное построение кодового дерева. На каждом шаге алгоритма любому элементу списка соответствует корневой узел бинарного дерева, состоящего из вершин, соответствующих элементам, объединением которых был получен данный элемент. При объединении двух элементов списка происходит

объединение соответствующих деревьев в одно новое бинарное дерево, в котором корневой узел соответствует новому элементу, помещаемому в список, а замещаемым элементам списка соответствуют дочерние узлы этого корневого узла. Алгоритм завершает работу, когда в списке остается один элемент, соответствующий корневому узлу построенного бинарного дерева. Это дерево называется *деревом Хаффмана*. Система префиксных кодов может быть получена путем присваивания конкретных двоичных значений ребрам этого дерева. Пример построения дерева Хаффмана приведен на рис. 3.3. (в качестве информационной модели здесь вновь выступает нулевая модель).

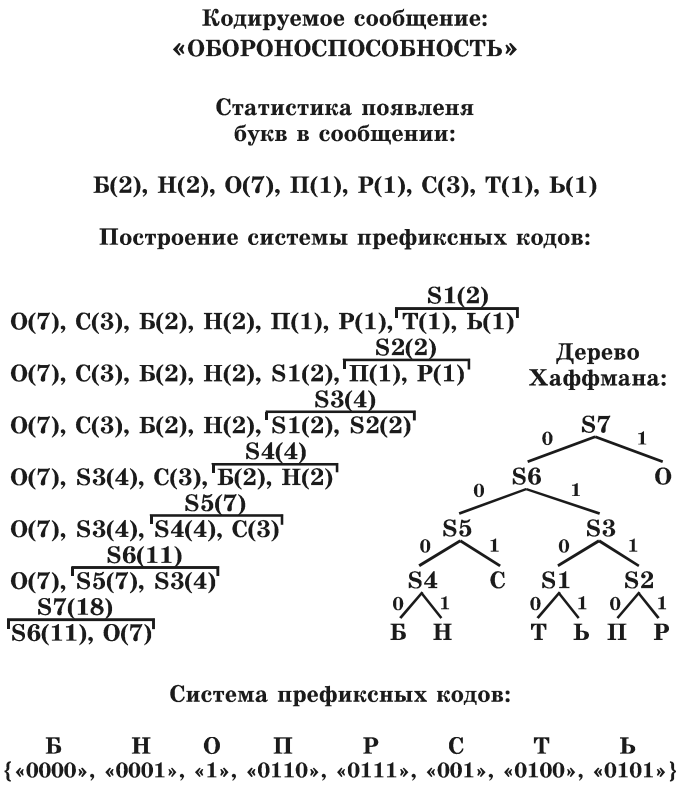



Рис. 3.3. Иллюстрация работы алгоритма Хаффмана

Докажем оптимальность алгоритма Хаффмана в рамках префиксного кодирования. Для этого достаточно показать, что система кодов, получаемая с использованием данного алгоритма, является решением задачи о минимизации суммы

$$\sum_{i=1}^{N} l_i p_i$$

при условии выполнения неравенства Крафта

$$\sum_{i=1}^{N} 2^{-l_i} \le 1$$

($l_1, \ldots, l_N$ – длины кодов, $p_1, \ldots, p_N$ – фиксированные вероятности появления символов). Для простоты предположим, что вероятности появления символов связаны неравенством $p_1 \ge p_2 \ge \ldots \ge p_{N-1} \ge p_N$. Тогда решение $\{l_1, \ldots, l_N\}$, очевидно, должно удовлетворять соотношению $l_1 \le l_2 \le \ldots \le l_{N-1} \le l_N$. Докажем, что $l_{N-1} = l_N$. Допустим, что равенство не выполняется, то есть $l_{N-1} < l_N$. Представим неравенство Крафта в следующем виде:

$$2^{l_N - l_1} + 2^{l_N - l_2} + \ldots + 2^{l_N - l_{N-1}} + 1 \le 2^{l_N}.$$

С учетом предположения $l_1 \le l_N \le \ldots \le l_{N-1} < l_N$ данное неравенство является строгим, так как

$$2 \cdot (2^{l_N - l_1 - 1} + 2^{l_N - l_2 - 1} + \ldots + 2^{l_N - l_{N-1} - 1}) + 1 \ne 2^{l_N}.$$

Поэтому можно написать

$$2^{l_N - l_1} + 2^{l_N - l_2} + \ldots + 2^{l_N - l_{N-1}} + 2 \le 2^{l_N},$$

что равносильно

$$\sum_{i=1}^{N-1} 2^{-l_i} + 2^{-(l_N - 1)} \le 1.$$

Таким образом, система $\{l_1, \ldots, l_N\}$ на самом деле не является решением указанной задачи, ибо существует более выгодная система $\{l_1, \ldots, l_N - 1\}$, удовлетворяющая неравенству Крафта. Следовательно, предпосылка $l_{N-1} < l_N$ ошибочна.

Так как $l_{N-1} = l_N$, задача о минимизации может быть несколько упрощена: минимизировать сумму

$$\sum_{i=1}^{N-2} l_i p_i + (l_N - 1) \cdot (p_{N-1} + p_N) + (p_{N-1} + p_N) = \sum_{i=1}^{N-1} l'_i p'_i + p'_{N-1}$$

при условии

$$\sum_{i=1}^{N-2} 2^{-l_i} + 2^{-l_N} + 2^{-l_N} = \sum_{i=1}^{N-2} 2^{-l_i} + 2^{-(l_N-1)} = \sum_{i=1}^{N-1} 2^{-l'_i} \le 1,$$

где $l'_1 = l_1$, $l'_2 = l_2$, ... , $l'_{N-2} = l_{N-2}$, $l'_{N-1} = l_N - 1$ и $p'_1 = p_1$, $p'_2 = p_2$, ... , $p'_{N-2} = p_{N-2}$, $p'_{N-1} = p_{N-1} + p_N$. Как видно, нам удалось уменьшить число неизвестных в условии задачи о минимизации с $N$ до $N-1$. Осталось заметить, что предложенная процедура сокращения числа неизвестных представляет собой не что иное, как алгоритм Хаффмана.

Итак, система кодов, полученная в результате построения дерева Хаффмана, является оптимальной системой префиксных кодов. Отсюда, однако, ни в коем случае не следует вывод о том, что префиксные системы, полученные с использованием других алгоритмов, всегда менее эффективны. Так, на рассмотренных примерах (см. рис. 3.2, 3.3) хорошо видно, что алгоритмы Шеннона-Фано и Хаффмана, хотя и строят совершенно разные кодовые системы (коды для одних и тех же символов различаются как по длине, так и по составу), в данном конкретном случае оказываются одинаково эффективными – при использовании каждого из этих алгоритмов объем представления сообщения «ОБОРОНОСПОСОБНОСТЬ» равен 48 битам. Таким образом, алгоритм Хаффмана является не единственным алгоритмом, позволяющим строить оптимальные системы префиксных кодов, однако единственным до конца универсальным алгоритмом – в отличие от других алгоритмов, данный алгоритм всегда оптимален.

Основной недостаток алгоритма Хаффмана заключается в сложности процесса построения системы кодов. Многие альтернативные алгоритмы префиксного кодирования, в частности алгоритм Шеннона-Фано, представляются в этом отношении более предпочтительными. Справедливости ради заметим, что данная особенность редко учитывается при выборе алгоритма кодирования, так как при необходимости частых перестроений системы кодов, как правило, применяются специализированные подходы (см. гл. 4).

Алгоритм Хаффмана является самым распространенным алгоритмом генерации кода переменной длины. На протяжении многих лет кодирование Хаффмана, основанное на простейшей нулевой информационной модели, выступало в роли самостоятельного алгоритма экономного кодирования[1]. Сегодня алгоритм Хаффмана используется исключительно как способ получения систем кодов переменной длины. Он находит свое воплощение в большинстве утилит сжатия и архивации информации.

Рассмотренные алгоритмы префиксного кодирования применимы и в случае *m*-ичной системы кодирования. В процессе работы алгоритма

[1] Следствием этого стало появление особого термина – «кодирование по Хаффману».

Шеннона-Фано ряд символов должен делиться на $m$ частей. Построение $m$-арного дерева Хаффмана необходимо производить объединением на каждой итерации $m$ последних элементов списка. При определении конкретных кодовых комбинаций каждому ребру дерева нужно ставить в соответствие одно из $m$ различных значений.

### 3.2. Арифметическое кодирование

Основой арифметического кодирования является неопубликованный алгоритм П. Элайеса. В том виде, в котором метод существует сегодня, он впервые рассматривается в [47, 50, 54]. Существенный вклад в развитие метода арифметического кодирования в разное время внесли П. Говард, М. Гуаззо, Д. Клири, Г. Лэнгдон, Э. Моффат, Р. Нил, Р. Паско, Д. Риссанен, Ф. Рубин, Я. Уиттен и М. Шиндлер [32, 47, 50, 51, 53, 54, 59].

Арифметическое кодирование использует весьма нестандартный подход к генерации кода. Несмотря на объективную сложность данного подхода, идея, лежащая в его основе, весьма проста. На любом этапе работы арифметического кодирования интервал [0, 1) вещественной прямой делится на интервалы типа $[a, b)$, каждый из которых соответствует какому-то символу информационного алфавита. Длина интервала численно совпадает с вероятностью появления соответствующего символа на выходе источника информации в выбранной информационной модели. Если выбрать любое вещественное число из интервала [0, 1), оно будет принадлежать некоторому интервалу в разбиении, соответствующему вполне определенному символу. Таким образом, число всегда однозначно определяет некоторый символ алфавита.

Пусть в сообщении встретился символ «x», которому соответствует интервал $[a, b)$, за которым следует символ «y» с соответствующим ему интервалом $[c, d)$. Осуществим равномерную проекцию интервала [0, 1) на интервал $[a, b)$. При такой проекции интервалу $[c, d)$ в интервале $[a, b)$ будет соответствовать интервал $[a + (b - a) \cdot c,\ a + (b - a) \cdot d)$. Любое число из последнего интервала однозначно определяет последовательность «xy». Действительно, оно принадлежит интервалу $[a, b)$, который соответствует символу «x», и подинтервалу данного интервала, который обратной проекцией $[a, b)$ на [0, 1) переводится в $[c, d)$. Интервал $[c, d)$ соответствует второму символу последовательности – «y». Процесс построения интервала проиллюстрирован на рис. 3.4.

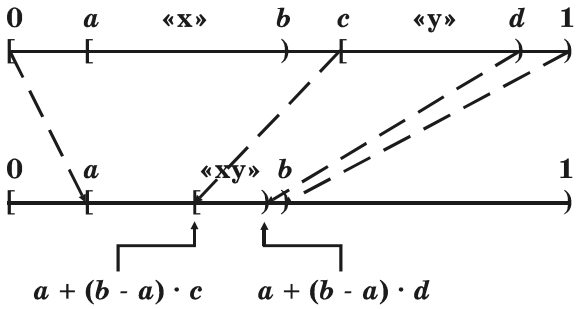



Рис. 3.4. Иллюстрация работы арифметического кодирования

Аналогичным образом производится построение интервала для трех и более следующих подряд символов. Каждый раз при добавлении очередного символа интервал сужается. В результате данной процедуры получается некоторый достаточно узкий интервал, любое число в котором однозначно определяет исходное сообщение. Обычно в качестве такого определяющего (кодирующего) числа выбирают нижнюю границу интервала.

В отличие от префиксного кодирования, в арифметическом кодировании символам не могут быть сопоставлены конкретные кодовые комбинации. Понятие «код» в данном случае несколько абстрактно и применимо только для сообщения в целом[1]. Здесь можно говорить только о вкладе, вносимом каждым символом, входящим в кодируемое информационное сообщение, в результирующую кодовую комбинацию. Этот вклад определяется длиной интервала, соответствующего символу, то есть вероятностью появления символа.

Рассмотрим пример, иллюстрирующий работу арифметического кодирования. Предположим, что необходимо закодировать сообщение «aaab». Относительная частота появления символа «a» в сообщении равна 3/4, а символа «b» – 1/4. Следовательно, символу «a» можно поставить в соответствие интервал [0, 3/4), а символу «b» – интервал [3/4, 1) (используется нулевая модель). Применим арифметическое кодирование к рассматриваемому сообщению. Последовательности «aa» соответствует интервал $[0+(3/4-0)\cdot 0,\ 0+(3/4-0)\cdot 3/4) = [0,\ 9/16)$, последовательности «aaa» – интервал $[0+(9/16-0)\cdot 0,\ 0+(9/16-0)\cdot 3/4) = [0,\ 27/64)$ и, наконец, последовательности «aaab» – интервал $[0+(27/64-0)\cdot 3/4,\ 0+(27/64-0)\cdot 1) = [81/256,\ 27/64)$. Внутри полученного интервала находится число $96/256 = 3/8$. В двоичной системе представления это число можно представить в виде 0.011. Таким образом, для кодирования сообщения «aaab» достаточно трех битов (целая часть дво-

[1] Такой код иногда называют *неразделимым*.

ичной дроби, равная нулю, может быть отброшена). Заметим, что минимальная длина информационного представления указанного сообщения, полученного с применением префиксного кодирования, равна четырем битам.

Для кодирования мы вполне могли выбрать и другое число, например, 81/256. Тогда длина кода была бы равна не трем, а восьми битам, так как 81/256 в двоичной форме представляется как 0.01010001. Ясно, что в этом случае мы получили бы крайне неэффективное информационное представление. В ситуациях, подобных данной, когда обрабатываемое сообщение является коротким, произвольный выбор способа кодирования может заметно повлиять на его эффективность, однако при обработке достаточно длинных сообщений разница в длине кода, обусловленная выбором кодирующего числа, в относительном выражении оказывается очень незначительной.

При обработке сообщений бесконечной длины арифметическое кодирование дает результат, эквивалентный применению теоретически оптимальной системы кодирования. Применительно к конечному объему информации арифметическое кодирование немного уступает оптимальному способу кодирования. Докажем, что расхождение результата арифметического кодирования с теоретически оптимальным результатом составляет менее одной информационной единицы при условии, что используется кодирующее число с минимальной длиной кода.

Пусть вследствие описанной выше процедуры вложения интервалов был получен некоторый интервал $[a, b)$ длины $p$ (так называемая *накопленная вероятность*). Нам необходимо выбрать некое число из этого интервала, представимое как можно меньшим числом $m$-ичных разрядов, где $m$ – основание системы представления. Для этого нужно построить на интервале $[0, 1)$ одномерную сеть с постоянным периодом, содержащую $m^N$ точек (местоположение любой точки указывается $N$-разрядным числом), ровно одна из которых (кодирующая) принадлежит интервалу $[a, b)$. Ясно, что искомая сеть должна иметь период, не превышающий $p$, то есть

$$\frac{1}{m^N} \le p .$$

Отсюда получаем, что $N = \lceil -\log_m p \rceil$. Таким образом, кодирующее число имеет $\lceil -\log_m p \rceil < -\log_m p + 1$ разрядов.

Выбор интервалов в арифметическом кодировании является достаточно условным. В рассмотренном примере символу «a» можно было сопоставить интервал $[1/4, 1)$, а символу «b» – интервал $[0, 1/4)$. При этом длина получившегося интервала не изменилась бы, а различие состояло бы только в местоположении этого интервала внутри интервала [0, 1). Здесь можно заме-

тить явное сходство с префиксным кодированием, где выбор конкретных кодовых комбинаций также является произвольным; однако если в префиксном кодировании значение имеют только длины кодов, то в арифметическом кодировании определяющими являются длины интервалов.

Метод арифметического кодирования достаточно сложно реализуется на практике. Затруднения вызывают операции над числами, содержащими большое количество знаков после запятой. В данном случае целесообразнее работать в рамках целочисленной арифметики. Результатом продолжительных исследований стали эффективные целочисленные реализации метода [32, 47, 50, 51, 53, 59]. Их работа основывается на следующем принципе. Число из интервала [0, 1) представляется $m$-ичной последовательностью аналогично тому, как это делалось в рассмотренном примере. Кодирование превращается в процесс уточнения членов этой последовательности. Неменяющаяся часть последовательности (при сужении интервала значения в некоторых разрядах его границ перестают меняться) выводится как часть выходного кода и в дальнейшем не рассматривается.

При определении границ интервалов неизбежно используются операции умножения и деления, имеющие, как известно, большую вычислительную сложность. Упростить процесс кодирования можно, заменив данные операции совокупностями более простых операций сложения и сдвига. Такой подход использовался исследователями из IBM при разработке быстрой реализации арифметического кодирования (см. [48]). Принципиально иной способ повышения производительности метода описан в [35]. Для упрощения расчета в работе предлагается использовать заранее полученные таблицы результатов арифметических операций.

Применение целочисленной арифметики и специальных приемов повышения производительности в методе арифметического кодирования позволяет избежать сложных операций с плавающей точкой и существенно упростить хранение данных, но иногда ведет к некоторому снижению эффективности кодирования. В целом, если сравнивать префиксное кодирование и практические реализации арифметического кодирования, последние по-прежнему остаются наиболее близкими к теоретически оптимальному способу кодирования.

## 4. Хранение информации о способе кодирования

Для правильного декодирования информационного представления необходимо иметь полную информацию о способе, с помощью которого это представление было получено. Возникает вопрос – как сделать эту информацию доступной в процессе декодирования? Решение этой проблемы тесно связано с разделением алгоритмов экономного кодирования на *адаптивные*

– в них способ кодирования меняется во время работы – и *неадаптивные* – в таких алгоритмах способ кодирования во время работы остается неизменным[1].

При обработке информации неадаптивным алгоритмом можно либо использовать заранее предопределенный способ кодирования, никак не зависящий от обрабатываемой информации, либо же применять некоторый специально подобранный способ кодирования и хранить вместе с закодированной информацией его описание. Первый подход не обладает универсальностью и пригоден для обработки очень ограниченного числа информационных источников. Более предпочтителен второй подход, эффективность которого зависит от соотношения между выигрышем, получаемым от кодирования, и издержками, связанными с хранением способа кодирования. Алгоритмы, использующие второй подход, часто называют *полуадаптивными алгоритмами*, поскольку они являются адаптивными только к информации в целом, но не к отдельным ее частям.

Адаптивные алгоритмы в большинстве своем позволяют не хранить дополнительную информацию о способе кодирования, даже несмотря на то, что во время их работы способ кодирования постоянно меняется. В таких алгоритмах изменения в способе кодирования совпадают на одних и тех же этапах работы кодера и декодера, так что если кодер и декодер начинают работу в одном и том же начальном состоянии, то они и в дальнейшем работают согласованно. Для примера рассмотрим префиксное кодирование. Для того, чтобы декодер мог корректно интерпретировать закодированную информацию, на каждом этапе декодирования должна строиться система префиксных кодов, идентичная системе префиксных кодов, построенной на соответствующем этапе кодирования. Такая система может строиться на основе распределения относительных частот появления символов в сообщении (см. п. 3.1). При кодировании/декодировании первого символа появление символов можно считать равновероятным (начальное состояние). При кодировании/декодировании последующих символов каждый раз необходимо перестраивать систему кодов с целью приведения ее в соответствие с частотным распределением в уже обработанной части информации. При использовании такого подхода нам не потребуется хранить вместе с кодом статистическую информацию, как это следовало бы сделать в случае с неадаптивным алгоритмом. Кроме того, из-за адаптации к изменению статистических характеристик источника информации, описанный адаптивный алгоритм может оказаться более эффективным по сравнению со своим не-

[1] Два способа кодирования считаются различными, если с их применением по-разному кодируется хотя бы одно информационное сообщение.

адаптивным аналогом, где используется не меняющаяся во времени (статическая) система префиксных кодов.

Однако существует и другой тип адаптивных алгоритмов. В таких алгоритмах декодер по объективным причинам не может без использования дополнительной информации адекватно реагировать на изменения в способе кодирования. К примеру, если кодирование осуществляется с учетом характеристик последующей, еще незакодированной части информации, генерируемый код невозможно декодировать, так как на соответствующем этапе декодирования эти характеристики являются неизвестными. В подобных случаях нам, естественно, не удастся избежать хранения дополнительной служебной информации, описывающей способ кодирования.

Из-за сложности процесса адаптации адаптивные алгоритмы обычно работают значительно медленнее своих неадаптивных и полуадаптивных аналогов. Так, при использовании префиксного кодирования адаптация часто сопряжена с весьма трудоемкой перестройкой кодового дерева, что, естественно, приводит к необходимости применения особых решений. Примером такого решения является быстрый адаптивный вариант алгоритма Хаффмана, разработанный Р. Галлагером. В [30] Галлагер предложил оригинальный способ изменения конфигурации кодового дерева Хаффмана, не требующий его полной перестройки. К сожалению, использование данного приема в большинстве случаев все равно не позволяет достигнуть желаемого результата – скорость адаптации по-прежнему остается достаточно низкой. Существенного увеличения производительности адаптивного префиксного кодирования удается добиться только за счет заметного снижения его эффективности. Одним из вариантов такого решения является использование *Splay-деревьев* (см. [37]).

Адаптивное арифметическое кодирование реализуется значительно проще, чем адаптивное префиксное кодирование. Адаптация в арифметическом кодировании сводится к изменению границ интервалов, соответствующих символам информационного алфавита. Поэтому адаптивное арифметическое кодирование в некоторых своих реализациях (см., например, [59]) превосходит по скорости многие реализации адаптивного префиксного кодирования.

Серьезным недостатком полуадаптивных и некоторых адаптивных алгоритмов является необходимость в заблаговременном определении способа кодирования, который будет использоваться при обработке информации. Информацию фактически нужно обработать дважды: при проведении анализа и непосредственно при кодировании. Преимущество неадаптивных алгоритмов и адаптивных алгоритмов, позволяющих не хранить дополнительную информацию, описывающую способ кодирования, состоит в том, что они не требуют проведения предварительного анализа и обрабатывают ин-

формацию за один проход. Таким образом, все алгоритмы экономного кодирования можно разделить на *однопроходные и многопроходные*. Последние, очевидно, обладают значительно более узкой областью применения.

Обобщая вышесказанное, можно сделать вывод о том, что выбор между адаптивными, полуадаптивными и неадаптивными алгоритмами должен зависеть от конкретной задачи. Неадаптивные и полуадаптивные алгоритмы имеют ограниченную применимость, но обладают достаточно высокой скоростью работы. Адаптивные алгоритмы являются менее производительными, однако они почти всегда применимы и во многих случаях более эффективны в сравнении со своими неадаптивными и полуадаптивными аналогами.

## 5. Основные методы экономного кодирования без потерь последовательной дискретной информации

С появлением метода арифметического кодирования проблема генерации кода была фактически решена. С тех пор основное внимание стало уделяться вопросам, связанным с моделированием. Существующие различия между информационными моделями послужили основой современной классификации методов экономного кодирования.

Методы экономного кодирования без потерь последовательной дискретной информации можно условно разделить на три группы: *статистические* методы, *словарные* методы и *контекстные* методы. Методы первых двух групп строятся на основе соответственно *статистических* и *словарных* моделей. Они достаточно многочисленны и представляют почти все существующие подходы в области экономного кодирования дискретной информации. Методы третьей группы менее многочисленны и плохо поддаются классификации по типу информационной модели. На сегодняшний день контекстная группа включает в себя только два метода, в одном из которых используется модель, которую следует скорее отнести к словарным моделям, а в другом – модель особого типа, не являющаяся ни словарной, ни статистической. Причиной выделения данных методов в отдельную группу послужило то, что они оперируют одними и теми же информационными объектами – *контекстами данных*.

Рассмотрим подробнее основные методы каждой из перечисленных групп.

### 5.1. Статистические методы

Для осуществления эффективного экономного кодирования сообщений, поступающих с выхода источника информации, требуется знание его харак-

теристик. При рассмотрении источников с памятью, как мы знаем, в качестве таких характеристик принято брать условные вероятности появления символов в сообщении вслед за различными символьными последовательностями. В случае, когда нет никакой дополнительной информации об источнике, определить эти вероятности можно только путем статистического анализа его информационной выборки. Данный принцип лежит в основе статистических методов экономного кодирования.

Последовательность символов, непосредственно предшествующую некоторому символу в информационном сообщении, принято называть *контекстом* этого символа, а длину этой последовательности – *порядком контекста*. На основе статистики, собираемой во время обработки информации, статистический метод позволяет оценить вероятность появления в текущем контексте произвольного символа или последовательности символов. Оценка вероятности определяет длину кода, который, как правило, генерируется с использованием арифметического кодирования.

Статистические методы в большинстве своем являются адаптивными методами. Вычисление вероятностных оценок осуществляется одним и тем же способом на этапах кодирования и декодирования, что позволяет не хранить описание информационной модели.

Статистические методы отличаются способом получения оценок условных вероятностей появления символов в различных контекстах. На сегодняшний день можно выделить четыре основных способа получения таких оценок, которые легли в основу соответственно четырех статистических методов: метода PPM, метода DMC, метода CTW и метода, основанного на использовании нейронных сетей.

Наибольшее распространение среди перечисленных методов получил предложенный еще в середине 80-х годов метод PPM, долгое время практически безраздельно являвшийся наиболее эффективным методом экономного кодирования. Метод DMC[1], хотя и появился практически одновременно с методом PPM, все же не получил такого широкого распространения. Он обладает меньшей эффективностью в сравнении с методом PPM и представляет интерес в основном только с идейной точки зрения. Оставшиеся два метода относятся к современными статистическим методам экономного кодирования. Метод CTW стал результатом целенаправленных теоретических исследований по поиску так называемого «универсального» метода экономного кодирования. В отличие от метода CTW, метод экономного кодирования, основанный на использовании нейронных сетей, первоначально вообще не имел прямого отношения к области экономного кодирования. Он разра-

[1] Термин «метод DMC» выступает здесь как собирательное название для возможных вариации алгоритма, впервые описанного в работах [26, 33].

батывался как общий метод моделирования и предсказания в рамках совершенно иной дисциплины – теории искусственных нейронных сетей. Оба указанных метода отражают наиболее перспективные направления в дальнейшем развитии статистического подхода в экономном кодировании и уже сегодня могут составить конкуренцию методу PPM.

### 5.1.1. Метод PPM

Оценку вероятности появления символов на выходе источника информации можно получать на основе контекстов только некоторого строго определенного порядка. В этом случае говорят, что используется *контекстная модель* соответствующего порядка. Такой метод, естественно, малоэффективен по причине того, что реальные информационные источники, как правило, не обладают постоянной памятью. Более разумно при определении вероятностей появления символов учитывать модели разных порядков, то есть производить так называемое *смешивание моделей*.

Ясно, что конечный вклад различных моделей в вероятностную оценку должен как-то различаться. Одним из возможных решений этой проблемы является введение весов, определяющих влияние той или иной модели на результирующую оценку[1]. В методе PPM (Prediction by Partial Matching – «предсказание по частичному совпадению», [25]) используется несколько более упрощенный подход. Вероятность появления символа всегда оценивается в текущем контексте какого-то одного порядка, то есть в некоторой конкретной контекстной модели.

Вариации метода PPM, как правило, работают по следующей схеме. Первоначально для оценки вероятности появления символа берется контекст некоторого заранее определенного достаточно большого порядка, то есть выбирается некая конкретная контекстная модель. Если кодируемый символ ранее в таком контексте не встречался, то вероятность его появления, очевидно, не может быть оценена с использованием выбранной модели. В этом случае кодер генерирует код служебного *символа перехода*[2] (*escape symbol*), сигнализирующего о невозможности использования рассматриваемой модели и необходимости перехода к другой модели. После этого осуществляется попытка оценить вероятность появления кодируемого символа с помощью модели следующего по убыванию порядка (возможен переход и на модель более низкого порядка). Если использование новой модели также не позволяет получить вероятностную оценку, кодер снова генерирует код служеб-

[1] Смешивание на основе весов, вероятно, впервые рассматривается в [5].

[2] Для обозначения данного символа часто употребляется другой, близкий по смыслу термин – *символ ухода*.

ного символа перехода и снова рассматривается модель меньшего порядка и т. д. Для гарантии завершения описанного процесса вводится дополнительная модель −1-го порядка, с одинаковой вероятностью оценивающая появление всех символов информационного алфавита (то есть символ будет оценен, даже если он ранее вообще не встречался). В дальнейшем рассмотренная схема будет называться *схемой контекстного спуска*.

Как видно, при оценке вероятности появления символа в методе PPM учитываются самые различные контекстные модели, причем, на первый взгляд, этого удается достичь без использования весов. В действительности это не совсем так. Вес незримо присутствует в вероятностной оценке и определяется вкладом, вносимым служебными символами перехода. Символы перехода фактически оказываются полноправными символами информационного алфавита. Как и обычным символам, символам перехода необходимо ставить в соответствие определенные кодовые комбинации. Это приводит к необходимости учета символов перехода при накоплении статистической информации, что следует рассматривать как неявное взвешивание.

Говорить о неких отдельно взятых кодовых комбинациях, когда речь идет об арифметическом кодировании, безусловно, не совсем корректно. Под кодами обрабатываемого и служебного символов на самом деле подразумевается вклад соответствующих условных вероятностей в результирующую вероятностную оценку, используемую для вычисления длины кода для всей обрабатываемой информации (см. п. 3.2). Вклад каждого кодируемого символа определяется вероятностями появления задействованных при его кодировании служебных символов перехода (данные вероятности определяются отдельно в рамках каждой модели) и вероятностью появления самого символа в *оценивающей модели* (то есть в модели, задействованной для оценки).

При оценке вероятности появления символа в контексте можно использовать два различных подхода. Первый сопряжен с вычислением вероятностной оценки в модели данного порядка без учета моделей более высоких порядков. Такой подход недостаточно эффективен, так как, если символ может быть оценен моделью некоторого порядка, то, согласно описанной схеме, он никогда не будет оценен моделью более низкого порядка, что, однако, никак не учитывается при получении оценки с применением моделей низших порядков. Для примера рассмотрим ситуацию, когда требуется оценить вероятность появления символа «c» в контексте «bcabdab», при условии, что оценка производится, начиная с модели второго порядка. Так как символ «c» в контексте «ab» ранее не встречается, то в кодовую последовательность добавляется код служебного символа перехода, сигнализирующий о необходимости оценки символа «c» с использованием модели первого порядка. Теперь оценка производится с учетом текущего контекста первого

порядка – «b». В данном контексте встречаются два символа: «c» и «d». Однако символ «d» встречается также и в контексте второго порядка «ab». Значит, в случае его появления в текущем контексте вместо символа «c» он был бы закодирован с применением модели второго, а не первого порядка. Следовательно, символ «d» не имеет смысла учитывать при оценке вероятности появления символа «c» в контексте «b». Итак, второй подход заключается в учете при вычислении вероятностной оценки контекстных моделей более высокого порядка по сравнению с порядком модели, используемой для оценки.

Про вариации метода PPM, основанные на последнем подходе, говорят, что они используют механизм *исключений* (*exclusions*), так как из вероятностной оценки исключается вклад моделей более высоких порядков. Применение механизма исключений приводит к повышению эффективности кодирования, однако из-за увеличения сложности оценки значительно снижается его производительность.

Для повышения скорости кодирования в практических реализациях метода PPM применяются так называемые *исключения при обновлении* (*update exclusions*). Обновление – процесс, заключающийся в адаптивном изменении параметров различных контекстных моделей во время обработки информации. Изменения, вносимые после кодирования (декодирования) очередного символа, обычно сводятся к увеличению содержимого счетчиков появления данного символа в текущих контекстах разного порядка. Содержимое счетчиков в дальнейшем используется при вычислении вероятностных оценок. В случае применения механизма исключений при обновлении содержимое счетчика для символа в контексте не увеличивается, если данный символ оценивается контекстом более высокого порядка (то есть производится обновление всех моделей, порядок которых больше или равен порядку модели, использованной для оценки символа). Такой подход дает значительный выигрыш в скорости работы, так как ограничивается количество операций, необходимых для обновления моделей на каждом шаге, а кроме того, использование данного подхода позволяет немного увеличить эффективность кодирования. (Последнее достигается за счет повышения роли контекстных моделей, непосредственно задействованных при получении оценок.) Заметим, что механизм исключений при обновлении не является заменой для механизма простых исключений, хотя исключения при обновлении частично выполняют функцию простых исключений. Поэтому для достижения наибольшей эффективности обычно задействуются сразу оба механизма.

Способ оценки вероятностей переходов и появлений символов в методе PPM является предметом особого рассмотрения. Как правило, вариации ме-

тода различаются именно по способу оценки. В методе PPMA[1] вероятность перехода к другой модели и вероятность появления символа в текущем контексте определяются в соответствии с формулами

$$p_{esc} = \frac{1}{c+1}, \quad p_s = \frac{c_s}{c+1},$$

где $c$ – число появлений текущего контекста, а $c_s$ – число появлений символа $s$ в текущем контексте. Для определения $c$ и $c_s$ для каждого символа в модели заводится отдельный счетчик. Метод PPMB использует другие оценки:

$$p_{esc} = \frac{d}{c}, \quad p_s = \frac{c_s - 1}{c},$$

где $d$ – число различных символов, встреченных в текущем контексте. Методы PPMA и PPMB разработали Д. Клири и Я. Уиттен [25]. При оценке вероятности в этих методах применяется механизм исключений.

Э. Моффат в [43, 44] предложил более эффективный и более высокопроизводительный метод PPMC. Для вычисления вероятностных оценок в данном методе используются формулы

$$p_{esc} = \frac{d}{c+d}, \quad p_s = \frac{c_s}{c+d}.$$

Символ перехода рассматривается здесь как обычный символ. Для него заводится особый счетчик, содержимое которого увеличивается на 1 при каждом появлении в текущем контексте ранее не встречавшегося символа. Появление такого символа означает увеличение суммарного веса модели на 2, так как на 1 увеличивается содержимое счетчиков самого символа и символа перехода. С целью повышения производительности в методе PPMC используются исключения при обновлении.

Дальнейшим развитием метода PPMC стал метод PPMD, разработанный П. Говардом и Д. Виттером [34, 35]. Он отличается от метода PPMC тем, что при появлении символа, который ранее не встречался в текущем контексте, содержимое счетчиков обычного символа и символа перехода увеличиваются не на 1, а на 1/2. Таким образом, суммарный вес модели все-

[1] Название расшифровывается как метод PPM с оценкой вероятности перехода по методу A. Подобный способ образования аббревиатуры де-факто является стандартным для вариаций метода PPM.

гда увеличивается на 1, что, как оказывается, положительно влияет на эффективность кодирования. Формулы для определения вероятностных оценок в данном случае приобретают вид

$$p_{esc} = \frac{d/2}{c}, \quad p_s = \frac{c_s - 1/2}{c}.$$

Среди методов, использующих другие оценки, особого упоминания заслуживают методы PPMP, PPMX и PPMXC [73].

Эффективность метода PPM во многом предопределяется правильностью выбора порядка первой оценивающей модели. Этот порядок должен находиться в полном соответствии с порядком памяти кодируемого информационного источника. Опытным путем было установлено [24, 61], что при обработке англоязычной текстовой информации в качестве первой оценивающей модели следует выбирать модель, порядок которой лежит в пределах от 4 до 6. Данное решение вполне оправданно по отношению не только к указанной информации, но и ко многим другим распространенным типам информации, в частности, к русскоязычной текстовой информации. При этом, однако, оно абсолютно неприменимо в случае, когда порядок памяти порождающего источника достаточно мал (частая генерация символов перехода при недостаточно адекватной оценке вероятности их появления может стать причиной снижения эффективности кодирования).

Несмотря на то, что выбор первой оценивающей модели далеко не однозначен, во всех вышеупомянутых вариациях метода PPM порядок этой модели фиксируется. В данном случае можно говорить о четкой ориентации на источники, порядок памяти которых ограничен и меняется в очень незначительных пределах (идеальной моделью такого источника может служить марковский источник). Как уже отмечалось, большинство реальных информационных источников весьма далеки от такого приближения. Поэтому совершенно очевидно, что наиболее эффективным кодирование станет только тогда, когда порядок первой оценивающей модели будет меняться в зависимости от особенностей обрабатываемой информации.

Алгоритм выбора первой оценивающей модели в процессе кодирования больше известен под аббревиатурой LOE (Local Order Estimation). Один из вариантов LOE состоит в том, чтобы в качестве первой оценивающей модели выбирать модель, лучше всего предсказывавшую в прошлом появление символов. Этот на первый взгляд очевидный способ выбора в большинстве случаев является неоправданным. Более удачное решение нашло свое отражение в методе PPM* [24].

Для оценки появления очередного символа в методе PPM* допускается использование контекстной модели неограниченного порядка. При оценке

задействованы так называемые *детерминированные контексты* (*deterministic contexts*), то есть такие контексты, которые предсказывают появление ровно одного символа (в данных контекстах ранее встречался какой-то один символ). Выбор первой оценивающей модели производится в соответствии со следующим правилом. Среди всех детерминированных контекстов выбирается контекст с минимальным порядком, а в случае отсутствия таковых берется уже ранее встречавшийся в сообщении контекст максимального порядка. Выбранный контекст используется первым при получении вероятностной оценки.

Вероятности появления символов в методе PPM* вычисляются по формулам, используемым в методе PPMC, которые в данном случае оказываются наиболее подходящими. Механизм исключений в методе PPM* не претерпел изменений, однако вместо исключений при обновлении здесь предлагается применять *частичные исключения при обновлении* (*partial update exclusions*). Отличие частичных исключений при обновлении от простых исключений при обновлении состоит в том, что при использовании частичных исключений обновлению подлежит содержимое счетчиков появления обработанного символа не во всех контекстах, а только в детерминированных контекстах и в одном недетерминированном контексте – в том, который имеет наибольший порядок (имеются в виду контексты, использованные для оценки символа). Благодаря нововведениям, использованным в методе PPM*, в некоторых (к сожалению, очень редких) случаях он оказывается более эффективным в сравнении с рассмотренными ранее вариациями метода PPM.

Альтернативное решение проблемы LOE было использовано Ч. Блюмом в алгоритме PPMZ [17]. В качестве первой оценивающей модели Блюм предлагает выбирать ту модель, в рамках которой вероятность появления наиболее вероятного символа имеет наибольшее значение. При этом Блюм не отказывается от вышеописанного механизма LOE, применяя его совместно с новым механизмом. Помимо оригинального подхода к выбору первой оценивающей модели в алгоритме PPMZ используется особый способ получения вероятностных оценок. Формулы, в соответствии с которыми определяются вероятности появления обычных символов и символов перехода, включают в себя свободные коэффициенты, меняющиеся в процессе кодирования в зависимости от особенностей обрабатываемой информации. В совокупности указанные решения позволяют добиться очень высокой эффективности кодирования[1].

При практических реализациях метода PPM основной проблемой является обеспечение хранения большого массива статистической информации.

[1] Однако существуют и более эффективные реализации метода PPM (см. [10, 22]).

Для того, чтобы пояснить, насколько серьезна данная проблема, приведем лишь один пример: если включать в рассмотрение только модели первого порядка, для хранения статистической информации в формате, в котором под каждый счетчик появлений символа в контексте отводится ровно один байт, может потребоваться до 64 килобайт памяти (здесь подразумевается, что информационный алфавит – 256-символьный). Для избежания переполнения памяти естественно наложить ограничение на количество поддерживаемых счетчиков, то есть фактически ограничить количество контекстов, принимаемых во внимание при производстве вероятностных оценок. Также имеет смысл ограничить и кодовое пространство, отводимое под каждый счетчик. При этом при переполнении счетчиков целесообразно прибегать к *масштабированию* (*scaling*), то есть к делению на константу содержимого счетчиков появления символов в контекстах. Кроме своего прямого предназначения, масштабирование выполняет и другую полезную функцию – оно привносит дополнительный элемент адаптивности в информационную модель путем повышения роли недавно собранной статистической информации.

Помимо обычного масштабирования на практике часто используется так называемое *выборочное масштабирование*. Работа данного механизма заключается в умножении на константу содержимого счетчика какого-либо символа или группы символов. Выборочное масштабирование позволяет увеличивать оценочные вероятности появления отдельных символов, что при умелом использовании может стать действенным инструментом повышения эффективности кодирования. Примером такого использования является увеличение содержимого счетчика, соответствующего символу, который в текущем оценочном контексте появлялся в последний раз (существует большая вероятность того, что он появится в данном контексте и на этот раз).

Для хранения накапливаемой статистической информации в практических реализациях метода PPM в большинстве случаев используются деревья цифрового поиска и всевозможные хэш-структуры, позволяющие компактно представлять данные и производить быстрый поиск контекстов (см., например, [35, 44]). Выбор конкретной структуры целиком зависит от реализуемого метода. Например, для методов, в которых порядок моделей не фиксируется, рекомендуется брать в качестве такой структуры дерево цифрового поиска PATRICIA [2, 24, 57].

В заключение рассмотрения метода PPM приведем его краткую характеристику. Метод PPM является методом экономного кодирования, реально используемым на практике. Он весьма эффективен и допускает быстрые реализации. К недостаткам большинства вариаций метода PPM стоит отнести несовершенство способа оценки вероятностей и повышенную требова-

тельность к ресурсам оперативной памяти, от объема которых во многом зависит производительность метода.

### 5.1.2. Метод CTW

Метод PPM был предложен в качестве простого решения проблемы смешивания в контекстном моделировании. В результате мы получили совокупность из множества достаточно нетривиальных, громоздких и не всегда оправданных механизмов повышения эффективности кодирования. Смешивание на основе весов в сравнении с вышеописанной методикой выглядит куда более естественно. К сожалению, ограниченность машинных ресурсов делает неприменимыми большинство методов, использующих этот подход. Одним из немногих реализуемых на практике методов является метод CTW.

Рассмотрим случай кодирования двоичного информационного источника. Любая строка *s*, поступающая с выхода такого источника, представляет собой последовательность символов $q_{1-l}q_{2-l}\ldots q_0$, где $q_{-i}\in\{0,1\}$ для $i=0,1,\ldots,l-1$. Определение вероятности появления символов в различных контекстах возможно с использованием так называемых *суффиксов*.

Строка $s'=q'_{1-l'}q'_{2-l'}\ldots q'_0$ называется суффиксом строки $s=q_{1-l}q_{2-l}\ldots q_0$, если $l\geq l'$ и $q_{-i}=q'_{-i}$ для $i=0,1,\ldots,l'-1$ (рис. 5.1). Для моделирования поведения информационного источника обычно выбирается некоторое множество суффиксов, отвечающее следующему требованию: множество должно содержать ровно один суффикс любой строки длины *D*, порождаемой источником. Каждому суффиксу множества ставится в соответствие параметр $\theta$, равный вероятности появления единицы в контексте данного суффикса. Вероятность появления любого двоичного символа в текущем контексте может быть легко оценена с использованием параметра $\theta$, соответствующего суффиксу данного контекста.

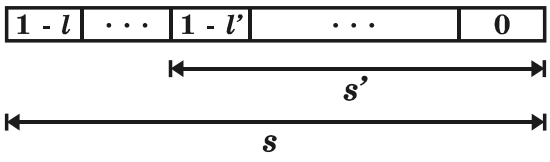



Рис. 5.1. Строка *s'* - суффикс строки *s*

Множество суффиксов принято называть *моделью*, а множество всех моделей, используемых для описания источника с памятью порядка $D$ – *модельным классом* $C_D$. Модельный класс $C_D$ можно представить в виде совокупности всевозможных бинарных деревьев с пронумерованными ребрами, имеющих не более чем $D+1$ уровень (если учитывать уровень, на котором находится корневой узел). Деревья в данном случае эквивалентны различным моделям (суффиксы, принадлежащие моделям, соответствуют маршрутам, ведущим от листовых узлов дерева к его корневому узлу).

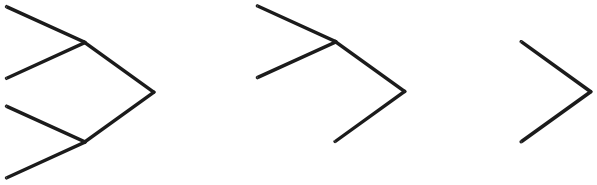


Рис. 5.2. Бинарные деревья, соответствующие представителям класса $C_2$ (нумерация ребер может быть осуществлена произвольным образом)

Оценка с использованием суффикса всегда производится на основе некоторой модели. Для оценки необходимо выбрать модель из соответствующего класса, наиболее адекватную кодируемому источнику. Ясно, что выбор некоторой конкретной модели почти всегда является заведомым ограничением, поскольку обычно ни одна из моделей полностью не отражает свойства моделируемого источника. Метод CTW представляет собой расширение описанного подхода, свободное от указанного ограничения. Рассмотрим вкратце его основную идею (подробнее с методом CTW можно ознакомится в [64, 65, 69, 70, 71, 72]).

Аббревиатура CTW (Context-Tree Weighting) означает «взвешивание с применением контекстных деревьев». Как следует из названия метода, для оценки вероятности в нем используются структуры, именуемые *контекстными деревьями*. Контекстным деревом порядка $D(T_D)$ называется бинарное дерево с пронумерованными ребрами, содержащее $D+1$ полностью заполненных уровней, каждому узлу $s$ которого соответствуют счетчики $a_s$ и $b_s$. Значение счетчика родительского узла должно быть равно сумме значений одноименных счетчиков дочерних узлов. Каждому узлу $s$ контекстного дерева присваивается вес $P_w^s$, определяемый в соответствии с рекурсивной формулой

$$P_w^s = \gamma_{l(s)} \cdot P_e(a_s, b_s) + (1-\gamma_{l(s)}) \cdot P_w^{0s} \cdot P_w^{1s},$$

где $\gamma_{l(s)}$ – некоторые коэффициенты, зависящие от уровня узла $l(s)$, такие что

$$\gamma_{l(s)} \in [0,1],\ \gamma_D = 1,$$

а $P_e(a_s, b_s) = \int_0^1 \frac{1}{\pi\sqrt{(1-\theta)\theta}} (1-\theta)^a \theta^b d\theta$ – оценка Кричевского-Трофимова [38][1].

В результате работы процедуры присвоения весов контекстное дерево становится *взвешенным*.

Дерево называется контекстным, потому что его узлы соответствуют тем или иным контекстам – тем или иным бинарным последовательностям, определяющим маршрут от некорневого узла к корневому узлу. Контекстное дерево $T_D$ содержит все возможные контексты двоичных символов порядка, не превышающего $D$.

Для получения оценки вероятности появления двоичной последовательности $x_1 x_2 \ldots x_T$ на выходе информационного источника с памятью порядка $D$ после последовательности $x_{1-D} x_{2-D} \ldots x_0$, в методе CTW используется контекстное дерево порядка $D$. Счетчик $a_s$ любого его узла $s$ содержит число нулей, появляющихся в последовательности $x_1 x_2 \ldots x_T$ в контексте, который соответствует узлу $s$, а счетчик $b_s$ – число единиц. К примеру, если $x_{1-D} x_{2-D} \ldots x_0 = 110$, а $x_1 x_2 \ldots x_T = 0101101100$, то в счетчиках узла, которому соответствует контекст «110», будут содержаться значения 2 и 1 (контекстное дерево с содержимым счетчиков для данного примера приведено на рис. 5.3). Оценкой вероятности появления последовательности $x_1 x_2 \ldots x_T$ служит вес корневого узла взвешенного контекстного дерева, который зависит от весов всех узлов этого дерева. Таким образом, в результирующей оценке одновременно учитывается вклад контекстных моделей разного порядка ($\le D$), то есть фактически осуществляется их взвешивание.

[1] Возможно использование модифицированной оценки (см. [64]).

**Контекстное дерево для последовательности «...110|0101101100»:**

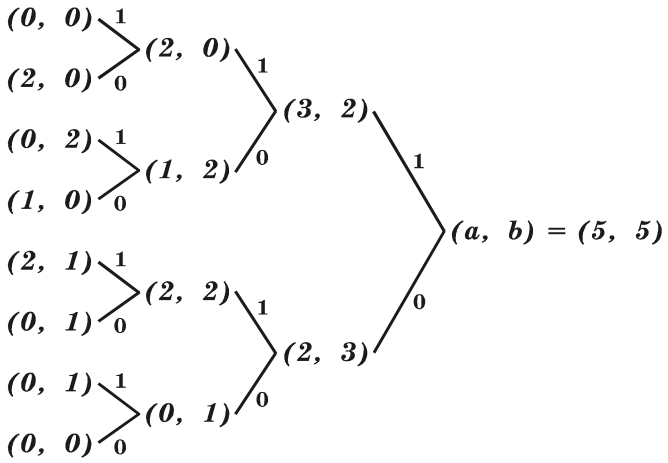



Рис. 5.3. Контекстное дерево без весов ($D = 3$)

В процессе работы метода CTW совсем не обязательно осуществлять полный пересчет параметров контекстного дерева на каждом этапе кодирования. Как оказывается, существует возможность последовательного уточнения этих параметров. Данная возможность обусловливается следующим важным свойством оценки Кричевского-Трофимова:

$$P_e(a+1,b) = \frac{a+1/2}{a+b+1} \cdot P_e(a,b),\ P_e(a,b+1) = \frac{b+1/2}{a+b+1} \cdot P_e(a,b).$$

Как видно из приведенных формул, процесс вычисления оценок Кричевского-Трофимова можно существенно упростить за счет использования результатов предыдущих вычислений.

Достоинством метода CTW является совместимость с методом арифметического кодирования. Поэтапное уточнение веса корневого узла, выступающего в роли вероятностной оценки, означает последовательное получение длины конечного интервала в методе арифметического кодирования. Таким образом, можно полностью или частично отказаться от использования при реализации арифметического кодирования операций умножения и деления, потребность в которых как раз и связана с необходимостью вычисления этой длины, ограничившись только операциями сложения и сдвига.

Хотя метод CTW предназначен для обработки двоичной информации, с помощью несложных приемов его можно адаптировать и к информации, представленной иным набором символов. Для этого достаточно преобразо-

вать исходный информационный алфавит в новый двоичный алфавит с учетом специфики обрабатываемой информации. Вариант такого преобразования подробно рассматривается в [64] в связи с проблемой обработки текстовой информации.

Изложенный метод представляет собой одну из возможных альтернатив описанному в предыдущем пункте методу PPM. Применение смешивания моделей на основе весов, как и ожидалось, оказалось вполне оправданным решением. Метод CTW обладает очень высокой эффективностью, что подтверждается как теоретически [69], так и экспериментально [64, 65]. К сожалению, существующие немногочисленные практические реализации этого метода пока что не позволяют объективно судить о его производительности.

### 5.1.3. Метод DMC

Метод DMC (Dynamic Markov Compression – «динамическое марковское сжатие», [26, 33]) заметно отличается от двух рассмотренных статистических методов экономного кодирования. Метод тесно связан с теорией конечных автоматов и использует *моделирование с конечным числом состояний* (*finite-state modeling*).

Напомним, что модель с конечным числом состояний, также именуемая марковской моделью, представляет собой систему, состоящую из конечного множества предполагаемых состояний моделируемого информационного источника и набора вероятностей всевозможных переходов между этими состояниями. Каждое сообщение, поступающее с выхода моделируемого источника, соответствует некоторому переходу из текущего состояния модели в ее новое состояние[1]. (Например, в ASCII-последовательности каждый символ определяет переход модели в одно из 128 новых состояний.) Наличие переходных вероятностей позволяет легко адаптировать марковские модели к задачам теории экономного кодирования, что, однако, компенсируется чрезмерной сложностью процесса построения таких моделей.

[1] Заметим, что модель состояний можно рассматривать в качестве своего рода обобщенной статистической модели, так как любая статистическая модель представима в виде совокупности конечного числа состояний, соответствующих различным информационным контекстам (вероятности переходов между этими состояниями определяются как оценки вероятностей появления символов в контекстах). В данном пункте речь идет о так называемых «явных» моделях состояний, в которых состояния и вероятности переходов представлены в явном виде.

Выбор структуры и параметров модели, на основе которой должна осуществляться генерация кода, сопряжен с серьезными трудностями. Помимо этого проблемы вызывает и поддержание большого числа состояний, и определение вероятностей переходов между состояниями, и изменение конфигурации модели. Следует признать, что метод DMC является далеко не оптимальным решением в моделировании с конечным числом состояний, однако он обладает рядом существенных преимуществ, которые делают его применимым на практике.

Так же как и метод CTW, метод DMC рассматривается исключительно для двоичных информационных сообщений, что, однако, как и в случае с методом CTW, не препятствует распространению метода на информационные сообщения с произвольным алфавитом. Модель в методе DMC представляет собой связный ориентированный граф, каждая вершина которого соответствует определенному состоянию (рис. 5.4). Из любой вершины графа выходят ровно две дуги, ведущие в две другие вершины. Эти дуги обозначают переходы из исходного состояния (исходная вершина) в два новых состояния (две новые вершины). Каждый переход соответствует определенному значению бинарного символа: «0» или «1». Во время работы метода DMC производится последовательный обход графа по маршруту, определяемому обрабатываемой двоичной последовательностью. Кодирование всегда осуществляется с учетом текущего состояния, то есть с учетом достигнутой на данный момент вершины графа.

Вероятность перехода из состояния в состояния в методе DMC определяется на основе статистики переходов. Для сбора этой статистики каждой дуге графа ставится в соответствие счетчик, содержимое которого увеличивается на единицу при каждом прохождении дуги во время обхода графа в процессе обработки информации.

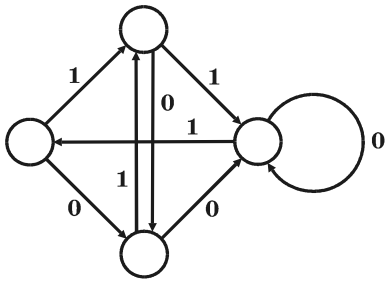



Рис. 5.4. Модель состояний

Необходимость адаптации модели к обрабатываемой информации требует осуществления изменений в ее структуре. Для перестройки модели в

методе DMC используется операция *клонирования состояний*. Если переход в одно из состояний становится достаточно популярным (т. е. содержимое счетчика перехода превышает некоторое наперед заданное число $N_1$, а суммарное содержимое счетчиков всех остальных переходов в это состояние превышает наперед заданное число $N_2$), то к графу добавляется вершина, соответствующая новому состоянию (клонирование исходного состояния), и две дуги, исходящие из нее. Эти дуги ведут в те же вершины, в которые ведут дуги, исходящие из клонируемой вершины. Кроме того, дуга, соответствующая популярному переходу в клонируемую вершину, перенаправляется в новую вершину (рис. 5.5).


Рис. 5.5. Клонирование состояния x, вызванное популярностью перехода $\overrightarrow{ax}$

При клонировании изменяются значения счетчиков некоторых переходов. В ситуации, приведенной на рис. 5.5, изменения производятся в соответствии с формулами

$$c(x', y) = c(a, x) \cdot \frac{c(x, y)}{c(x, y) + c(x, z)}, \quad c(x', z) = c(a, x) \cdot \frac{c(x, z)}{c(x, y) + c(x, z)},$$

$$c(x, y) = c(x, y) - c(x', y), \quad c(x, z) = c(x, z) - c(x', z),$$

$$c(a, x') = c(a, x),$$

где $c(i, j)$ – значение счетчика перехода из состояния $i$ в состояние $j$.

Выбор коэффициентов $N_1$ и $N_2$, в принципе, должен зависеть от начальной модели. Кроме того, целесообразно было бы менять значения этих коэффициентов в процессе кодирования. Однако на практике часто идут по

более простому пути, используя в качестве коэффициентов $N_1$ и $N_2$ эмпирически полученные значения 1 и 1.

При реализации метода DMC необходимо очень аккуратно подходить к выбору начальной модели, так как начальная модель предопределяет будущую конфигурацию графа переходов. Наиболее простым и достаточно неоптимальным решением является выбор начальной модели с одним состоянием (граф с одной вершиной и двумя петлями). С целью достижения большей эффективности необходимо руководствоваться при выборе модели априорными сведениями об обрабатываемой информации.

Благодаря предельной простоте метод DMC обладает очень высокой производительностью. К сожалению, это не верно в отношении его эффективности. Одной из причин недостаточной эффективности является ограничение числа состояний в информационной модели, вводимое в практических реализациях метода. (Такое ограничение, безусловно, необходимо, так как поддержание моделей с большим числом состояний сопряжено со значительным расходом оперативной памяти.) Однако существует и более существенная причина, обусловливающая малую эффективность метода DMC. Эта причина, как ни странно, заключается в несовершенстве информационной модели. Практика показывает, что модель состояний, используемая в методе DMC, недостаточно пригодна для описания свойств реальных информационных источников.

### 5.1.4. Экономное кодирование, основанное на использовании искусственных нейронных сетей

Работа статистических методов экономного кодирования сводится к предсказанию поведения информационных источников. Описанные в предыдущих пунктах методы весьма эффективны по части такого предсказания. Однако, как показывают опыты (см. [3, 9]), они все же не могут соперничать с аналогичной способностью человека. Установлено, к примеру, что человек значительно чаще правильно подбирает по началу фразы ее продолжение.

С одной стороны, причина превосходства человека состоит в априорных знаниях о языке, то есть, выражаясь строго, человек априори владеет информационной моделью языка. С другой стороны, человеческий мозг, по всей вероятности, использует особый метод предсказания, совершенно не похожий на рассмотренные методы. Первый аспект превосходства человека над статистической схемой предсказания не заслуживает особого внимания, ибо, в принципе, любая статистическая модель, построенная на основе достаточно большого объема накопленной статистической информации, может иметь параметры, очень схожие с реальными параметрами рассматриваемого языка. Однако модель может неадекватно отражать свойства моделируемого объекта. Здесь как раз вступает в силу второй аспект. Не вызывает со-

мнений тот факт, что многие существующие информационные источники в той или иной форме являются результатом человеческой деятельности. Как следствие, человек достаточно точно предсказывает их поведение. Ясно, что, если мы хотим добиться от статистического метода гарантированного достижения «человеческого» уровня предсказания, необходимо, чтобы это предсказание осуществлялось тем же способом, который использует человек. Для этого метод предсказания, очевидно, должен иметь в своей основе модель, схожую с моделью человеческого мозга.

На сегодняшний день наиболее точной математической моделью человеческого мозга является искусственная нейронная сеть. Нейронные сети весьма успешно используются при решении многих проблем, связанных с моделированием и принятием решений, в частности, они находят свое применение в области экономного кодирования. Рассмотрим наиболее общую схему организации экономного кодирования на основе искусственной нейронной сети.

Для осуществления экономного кодирования информации чаще всего используется многослойная нейронная сеть, имеющая по одному нейрону выходного слоя на каждый символ информационного алфавита (рис. 5.6). С выходов этих нейронов поступают сигналы, соответствующие оценкам вероятностей появления различных символов в текущем контексте. Текущий контекст «вводится» в сеть через нейроны входного слоя. Одним из возможных способов организации этого слоя является поддержание $K$ групп нейронов, каждая из которых содержит $N$ нейронов (здесь $K$ равно порядку вводимого контекста, а $N$ – количеству символов в информационном алфавите). $i$-я нейронная группа соответствует символу, находящемуся на $i$-й позиции в контексте. При вводе этого символа на вход одного из $N$ нейронов $i$-й нейронной группы поступает 1, а на входы остальных нейронов – 0. Таким образом, при вводе очередного контекста на нейроны входного слоя поступает $K$ единиц и $K \cdot (N-1)$ нулей.

Нейронная сеть может содержать до нескольких скрытых слоев (см. рис. 5.6). Как правило, это один или два слоя, хотя, в идеале, количество скрытых слоев должно меняться в зависимости от свойств обрабатываемого информационного источника. Для нейронов скрытых слоев в качестве активационных функций целесообразно использовать сигмоидальные функции. Для нейронов выходного слоя вместо сигмоидальных функций рекомендуется брать нормированные экспоненциальные функции [39]. При обучении сети, построенной в соответствии с описанными принципами, вполне естественно применять алгоритм обратного распространения.

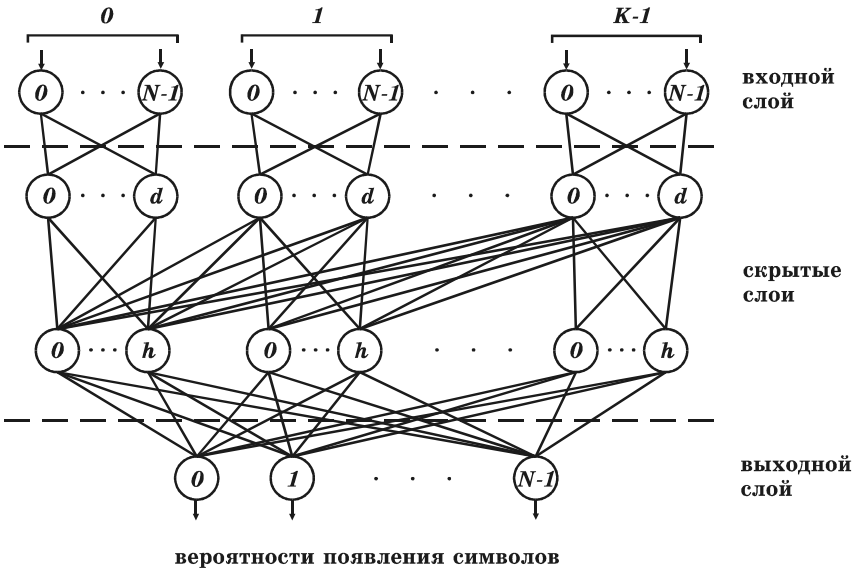



Рис. 5.6. Четырехслойная нейронная сеть, используемая в методе PSM

Использование нейронных сетей для оценки вероятностей появления символов на выходе информационного источника в целом нельзя назвать оригинальным подходом. В действительности мы имеем дело с еще одним воплощением идеи смешивания моделей на основе весов. Безусловным преимуществом данного решения является возможность адаптации информационной модели к обрабатываемой информации с использованием фиксированного объема памяти (число нейронов всегда фиксировано). Недостаток заключается в низкой производительности, что связано с трудоемкостью процесса обучения сети.

Несмотря на высокую популярность искусственных нейронных сетей, для целей экономного кодирования информации они применяются относительно недавно. Наиболее известными на сегодня методами экономного кодирования, в которых используются нейронные сети, являются методы NN [56] и PSM [39]. Они обладают высокой эффективностью (в особенности метод PSM), вполне сравнимой с эффективностью рассмотренных статистических методов, но, являясь недостаточно производительными, вряд ли в скором времени будут применяться на практике.

### 5.1.5. Об эффективности статистических методов

Оценить эффективность того или иного метода экономного кодирования не так просто. Оценку можно дать либо с позиций теории, путем исследования асимптотического поведения метода в рамках определенной информационной модели, либо экспериментально, исходя из результатов применения метода к некоторому выбранному множеству информационных ис-

точников. Первый подход является, естественно, более строгим, однако он, как правило, оказывается совершенно бесполезным ввиду того, что реальные источники информации в большинстве своем не соответствуют идеализированным информационным моделям.

Недостаток второго подхода заключается в его ограниченной репрезентативности. Совершенно очевидно, что невозможно подвергнуть исследованию все существующие источники информации, а те из них, которые все же удастся изучить, наверняка не будут обладать всеми особенностями неизученных источников. Тем не менее использование данного подхода является, вероятно, единственным способом оценки реальной эффективности метода. Количественное сравнение методов экономного кодирования принято проводить на файлах стандартных наборов (корпусов) Calgary [78] и Canterbury [79]. Файлы наборов представляют собой наиболее распространенные типы данных (текст, графика, исполняемые файлы и т.д.). Несмотря на кажущуюся полноту, данная выборка все же остается достаточно нерепрезентативной.

Другая проблема связана с конкретными реализациями методов экономного кодирования информации. Не секрет, что эффективность метода во многом зависит от доступного объема оперативной памяти, способа кодирования, выбора начальной модели и многих других факторов. Поэтому высказывать объективные суждения об эффективности можно только в отношении конкретных алгоритмов, при строго определенных условиях их применения.

Если говорить об оценке производительности, то здесь, помимо вышеуказанных факторов, надо также учитывать и оптимальность машинной реализации того или иного алгоритма. По-видимому, объективно такое сравнение можно провести только теоретически.

Однако, вернемся к статистическим методам. Нужно объективно признать, что статистические методы, как правило, обладают наибольшей эффективностью кодирования. Особенно наглядно это можно продемонстрировать на примере текстовой информации. При использовании практических реализаций методов DMC [26], PPMC [22], PPMD [22], CTW [65] и алгоритма PPMZ2 [17] средняя эффективность представления текстовых файлов book1 и book2, принадлежащих набору Calgary, оказывается равной 2.38, 2.17, 2.13, 2.01 и 2.02 бит/сим соответственно. Для сравнения, наилучший результат, получаемый с использованием широко распространенной утилиты сжатия информации GZIP, составляет 2.98 бит/сим.

Если явным достоинством статистических методов является высокая эффективность, то столь же явный недостаток заключается в повышенных требованиях к ресурсам вычислительной системы. Ограниченность этих ре-

сурсов может стать причиной как недостаточной эффективности, так и крайне низкой производительности.

Затрагивая тему производительности статистических методов, нельзя обойти стороной одно отрицательное свойство, присущее всем статистическим методам. Благодаря характеру адаптивности скорость кодирования у статистических методов почти совпадает со скоростью декодирования. Это означает, что время, потраченное на получение информационного представления, приблизительно эквивалентно времени интерпретации данного представления. С учетом того, что во многих случаях нам хочется иметь «легко читаемое» информационное представление, использование статистических методов часто оказывается совершенно неприемлемым решением.

Таким образом, о статистических методах экономного кодирования информации можно говорить как о методах, наиболее выгодных с точки зрения эффективности, но обладающих рядом недостатков, препятствующих их использованию на практике. Дальнейшее развитие вычислительной техники, по всей видимости, приведет к изменению этого положения.

## 5.2. Словарные методы

В предыдущем изложении информационные закономерности рассматривались как чисто вероятностные. Альтернативная трактовка подразумевает использование так называемого *комбинаторного подхода*. В новой трактовке проявлением информационных закономерностей является присутствие в информации одних символов или символьных комбинаций и полное отсутствие других. Заметим, что в данном случае мы не говорим о вероятностных межсимвольных взаимосвязях, а просто констатируем факт наличия или отсутствия отдельных символьных цепочек в информации. Почему возможна такая замена терминологии? Потому что речь идет всего лишь об информационных моделях. Выбор конкретной модели есть, очевидно, не более чем формальное соглашение.

Переход от вероятностного подхода к комбинаторному неизбежно приводит к изменению содержания некоторых базовых понятий теории экономного кодирования. В частности, меняется определение энтропии информационного источника. В рамках комбинаторного подхода энтропию логично определять как отношение логарифма числа всевозможных символьных комбинаций, встречающихся на выходе источника информации, к средней длине одной комбинации. Следствием такого определения энтропии становится принципиально иной способ организации экономного кодирования. Кодирование в рамках комбинаторного подхода основывается на построении кодовых систем, содержащих коды только тех информационных последовательностей, которые реально порождаются информационным ис-

точником[1]. Методы, в явном виде представляющие указанный подход, – методы словарной группы, впервые описанные в знаменитых работах А. Лемпела и Я. Зива [75, 76].

Словарные методы используют словарные модели, основу которых составляет информационная структура, именуемая *словарем*. Во время работы словарного метода словарь включает в себя части уже обработанной информации, выступающие в качестве материала, на основе которого осуществляется кодирование. В процессе кодирования составляющие символьной последовательности, поступающей с выхода кодируемого источника, кодируются посредством ссылок на идентичные им элементы словаря (совпадения). Словарные методы отличаются друг от друга способом организации словаря, схемой поиска совпадений и видом ссылки на найденное совпадение.

Как следует из описанной схемы, работа любого словарного метода сопряжена с разбиением обрабатываемой информационной последовательности на составляющие блоки, идентичные некоторым блокам в уже обработанной части информации. Здесь, естественно, возникают два вопроса: как выбирать наилучшее разбиение и что делать в случае, когда разбиение вообще невозможно причине отсутствия совпадений? Попытаемся по отдельности дать ответ на каждый из этих вопросов.

Разбиение обрабатываемой последовательности на кодируемые составляющие представляет собой трудноразрешимую задачу, от оптимальности решения которой в высокой степени зависит эффективность словарного метода. Рассмотрим эту проблему на конкретном примере. Предположим, что необходимо обработать последовательность символов «abac», и при этом известно, что словарь содержит четыре строки: «a», «ab», «c», «bac». Очевидно, что последовательность можно разбить на составляющие двумя различными способами: «abac» = «ab» + «a» + «c» или «abac» = «a» + «bac». В основе первого способа, называемого *«жадным» разбором* (*greedy parsing*), лежит выбор наиболее длинного совпадения при последовательном разбиении последовательности слева направо. Второй способ, именуемый *оптимальным разбором* (*optimal parsing*), заключается в нахождении наилучшего разбиения из всех возможных. Из-за колоссальной вычислительной сложности оптимальный разбор не используется на практике. Вместо него в большинстве случаев используется «жадный» разбор, который легко реализуем и, кроме того, удобен при последовательной обработке информации. Однако «жадный» разбор не оптимален с точки зрения эффек-

[1] Количество таких кодов будет меньше количества кодов, необходимых для описания всех возможных информационных последовательностей, получаемых путем произвольного комбинирования символов информационного алфавита. Отметим, что данная идея в несколько упрощенном виде уже фигурировала выше (см. гл. 1).

тивности. Для ее повышения рекомендуется производить поиск совпадений, начиная не только с первой обрабатываемой позиции, но и нескольких следующих позиций. Совпадение следует выбирать, руководствуясь соображениями оптимальности длины получаемой кодовой комбинации. Данный подход принято называть *ленивым поиском* или *ленивым сопоставлением* (*lazy matching*).

С целью рассмотрения второго вопроса несколько изменим условия предыдущего примера. Пусть словарь содержит не четыре строки, а три: «a», «ab», «bac». В этом случае обрабатываемая последовательность «abac» уже не может быть разбита на составляющие с использованием «жадного» разбора: после частичного разбиения «aba» = «ab» + «a» возникает проблема дальнейшего разбиения последовательности (в новых условиях в словаре отсутствует строка, состоящая из одного символа «c»). Существует несколько вариантов решения данной проблемы, среди которых следует выделить два основных подхода. Первый связан с введением дополнительной кодовой комбинации, указывающей на отсутствие совпадения и, следовательно, на невозможность кодирования с использованием словарной ссылки. В процессе декодирования данная кодовая комбинация сигнализирует о необходимости интерпретации следующего за ней кода не как ссылки на совпадение, а некоторым иным образом, например, код может интерпретироваться как информация в ее исходном (незакодированном) представлении. Второй подход подразумевает обязательное хранение в словаре элементов, обеспечивающих частичное совпадение с любой кодируемой последовательностью. Обычно в качестве обязательного набора дополнительных элементов берутся символы информационного алфавита.

Словарные алгоритмы являются вычислительно менее сложными и, соответственно, более быстрыми в сравнении с реализациями рассмотренных методов статистической группы. (Наиболее трудоемкой операцией в словарном алгоритме является поиск совпадений в словаре). Словарные алгоритмы не обладают эффективностью статистических алгоритмов, однако получаемый результат, как правило, вполне приемлем, особенно с учетом непритязательности словарных алгоритмов к ресурсам вычислительной системы.

За время, прошедшее с появления первого словарного алгоритма LZ77, было предложено не менее двух десятков других словарных алгоритмов. Ниже будут рассмотрены те из них, которые отражают наиболее важные направления в развитии словарных методов экономного кодирования информации.

### 5.2.1. Алгоритмы семейства LZ77

Алгоритм LZ77 был описан А. Лемпелем и Я. Зивом в [75] и получил свое название по первым буквам фамилий авторов и году публикации. С тех пор традиционно аббревиатуры почти всех словарных алгоритмов начинаются с «LZ», вследствие чего словарные алгоритмы и методы часто называют *алгоритмами и методами группы LZ*. В данном пункте рассматривается алгоритм LZ77 и алгоритмы, являющихся непосредственными его производными (так называемые *алгоритмы семейства LZ77*).

В алгоритме LZ77 в качестве словаря выступают $N$ последних обработанных символов. Каждая кодовая комбинация состоит из трех компонентов (кодовая триада). Первые два компонента указывают на строку словаря[1], идентичную начальной части кодируемой символьной последовательности: один из этих компонентов является смещением строки в словаре (смещение определяется относительно текущей обрабатываемой позиции), а второй – длиной этой строки. Третий компонент представляет собой первый символ кодируемой последовательности, не принадлежащий строке, описываемой первыми двумя компонентами (символ записывается в кодовой триаде в своем исходном представлении). Длина совпадения ограничивается сверху некоторым числом $M$. В процессе работы алгоритма на каждом шаге в словаре осуществляется поиск строки, имеющей наибольшее совпадение с кодируемой символьной последовательностью. В случае, когда совпадение в словаре отсутствует, обрабатывается только первый символ последовательности[2], для чего в кодовой триаде указывается нулевое смещение (обрабатываемый символ является третьим компонентом триады).

Декодирование кода, полученного с применением алгоритма LZ77, сводится к копированию строк указанных в триаде длин, начиная с указанных смещений. После копирования очередной строки к ней добавляется символ, являющийся третьим компонентом кодовой триады. Благодаря простоте декодера алгоритм LZ77 и его производные обладают очень высокой скоростью декодирования. Процесс кодирования является существенно более медленным, что обусловливается необходимостью осуществления поиска совпадений.

Рассматриваемый алгоритм относится к адаптивным алгоритмам. Способ кодирования того или иного сообщения (в данном случае в качестве со-

---

[1] Под «строкой» здесь и в дальнейшем понимается последовательность символов ограниченной длины. Данный термин используется для проведения четкого разграничения между обработанными последовательностями ограниченной длины, составляющими словарь (ограничение накладывается на длину совпадения), и обрабатываемой последовательностью, длина которой, вообще говоря, может быть не ограничена.

[2] В дальнейшем изложении такие символы будут именоваться *незакодированными*.

общения выступает строка символов) непосредственно зависит от содержимого словаря, которое меняется в процессе работы.

Проиллюстрируем действие алгоритма LZ77 на примере. Закодируем слово «ОБОРОНОСПОСОБНОСТЬ», используя параметры $N = 7$, $M = 2$. Результатом применения алгоритма является следующая кодовая последовательность: <0,1,»О»> <0,1,»Б»> <2,1,»Р»> <2,1,»Н»> <2,1,»С»> <0,1,»П»> <3,2,»О»> <0,1,»Б»> <0,1,»Н»> <5,2,»Т»> <0,1,»Ь»>. При выбранных параметрах минимальная длина кодовой триады при условии, что исходная информация представлена символами 256-символьного алфавита, равна 12 битам. Таким образом, длина кодовой последовательности равна $12 \cdot 11 = 132$ битам, а эффективность информационного представления – $132/18 \approx 7.33$ бит/сим. Отметим, что данный результат является конечным, так как, благодаря характеру адаптивности, алгоритм не требует хранения дополнительной информации о способе кодирования.

Ограничения, накладываемые на размер словаря и длину совпадения, породили понятие *скользящее окно* (*sliding window*). Скользящее окно состоит из двух частей – задней, именуемой *буфером поиска* (*search buffer*), и передней, называемой *буфером просмотра* (*look-ahead buffer*). Кодируемая строка всегда находится внутри буфера просмотра, а идентичная ее части строка словаря имеет свое начало в пределах буфера поиска[1]. Буфер поиска и буфер просмотра имеют длины *N* и *M* соответственно, причем, как правило, *N* значительно превосходит *M*. В процессе работы алгоритма осуществляется последовательное перемещение (скольжение) окна по обрабатываемой информации. При этом каждая обработанная строка или символ переходят из передней части скользящего окна в его заднюю часть, а в переднюю часть добавляется очередная порция информации, равная по длине обработанному информационному блоку. В связи с ограничением, накладываемым на размер буфера поиска, при добавлении в него очередной строки из буфера, начиная с конца, удаляется строка, по длине идентичная удаленной строке. Составляющие удаленной строки более не являются частью словаря.

При практической реализации механизма скользящего окна весьма удобным оказывается использование циклического буфера, размер которого идентичен размеру скользящего окна. Скольжение окна в таком буфере сводится к циклическому перемещению границы между его задней и передней частями (текущей обрабатываемой позиции) в пределах буфера (см. рис. 5.7). При этом добавление новой порции информации в переднюю часть окна автоматически означает удаление идентичной по длине порции из его задней части.

[1] Идентичная строка может частично перекрывать буфер просмотра.

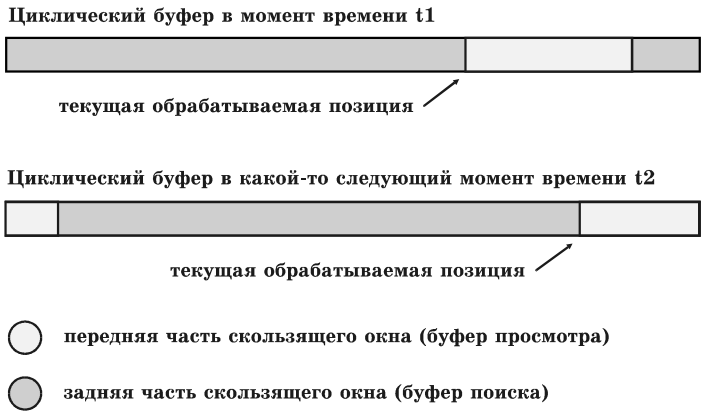



Рис. 5.7. Скользящее окно в циклическом буфере

Выбор параметров скользящего окна самым непосредственным образом влияет на эффективность кодирования. С одной стороны, увеличение размера буфера поиска означает расширение словаря и ведет к повышению вероятности нахождения идентичных строк, а увеличение размера буфера просмотра позволяет учитывать при кодировании более длинные совпадения. С другой стороны, размер кодового пространства, необходимого для хранения смещений строк и длин совпадений, возрастает с увеличением соответствующих составляющих скользящего окна по логарифмическому закону. Алгоритм LZR [52], основное отличие которого от алгоритма LZ77 заключается в наличии неограниченного буфера поиска, является недостаточно эффективным именно из-за того, что расширение возможностей поиска не всегда компенсирует издержки кодирования, обусловленные большим размером кодовых комбинаций. При реализации словарных алгоритмов необходимо правильно выбирать способ и параметры кодирования, которые, вообще говоря, должны зависеть от характеристик обрабатываемой информации.

На рассмотренном выше примере видны отдельные недостатки алгоритма LZ77. Во-первых, длина кодовой триады, кодирующей только один символ, превышает длину исходного представления этого символа. Во-вторых, никак не учитывается тот факт, что одни значения смещений во время кодирования используются чаще, чем другие, а некоторые из них вообще не используются. В-третьих, весьма спорной является необходимость включения в каждый код в качестве третьего компонента дополнительного символа. Данные недостатки были частично устранены в алгоритме LZSS.

Алгоритм LZSS был предложен Д. Сторером и Т. Сжиманским [60], а затем усовершенствован Т. Беллом [14]. Отличие данного алгоритма от алгоритма LZ77 заключается в отказе от последнего компонента кодовой

триады. Для описания совпадений в данном случае используются кодовые пары типа <смещение совпадения, длина совпадения>. Проблема отсутствия совпадений в словаре решается путем введения дополнительного служебного бита, значение которого определяет, является ли следующая за ним кодовая комбинация кодовой парой, или она представляет собой незакодированный символ в его исходном представлении. Отметим, что кодовые пары используются только в том случае, когда длина кодовой комбинации с учетом служебного бита не превышает длины найденного совпадения в словаре.

Кодовая последовательность для слова «ОБОРОНОСПОСОБНОСТЬ», полученная с применением алгоритма LZSS с параметрами $N=8$, $M=2$, выглядит следующим образом: «0»<«О»> | «0»<«Б»> | «1»<2,1> | «0»<«Р»> | «1»<2,1> | «0»<«Н»> | «1»<2,1> | «0»<«С»> | «0»<«П»> | «1»<3,2> | «1»<2,1> | «0»<«Б»> | «1»<8,2> | «1»<5,1> | «0»<«Т»> | «0»<«Ь»>. Длина кода равна $(8+1)\cdot 9+(4+1)\cdot 7=116$ битам (9 незакодированных символов и 7 кодовых пар). Эффективность информационного представления составляет $116/18\approx 6.44$ бит/сим, что, как видно, превышает результат, полученный с использованием алгоритма LZ77.

Несомненным достоинством алгоритма LZSS является предсказуемость его поведения в худшей ситуации. Эта предсказуемость заключается в том, что увеличение объема представления информации, полученное в результате ее обработки данным алгоритмом, не может превысить 1 бит/сим. На самом деле подобное заключение верно и в отношении алгоритма LZ77, однако в случае с алгоритмом LZ77 увеличение объема представления информации может оказаться более существенным.

Помимо оригинальной системы кодирования в алгоритме LZSS используется особый способ поиска совпадений в словаре. При кодировании поддерживается бинарное лексикографически упорядоченное дерево поиска, в котором каждому узлу (как внутреннему, так и листовому) соответствует определенная строка словаря длины $M$ (максимальная длина совпадения). Поиск осуществляется путем последовательного сравнения кодируемой строки со строками, соответствующими узлам дерева, проходимым при движении по дереву от корневого узла до некоторого листового узла. Направление движения всегда определяется лексикографическим отношением сравниваемых строк. В процессе кодирования дерево поиска трансформируется. Трансформация производится посредством удаления и добавления строк. Если удаление строки требует некоторых дополнительных усилий, то операция добавления предельно проста и осуществляется прямо в процессе поиска совпадений. Более подробно с алгоритмом поиска можно ознакомиться, например, в [46].

Хотя алгоритм LZSS и представляет собой значительный шаг вперед по сравнению с алгоритмом LZ77, он все же далеко не безупречен. Помимо от-

дельных элементов «неоптимальности», унаследованных от алгоритма LZ77, алгоритм LZSS обладает и собственным очевидным недостатком: при кодировании не учитываются существующие статистические закономерности в появлении значений служебных битов (в рассмотренном примере ноль встречался чаще, чем единица). Дальнейшее усовершенствование метода может быть достигнуто за счет использования для представления кодовых компонентов кодов переменной длины[1]. Коды переменной длины применяются в алгоритмах LZB [13] и LZH [21]. Более подробно этот подход будет освещен в п. 5.2.4.

### 5.2.2. Алгоритмы семейства LZ78

В 1978 году А. Лемпел и Я. Зив предложили алгоритм LZ78 [76], положивший начало еще одному семейству словарных алгоритмов.

В данном алгоритме словарь представляет собой множество пронумерованных строк различной длины. Так же, как и в алгоритмах семейства LZ77, на каждом шаге работы алгоритма LZ78 в словаре осуществляется поиск наиболее длинной строки, идентичной части кодируемой символьной последовательности. Кодовая комбинация состоит из двух частей. Первая является индексом найденной в словаре строки. В качестве второй части выступает символ, следующий за кодируемой строкой. Главное отличие алгоритма LZ78 от алгоритмов семейства LZ77 состоит в том, что длина совпадения здесь в явном виде не фигурирует как компонент кода. Это становится возможным благодаря тому, что в алгоритме LZ78, а также в других алгоритмах семейства, каждому индексу в словаре соответствует строка вполне определенной длины.

Нумерация строк в словаре начинается с первого индекса. Нулевой индекс соответствует пустой строке и зарезервирован для случая отсутствия совпадений. В процессе работы после генерации очередной порции кода только что обработанная последовательность символов добавляется в словарь с присвоением ей первого в порядке возрастания неиспользуемого индекса. Таким образом, словарь каждый раз пополняется строкой, отличающейся от некоторой уже имеющейся в словаре строки одним дополнительным символом. Данная особенность позволяет экономно представлять словарь в виде дерева, каждому узлу которого соответствует ровно один символ. Добавление строк в словарь сводится к добавлению дочернего узла к

[1] Стоит отметить, что такая модификация сопряжена со снижением производительности алгоритма, так как при его реализации неизбежно возникает потребность в применении битовых операций, которых можно избежать при использовании фиксированных кодовых пространств, кратных размеру машинного слова.

узлу, соответствующему строке, имеющей на данный момент наибольшее совпадение с кодируемой символьной последовательностью.

Кодовое пространство, отводимое в алгоритме LZ78 под хранение индекса, находится в логарифмической зависимости от количества используемых индексов. К примеру, если в словаре имеется всего три строки, то для описания совпадения, учитывая нулевой индекс, потребуется 2 бита (используются 4 индекса: 0, 1, 2 и 3). Размер второго компонента кодовой комбинации, кодирующего отдельный символ, постоянен и зависит от размера информационного алфавита.

Так как код в алгоритме LZ78 включает в себя индексы словаря, то словарь необходимо поддерживать не только на этапе кодирования, но и на этапе декодирования[1]. Как и в любом адаптивном алгоритме, в алгоритме LZ78 изменения в словаре производятся синхронно на обоих этапах. При декодировании после интерпретации очередной кодовой комбинации в словарь добавляется декодированная последовательность символов, состоящая из строки, уже имеющейся в словаре, и дополнительного символа.

Приведенная ниже таблица иллюстрирует процесс кодирования слова «ОБОРОНОСПОСОБНОСТЬ» алгоритмом LZ78. Длина кода для данного слова равна $0+1+2+2+3+3+3+3+4+4+8\cdot 10=105$ битам. Эффективность информационного представления соответственно равна $105/18\approx 5.83$ бит/сим. На рис. 5.8 отображено конечное состояние словаря (словарь представлен в виде дерева).

**Кодирование слова «ОБОРОНОСПОСОБНОСТЬ» алгоритмом LZ78**

| Код | 0, «О» | 0, «Б» | 1, «Р» | 1, «Н» | 1, «С» | 0, «П» | 5, «О» | 2, «Н» | 5, «Т» | 0, «Ь» |
|---|---|---|---|---|---|---|---|---|---|---|
| Добав-Ляемая строка | «О» | «Б» | «ОР» | «ОН» | «ОС» | «П» | «ОСО» | «БН» | «ОСТ» | «Ь» |
| Индекс | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

Алгоритм LZ78 и его производные особенно хорошо зарекомендовали себя на текстовой информации. Стоит отметить, однако, что даже применительно к текстовой информации они нередко уступают в эффективности алгоритмам семейства LZ77[2], что объясняется избирательностью при добавлении строк в словарь (частичное добавление строк снижает вероятность нахождения совпадений в процессе поиска). Неоспоримым достоинством ал-

[1] При этом совсем не обязательно использовать идентичную древовидную структуру хранения. Более того, с целью обеспечения наибольшей производительности на этапе декодирования рекомендуется заменить древовидную структуру линейной (можно, например, использовать индексированный массив).

[2] Однако если сравнивать алгоритм LZ78 с простейшими алгоритмами семейства LZ77, то предпочтение все же следует отдать алгоритму LZ78.

горитмов семейства LZ78 является очень высокая скорость кодирования. Быстрый поиск совпадений становится возможным благодаря ограниченности словаря (словарь содержит далеко не все обработанные строки) и использованию для его хранения древовидной структуры. Что касается декодирования, то, хотя оно и является более быстрой операцией, чем операция кодирования, разница оказывается не такой значительной, как в случае с алгоритмами семейства LZ77[1]. Поэтому алгоритмы семейства LZ78 более выгодны с точки зрения скорости получения информационного представления и менее выгодны с точки зрения скорости его интерпретации.

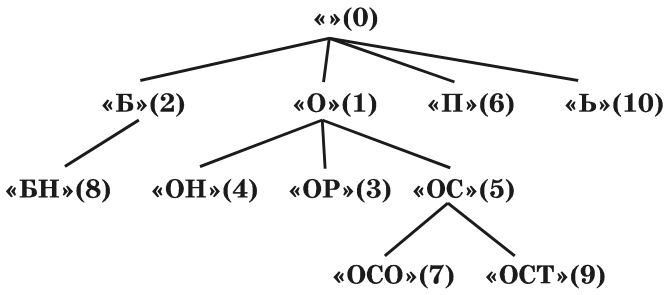



Рис. 5.8. Состояние словаря после обработки слова «ОБОРОНОСПОСОБНОСТЬ» (в скобках указаны индексы строк)

Как нетрудно заметить, алгоритм LZ78 имеет тот же недостаток, что и алгоритм LZ77 – обязательное наличие символа в каждой кодовой комбинации подчас выглядит совершенно необоснованно. Альтернативный способ кодирования был реализован Т. Уэлчем в алгоритме LZW [68][2]. Идея состоит в изначальном включении в словарь всех символов информационного алфавита. Это дает возможность кодировать информацию с использованием только индексного кодового компонента.

Обновление словаря в алгоритме LZW требует некоторых дополнительных усилий. При кодировании с использованием данного алгоритма каждый раз в словарь добавляется новая строка, последний символ которой принадлежит еще не закодированной части информации (это обстоятельство обусловлено отказом от включения в кодовую комбинацию дополнительного символа). На соответствующем этапе декодирования этот символ неизвестен. Поэтому добавление строки в процессе декодирования необходимо откладывать до момента, когда данный символ будет получен как первый символ очередной декодированной строки. В связи с этим возникает проблема, связанная с возможностью использования при кодировании строки

[1] Недостаточно высокая скорость декодирования в алгоритмах семейства LZ78 связана с поддержкой на этапе декодирования механизма индексирования.

[2] Независимо от Уэлча данный алгоритм предложили В. Миллер и М. Вегман [42].

индекса предыдущей закодированной строки. Она заключается в том, что первый символ новой строки определяется по соответствующему индексу как часть еще не добавленной строки, добавление которой производится только на текущем шаге и на первый взгляд невозможно без знания данного символа. Возникающее затруднение, как оказывается, легко разрешимо. Достаточно заметить, что неизвестный символ является первым символом предыдущей декодированной строки.

Высокая скорость работы и приемлемая эффективность алгоритма LZW позволяют использовать его во многих высокопроизводительных приложениях, требующих осуществления экономного кодирования. Наиболее известные области применения данного алгоритма – эффективное представление графических изображений (форматы CompuServe GIF и TIFF) и экономное кодирование при передаче информации по линиям связи (коммуникационный стандарт V.42bis[1]).

В предыдущем изложении пока ничего не говорилось о размере индексированного словаря, хотя совершенно очевидно, что его выбор представляет собой определенную проблему. Не секрет, что хранение большого количества фраз, даже с использованием компактной структуры дерева, требует больших объемов оперативной памяти. При ограниченных ресурсах памяти постоянное обновление словаря может очень быстро привести к ее переполнению. Кроме того, так же, как и в алгоритмах семейства LZ77, в рассмотренных алгоритмах LZ78 и LZW от объема словаря напрямую зависит размер кодового пространства, который может стать неоправданно большим.

Для того, чтобы избежать неприятных ситуаций, проще всего как-то ограничить размер словаря. Технически это можно осуществить следующим образом: при достижении словарем некоторого максимально допустимого размера необходимо прибегнуть к его полной очистке и вести последующую обработку информации, начиная с пустого словаря. Ограничение размера словаря в алгоритмах рассматриваемого семейства осуществляется путем наложения ограничения на количество индексов, что в конечном счете можно рассматривать как ограничение количества строк в словаре. Как правило, максимальное количество индексов выбирается равным степени основания системы кодирования. В этом случае очистка словаря производится только после полного исчерпания некоторого отведенного кодового пространства. В большинстве практических приложений кодовое пространство ограничивается 12 битами.

Особого внимания в связи с проблемой ограничения размера словаря заслуживает алгоритм LZC [62], представляющий собой модификацию алго-

[1] В действительности в стандарте V.42bis [77] описывается алгоритм, несколько отличный от оригинального алгоритма LZW.

ритма LZW. В данном алгоритме при достижении предельного размера словаря (размер задается предварительно) мгновенная очистка не производится и откладывается до момента, когда использование словаря станет неэффективным. Таким образом, на некоторых этапах обработки информации алгоритм LZC становится неадаптивным алгоритмом (используется фиксированный словарь). В сравнении с алгоритмом LZW алгоритм LZC является более выгодным с точки зрения эффективности. Алгоритм LZC лежит в основе утилиты сжатия информации COMPRESS, включаемой в поставку операционных систем семейства UNIX.

Можно ли считать подход, используемый в алгоритме LZC, оптимальным? Вероятно, нет. При пополнении словаря в словарном алгоритме происходит накопление информации о кодируемом источнике. Если свойства источника со временем не меняются, то для кодирования можно использовать однажды полученный словарь. В противном случае необходимо модифицировать словарь в процессе кодирования. В алгоритме LZC применяется слишком радикальный подход к модификации словаря, заключающийся в его полной очистке. Такое решение вполне приемлемо только в случае резких изменений статистических характеристик источника. В реальных источниках изменения часто происходят более плавно, что приводит к необходимости осуществления таких же плавных изменений и в словаре. Эти изменения могут осуществляться, например, путем добавления в словарь новых строк взамен строк, наиболее редко используемых при кодировании (так называемое *LRU-замещение*). Данное решение легло в основу алгоритма LZT П. Тишера [63]. Следует отметить, что помимо оригинального способа обновления словаря, в алгоритме используется и несколько иной способ кодирования совпадений.

Правило добавления строк, применяемое в рассмотренных алгоритмах семейства LZ78, весьма спорно. С точки зрения производительности данное решение обоснованно, однако оно не всегда обоснованно с точки зрения эффективности. В качестве альтернативы можно предложить механизм добавления строк, используемый в алгоритме LZMW В. Миллера и М. Вегмана [42]. Во время работы данного алгоритма на каждом шаге кодирования в словарь добавляется строка, составленная из двух последних закодированных строк. Таким образом, в отличие от рассмотренных алгоритмов, алгоритм LZMW оперирует не одиночными символами, а целыми строками. Алгоритм LZMW отличается от других алгоритмов семейства LZ78 большей сложностью и более высокими требованиями к объему оперативной памяти, однако при этом он обладает и большей эффективностью.

Алгоритм Миллера-Вегмана по существу весьма схож с рассмотренными выше алгоритмами семейства LZ78, и поэтому его использование не мо-

жет привести к существенному повышению эффективности кодирования. Главной причиной недостаточной эффективности по-прежнему является неполнота словаря. Тотальное включение строк в словарь может стать хорошим решением, однако здесь возникает закономерный вопрос: а не будет ли алгоритм, построенный на основе данного решения, фактически эквивалентен некоторому алгоритму семейства LZ77? Оказывается, что нет. Ясно, что в словарь имеет смысл включать только те строки, которые ему не принадлежат. Из-за особенностей способа кодирования данный подход является реализуемым в алгоритмах семейства LZ78 и почти не реализуемым в алгоритмах семейства LZ77. Таким образом, алгоритм семейства LZ78, при работе которого в словарь добавляются только все уникальные строки, имеет определенное преимущество перед любым алгоритмом семейства LZ77 (при работе алгоритмов семейства LZ77 в словарь добавляются вообще все строки). Отметим, правда, что данное преимущество не всегда определяет превосходство в эффективности.

Описанная идея была впервые использована в алгоритме LZJ М. Якобсона [36]. Во время работы алгоритма в словарь добавляются все уникальные строки с длиной, не превышающей некоторое наперед заданное значение. К выбору этого значения надо подходить очень тщательно, так как невыгодным является как хранение только коротких строк, так и хранение всех уникальных строк с длиной, ограниченной очень большим значением. В качестве словаря в алгоритме LZJ выступает дерево, каждый узел которого содержит ровно один символ и соответствует некоторой уникальной строке (строка получается последовательным прочтением символов, принадлежащих узлам дерева, проходимым при движении по дереву от корневого узла к выбранному узлу). Каждому узлу присваивается уникальный индекс, непосредственно используемый при кодировании совпадений. Количество индексов в словаре, как обычно, ограничивается. При добавлении новой строки в случае отсутствия свободных индексов из дерева удаляется строка, ранее не использовавшаяся при кодировании, а ее индекс присваивается узлу, соответствующему новой строке. Проблема отсутствия совпадений решается способом, аналогичным тому, который применяется в алгоритме LZW.

Алгоритм LZJ послужил основой для более сложного алгоритма LZFG Э. Фиала и Д. Грини [29], обладающего большей эффективностью и производительностью. Алгоритм LZFG нельзя однозначно отнести к семейству LZ78, так как он частично использует способ кодирования, применяемый только в алгоритмах семейства LZ77. Автор оставляет данный алгоритм за рамками монографии, отсылая читателя к работе [29].

Завершая рассмотрение семейства LZ78, необходимо обратить внимание на серьезный недостаток, свойственный почти всем его представителям:

в худшем случае кодирование с применением практически любого алгоритма семейства может привести к значительному увеличению объема исходного представления информации. Дело в том, что размер кодового пространства в алгоритмах семейства LZ78 может существенно превысить размер исходного представления символа, что в случае обработки большого числа одиночных символов очень отрицательно скажется на эффективности кодирования. Решением данной проблемы может стать применение хорошо известного механизма служебных битов, позволяющих различать строки, состоящие из одного символа, и строки, состоящие из нескольких символов. Использование такого решения позволит избежать резких «провалов» в эффективности кодирования, хотя и, безусловно, приведет к снижению этой эффективности в среднем.

### 5.2.3. Алгоритмы группы LZRW

Автором алгоритмов группы LZRW (1989–91 гг.) является Р. Уильямс. Для избежания путаницы с алгоритмом Т. Уэлча (LZW) в аббревиатуру алгоритмов были включены полные инициалы автора.

Содержание данного пункта заметно выделяется на фоне предыдущего изложения. Выше рассматривались весьма общие идеи и подходы, здесь же описываются конкретные алгоритмические решения. Причины этому две. Во-первых, алгоритмы группы LZRW являются очень неплохой иллюстрацией того, как следует воплощать на практике методы экономного кодирования. Во-вторых, подробное рассмотрение алгоритмов этой группы позволяет проследить своеобразную эволюцию решений, начиная от простейших, использованных в первых алгоритмах семейства LZ77 (одно из них представляет алгоритм LZRW1, являющийся упрощенной реализацией алгоритма LZSS), вплоть до наиболее современных решений (см. п. 5.2.5).

При разработке алгоритмов группы LZRW одной из первоочередных задач являлось достижение как можно более высокой производительности при условии ограниченности ресурсов вычислительной системы. Алгоритмы предназначены для работы на вычислительных устройствах с 16-битовой длиной машинного слова и 8-битовой длиной минимальной единицы хранения информации[1].

Ниже будут детально рассмотрены алгоритмы LZRW1, LZRW2, LZRW3 и LZRW4 с вариациями. В рассмотрение сознательно не включен алгоритм LZRW5, являющийся не более чем практическим воплощением алгоритма LZW.

[1] При этом подразумевается, что обрабатываемая информация изначально имеет 8-битовое представление.

Использование битовых операций при реализации алгоритма крайне нежелательно, если решающим фактором является производительность. По этой причине составляющие кода в алгоритме LZRW1 комбинируются из расчета на заполнение нескольких идущих подряд байтов. Механизм компоновки кода достаточно прост. Служебные биты, различающие типы кодовых комбинаций (кодовые пары и незакодированные символы; см. алгоритм LZSS из п. 5.2.1), объединяются по 16 в одно машинное слово. Сами кодовые комбинации следуют непосредственно за данным словом и имеют размер, равный либо 8 битам (незакодированный символ), либо 16 битам (кодовая пара <смещение совпадения, длина совпадения>).

Размер буфера поиска в алгоритме LZRW1 равен 4095 символам, а буфера просмотра – 16 символам. Соответственно, из 16 битов кодовой пары 12 битов отводится для хранения смещения строки в буфере поиска, а 4 бита – для хранения длины совпадения. В алгоритме LZRW1 вводится ограничение на минимальную длину совпадения. Минимальная длина совпадения определяется равной трем символам.

Для поиска идентичных строк в алгоритме LZRW1 применяется хэширование (рис. 5.9). Хэш-таблица состоит из 4096 ячеек, каждая из которых имеет уникальный индекс (в дальнейшем – *индекс хэширования* или *хэш-индекс*). Ячейка хэш-таблицы содержит указатель на позицию в обработанной части информационного потока. Эта позиция является начальной позицией строки, которой соответствует индекс данной ячейки. Соответствие между строками и индексами устанавливается с помощью функции $\mathrm{Hash}(s) = ((40543 \cdot ((((s[0] \text{ shl } 4) \text{ xor } s[1]) \text{ shl } 4) \text{ xor } s[2])) \text{ shr } 4) \bmod 4096$, в качестве аргументов которой выступают первые три символа строки ($s[0]$, $s[1]$, $s[2]$).

В процессе кодирования информации алгоритмом LZRW1 каждый раз по первым трем символам обрабатываемой символьной последовательности вычисляется хэш-индекс. Строка, указатель на которую хранится в соответствующей данному индексу ячейке хэш-таблицы, сравнивается с кодируемой последовательностью. Если совпадение имеет длину, большую либо равную трем символам, и смещение идентичной строки находится в пределах буфера поиска, совпадение считается допустимым, и для его описания используется кодовая пара. В противном случае первый символ обрабатываемой последовательности представляется как незакодированный. После получения очередной кодовой комбинации рассматриваемая ячейка хэш-таблицы обновляется ссылкой на начальную позицию только что закодированной строки или символа.

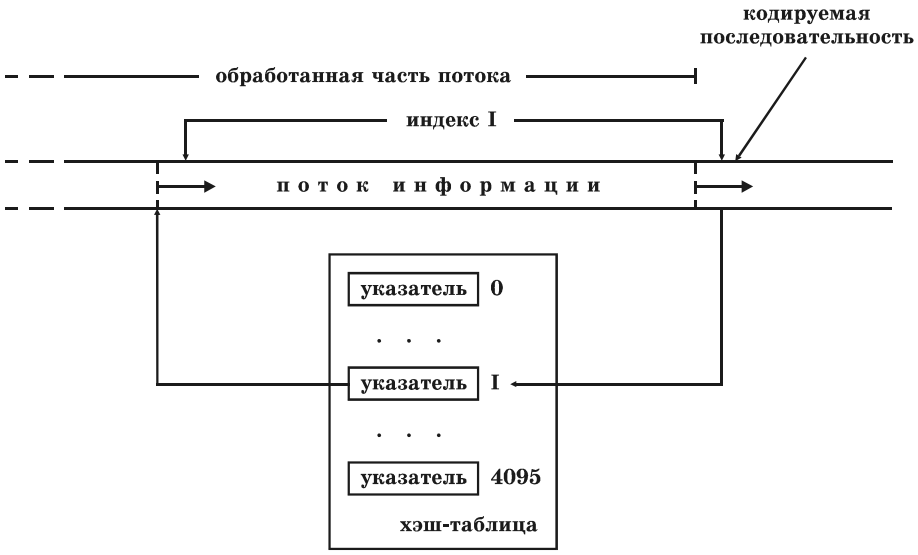



Рис. 5.9. Организация поиска в алгоритме LZRW1

Особенность, которая сразу бросается в глаза при рассмотрении алгоритма LZRW1, – неэффективность представления длины совпадения. Для кодирования длины совпадения в алгоритме резервируется 4 бита, что позволяет хранить одно из 16 возможных значений. Однако, так как совпадения с длиной менее трех символов не допускаются, реально используются только 14 из них. Для устранения этого явного недостатка Уильямсом была предложена модификация алгоритма LZRW1 – алгоритм LZRW1-A. Отличие алгоритма LZRW1-A от алгоритма LZRW1 состоит в расширении интервала допустимых длин совпадений с [3, 16] до [3, 18].

Алгоритм LZRW1 в 3–4 раза опережает по производительности алгоритм LZC (см. п. 5.2.2), отставая по эффективности кодирования в среднем на 10–15%. Высокая скорость кодирования и низкая эффективность обусловлены предельно упрощенной процедурой поиска совпадений, а высокая скорость декодирования – принадлежностью алгоритма к семейству LZ77. Благодаря малой вычислительной сложности алгоритм LZRW1 получил наибольшее распространение в приложениях реального времени. В частности, он послужил основой для нескольких достаточно распространенных утилит сжатия жесткого диска.

Несмотря на общепринятое название описанного алгоритма, Уильямс не является его первооткрывателем. Впервые подобное решение было предложено и запатентовано несколькими годами ранее Д. Уотерворфом.

Малая эффективность алгоритма LZRW1 побудила Уильямса к поиску новых решений. Первым шагом на пути к усовершенствованию алгоритма LZRW1 стал алгоритм LZRW2. Особенностью алгоритма LZRW2 является оригинальная информационная структура, именуемая *таблицей*

*фраз* (*phrase table*). Данная структура представляет собой циклически замкнутый массив из указателей на первые позиции 4096 обработанных строк (фраз). Таблица фраз используется как на этапе кодирования, так и на этапе декодирования. Кроме таблицы фраз, на этапе кодирования также используется и хэш-таблица, только в данном случае хэш-таблица содержит не указатели на позиции в буфере поиска, а номера ячеек таблицы фраз.

Процедура поиска, применяемая в алгоритме LZRW2, проиллюстрирована на рис. 5.10. Кодируемая последовательность сравнивается со строкой, указатель на которую хранится в одной из ячеек таблицы фраз. Номер этой ячейки определяется с помощью хэш-таблицы по индексу, вычисленному по первым трем символам кодируемой последовательности.

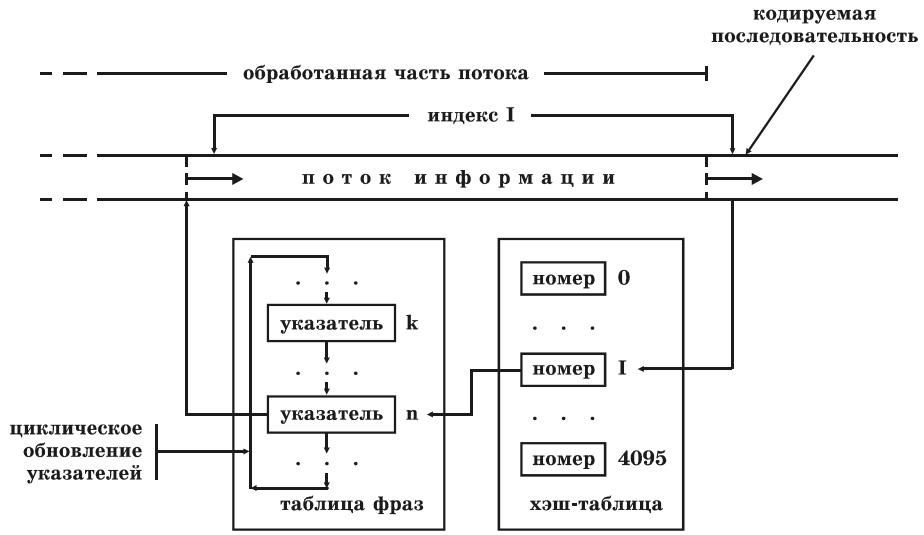



Рис. 5.10. Организация поиска в алгоритме LZRW2

В процессе работы алгоритма LZRW2 требуется осуществлять обновление таблицы фраз. Обновление таблицы фраз производится после обработки очередной порции информации (строки или одиночного символа) и заключается в замене одного из указателей в таблице фраз указателем на начало только что обработанной последовательности символов. Обновляемая позиция меняется циклически. Полный цикл обновления таблицы фраз занимает соответственно 4096 циклов кодирования. Помимо обновления таблицы фраз производится и обновление хэш-таблицы. Для этого в соответствующую ячейку хэш-таблицы записывается номер обновляемой ячейки в таблице фраз.

Главное различие между алгоритмами LZRW1 и LZRW2 состоит в способе кодирования. В алгоритме LZRW2 в каждой кодовой паре вместо смещения совпадения присутствует номер ячейки в таблице фраз, содержащей указатель на это смещение. Потребность в изменении способа кодирования

вызвана нерациональным использованием кодового пространства, отводимого под описание смещений строк в буфере поиска. Алгоритм LZRW1, как мы знаем, допускает включение в кодовую комбинацию произвольных позиций внутри буфера поиска, однако реально могут быть использованы только те позиции, на которые ссылаются указатели в хэш-таблице. В алгоритме LZRW2 строки добавляются одновременно в таблицу фраз, выступающую в данном случае в роли словаря, и в хэш-таблицу. Таким образом, каждая добавляемая строка является потенциальным кандидатом на роль совпадения, а ее смещение потенциально может стать частью кодовой комбинации.

Подход, связанный с использованием таблицы фраз, привлекателен также и по другой причине. В алгоритме LZRW1 буфер поиска ограничен 4095 символами. В алгоритме LZRW2 подобное ограничение отсутствует: при достаточном объеме памяти, отводимом под хранение указателя, возможна ссылка практически на любую позицию в обработанной части информации. Примечательно, что расширение области поиска в данном случае было достигнуто без соответствующего увеличения размера кодовой комбинации.

Применение в алгоритме LZRW2 нового способа кодирования позволило несколько увеличить эффективность. На файлах набора Calgary это увеличение составило в среднем около 4%. При этом алгоритм LZRW2 немного (примерно на 5%) опередил своего предшественника – алгоритм LZRW1-A – в скорости кодирования и на 20–30% уступил ему в скорости декодирования.

Зададимся вопросом: насколько оптимален способ кодирования, применяемый в алгоритме LZRW2? Был ли в нем полностью устранен недостаток алгоритма LZRW1, связанный с наличием неиспользуемых значений кодовых компонентов? Оказывается, что нет – проблема осталась актуальной. Каждый раз, когда производится обновление хэш-таблицы, ссылка на некоторую ячейку таблицы фраз заменяется ссылкой на другую ячейку. Ячейка, номер которой удаляется из хэш-таблицы, до момента ее очередного обновления полностью выпадает из рассмотрения. Таким образом, словарь в алгоритме LZRW2 всегда содержит некоторое количество неиспользуемых элементов, которые формально продолжают учитываться при формировании кода.

Для полного устранения указанного недостатка кодирования необходимо каким-то образом увязать распределение кодового пространства с процедурой поиска совпадений. Наиболее изящным решением является использование в качестве словаря хэш-таблицы. Данное решение легло в основу алгоритма LZRW3.

В алгоритме LZRW3 используются те же структуры данных, что и в алгоритме LZRW1, однако хэш-таблица поддерживается не только на этапе кодирования, но и на этапе декодирования. Это связано с изменением способа кодирования совпадений – в алгоритме LZRW3 первый компонент кодовой пары представляет собой номер ячейки хэш-таблицы, содержащей указатель на совпадение.

Необходимость в обработке хэш-таблицы на этапе декодирования несколько затруднила ее обновление. Проблема состоит в том, что обновление хэш-таблицы на этапах кодирования и декодирования не всегда может быть осуществлено синхронно, так как на этапе декодирования не всегда можно своевременно определить индекс обновляемой ячейки. Если декодируемая кодовая комбинация представляет собой пару <индекс совпадения в хэш-таблице, длина совпадения>, затруднений с определением ячейки, очевидно, не возникает – индекс обновляемой ячейки передается как часть кода. Неприятности связаны с наличием в кодовой последовательности незакодированных символов. Обновление хэш-таблицы не может быть произведено сразу после интерпретации незакодированного символа, так как для определения обновляемой ячейки знания одного символа недостаточно – надо знать следующие два символа. С целью избежания подобной ситуации обновление хэш-таблицы в алгоритме LZRW3 откладывается до того момента, когда будут обработаны все три символа (задержка, естественно, производится как на этапе кодирования, так и на этапе декодирования).

К достоинствам алгоритма LZRW3 относится ограничение дальности поиска, опять же зависящее только от размера памяти, отводимой под указатель, а также потенциальная возможность использования при кодировании всего кодового пространства. Недостатком алгоритма является повышенная сложность процесса декодирования. Алгоритм LZRW3 опережает по эффективности кодирования алгоритм LZRW2 в среднем на 1%. Он превосходит алгоритм LZRW2 в скорости кодирования на 5–10%, но уступает примерно в 2 раза в скорости декодирования.

Идея, впервые воплощенная в алгоритме LZRW3, легла в основу еще одного алгоритма группы LZRW – алгоритма LZRW3-A. Данный алгоритм представляет собой модификацию алгоритма LZRW3, созданную с целью повышения эффективности кодирования.

В алгоритме LZRW3-A хэш-таблица условно разбивается на 512 блоков. Каждый блок содержит 8 указателей на начальные позиции 8 обработанных строк, которым соответствует один и тот же индекс хэширования (рис. 5.11). В отличие от предыдущих алгоритмов группы, где поиск совпадений сводился к сравнению кодируемой строки ровно с одной строкой словаря, в алгоритме LZRW3-A в поиске участвуют сразу 8 строк, указатели на которые принадлежат одному из блоков хэш-таблицы (ищется наиболее

длинное совпадение). Номер блока соответствует хэш-индексу кодируемой последовательности, вычисляемому с использованием функции $\mathrm{Hash}(s) = ((40543 \cdot ((((s[0]\ \mathrm{shl}\ 4)\ \mathrm{xor}\ s[1])\ \mathrm{shl}\ 4)\ \mathrm{xor}\ s[2]))\ \mathrm{shr}\ 4) \bmod 512$.

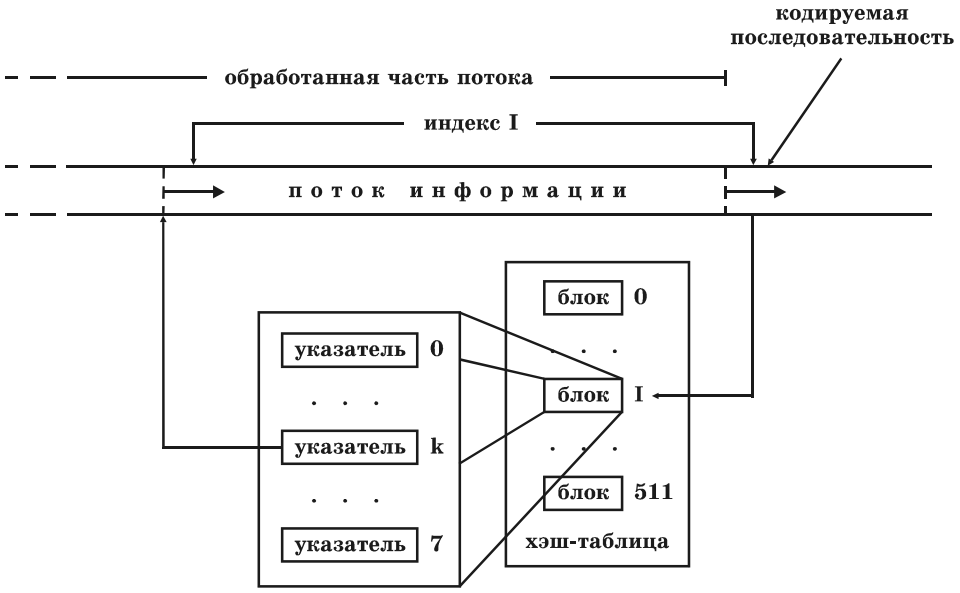



Рис. 5.11. Организация поиска в алгоритме LZRW3-A

Изменения в алгоритме поиска не повлекли за собой изменения в способе кодирования – алгоритм LZRW3-A использует тот же способ кодирования, что и алгоритм LZRW3. Отсутствие необходимости в изменении способа кодирования объясняется сохранением формата представления информации – для представления хэш-индекса по-прежнему требуется 12 бит кодового пространства.

Новая структура хэш-таблицы потребовала использования особого способа обновления указателей. Наиболее разумным решением в данной ситуации было бы введение 512 маркеров, указывающих на обновляемые позиции внутри блоков таблицы хэширования. Циклически перемещая маркеры в пределах каждого из блоков, мы тем самым обеспечили бы последовательность в обновлении указателей. В алгоритме LZRW3-A применяется несколько иной, более упрощенный подход: для всех элементов хэш-таблицы заводится единый маркер, который циклически перемещается в пределах некого виртуального блока, в качестве которого в разные моменты времени выступают различные реальные блоки хэш-таблицы. Такой способ обновления, конечно, не оптимален, однако он достаточно прост в реализации, что в данном случае имеет большое значение.

Применение поиска совпадений с альтернативами в алгоритме LZRW3-A позволило на несколько процентов повысить эффективность кодирования.

При этом заметным недостатком стало трехкратное снижение скорости кодирования, обусловленное увеличением числа операций, требуемых для осуществления поиска.

Одной из задач, ставившихся при разработке алгоритма LZRW3-A, было создание алгоритма, способного конкурировать с часто используемым в то время на практике алгоритмом LZC (см. п. 5.2.2). В связи с этим небезынтересно будет сравнить данные алгоритмы. Оказывается, что алгоритм LZRW3-A опережает алгоритм LZC примерно в два раза по скорости кодирования и примерно в три раза по скорости декодирования. Алгоритм LZRW3-A обладает большей эффективностью применительно к файлам небольшого размера (несколько десятков килобайт) и к файлам с малой избыточностью. Во всех остальных случаях лучшим с точки зрения эффективности оказывается алгоритм LZC.

Алгоритм LZRW3-A стал последним алгоритмом группы LZRW, относящимся к семейству LZ77. Выбранный подход более не позволял добиться существенного прироста в эффективности. Главная причина – необоснованно большой объем кодового пространства, отводимый под описание совпадений. Заметное повышение эффективности могло стать возможным только за счет уменьшения длины кода, однако очевидное решение использовать коды переменной длины было абсолютно неприемлемо из-за чрезмерной вычислительной сложности. Требовалось найти более простой подход. Такой подход в конечном итоге был найден. Он послужил основой алгоритма LZRW4.

Для того, чтобы лучше понять сущность нового подхода, потребуется вспомнить принцип работы статистических методов. Кодирование в статистическом методе основано на предсказании вероятности появления символа в текущем контексте. Для осуществления такого предсказания во время работы поддерживается некоторая информационная структура, содержащая сведения статистического порядка об обработанной части информации. Данная информационная структура фактически эквивалентна словарю, за тем исключением, что на ее содержимое при кодировании не допускаются прямые ссылки. «Бессылочное» кодирование становится возможным благодаря тому, что кодер и декодер на любом этапе своей работы имеют как бы общую точку отсчета, в роли которой выступает текущий контекст. Наличие этой точки отсчета позволяет заменить абсолютный способ кодирования (*ссылочный*) на относительный (*контекстно-зависимый*)[1]. Нетрудно понять, что подобная замена возможна не только в статистических методах. Распространение идеи контекстно-зависимого кодирования на словарные

[1] Деление способов кодирования на ссылочные и контекстно-зависимые весьма условно. В частности, не лишена основания точка зрения, согласно которой все способы кодирования в той или иной мере являются контекстно-зависимыми.

методы как раз и составляет сущность нового подхода, использованного в алгоритме LZRW4.

На этапах кодирования и декодирования в алгоритме LZRW4 поддерживается хэш-таблица, традиционно состоящая из 4096 ячеек. Каждая ячейка содержит 32 указателя на начальные позиции обработанных строк (рис. 5.12). При кодировании очередной последовательности символов по двум предыдущим только что закодированным символам вычисляется хэш-индекс, определяющий одну из ячеек таблицы хэширования (в данном случае функция, в соответствии с которой вычисляется хэш-индекс, приобретает вид $\mathrm{Hash}(s) = ((40543 \cdot ((s[-2] \,\mathrm{shl}\, 8) \,\mathrm{xor}\, s[-1])) \,\mathrm{shr}\, 4) \bmod 4096$). Ячейка содержит указатели на строки, встречавшиеся ранее в двухсимвольных контекстах, которым соответствует вычисленный хэш-индекс. Существует большая вероятность того, что две пары символов, которым соответствует один и тот же индекс хэширования, совпадают. Учитывая, что одинаковые пары символов нередко являются составными частями более длинных идентичных последовательностей, не исключено, что один из 32 указателей в выбранной ячейке указывает на начало обработанной строки, частично совпадающей с кодируемой символьной последовательностью. Здесь мы переходим к самому интересному этапу – этапу кодирования. Если какой-то из 32 указателей указывает на совпадение, то для его описания в кодовую комбинацию не надо включать индекс ячейки хэш-таблицы – достаточно только номера указателя в ячейке! В этом заключается бесспорное преимущество контекстно-зависимого подхода.

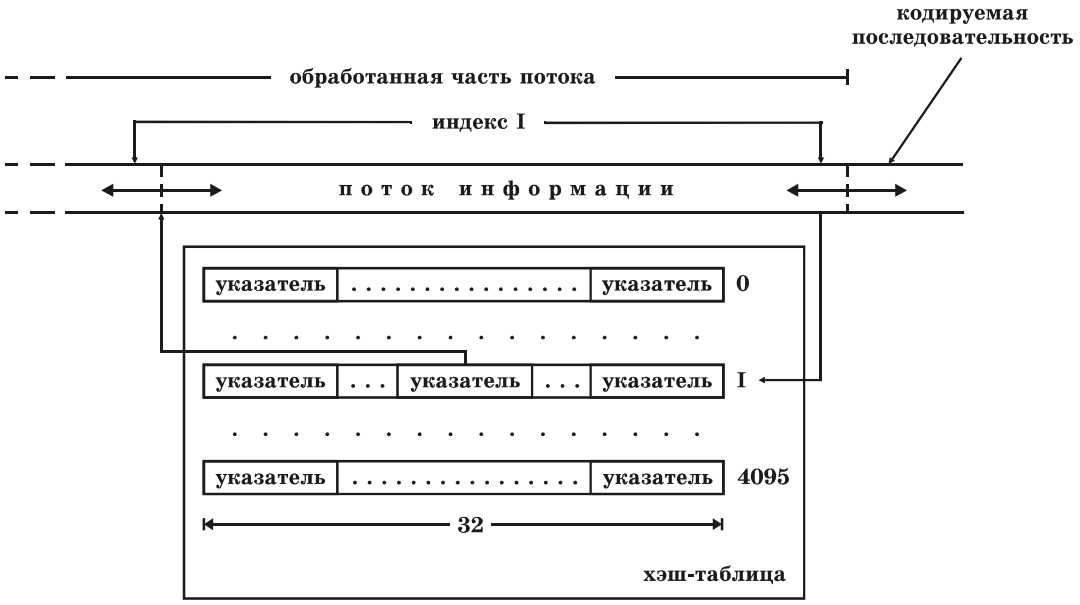



Рис. 5.12. Организация поиска в алгоритме LZRW4

Поиск совпадений в алгоритме LZRW4 считается успешным в случае, когда длина найденного совпадения превышает один символ. Каждое совпадение кодируется парой <номер указателя, длина совпадения>. Под номер указателя отводится 5 битов ($2^5 = 32$ указателя), а под длину указателя – 3 бита ($2^3 = 8$ различных длин). При этом допускается кодирование следующих длин совпадений: 2, 3, 4, 5, 6, 7, 8, 16 (если длина совпадения не совпала ни с одним из этих значений, то выбирается ближайшее значение, не превышающее эту длину). Как видно, длина кодовой пары в алгоритме LZRW4 равна всего 8 битам, тогда как в других алгоритмах группы LZRW она составляет 16 битов.

Немалый интерес в алгоритме LZRW4 вызывает способ обновления хэш-таблицы. Добавление нового указателя на начало только что закодированной последовательности символов производится в случае, когда длина совпадения составляет не более 3-х символов. Новый указатель добавляется в ячейку таблицы на первую позицию (указатели в ячейках таблицы хэширования находятся на пронумерованных позициях; эти номера используются при кодировании совпадений). При этом из таблицы удаляется указатель, находившийся на последней позиции данной ячейки, а все остальные указатели сдвигаются на одну позицию в соответствии с возрастанием номера. Однако существует одна особенность. Если при кодировании только что обработанной последовательности использовалась кодовая пара (то есть длина совпадения превысила один символ), то перед добавлением нового указателя (если, конечно, это требуется) меняются местами указатель на найденное совпадение и указатель, находящийся на позиции, номер которой получается делением пополам номера позиции указателя совпадения. Таким образом, из хэш-таблицы каждый раз удаляется наиболее старый указатель из тех, которые за несколько последних шагов кодирования ни разу не указывали на наибольшее совпадение.

Поиск с обработкой большого числа указателей требует большого количества операций сравнения. Поэтому из всех алгоритмов группы LZRW алгоритм LZRW4 обладает самой низкой скоростью кодирования. По скорости декодирования алгоритм LZRW4 вполне сравним с другими алгоритмами рассматриваемой группы.

Несмотря на значительное сокращение размера кодового пространства, алгоритм LZRW4 не дал ощутимого прироста в эффективности кодирования: в среднем он опережает по эффективности алгоритм LZRW3-A всего на несколько процентов. Отчасти это можно объяснить снижением эффективности поиска совпадений (в алгоритме LZRW4, помимо оригинального способа кодирования, используется и совершенно иной способ поиска совпадений). Впрочем, эффективность алгоритма LZRW4 в большой степени зависит от особенностей обрабатываемой информации: на информации,

обладающей большой избыточностью, он ведет себя значительно лучше, чем на малоизбыточной информации, применительно к которой более эффективным часто оказывается алгоритм LZRW3-A.

Мы рассмотрели все обещанные алгоритмы группы LZRW4. Подведем итог. Алгоритмы группы LZRW занимают отдельную нишу в ряду алгоритмов экономного кодирования, являясь простейшими представителями словарного подхода. Будучи нетребовательными к ресурсам, они оказываются применимыми практически в любых ситуациях. Среди алгоритмов группы LZRW нельзя однозначно выделить какой-то один алгоритм. В этом отношении мы обладаем достаточно широкими возможностями для выбора. С точки зрения эффективности наиболее предпочтительными являются алгоритмы LZRW3-A и LZRW4, а с точки зрения производительности – алгоритмы LZRW1-A, LZRW2 и LZRW3. Сегодня алгоритмы группы LZRW все реже и реже применяются на практике, уступая свое место более современным алгоритмам. Во многом это объясняется всевозрастающими требованиями, предъявляемыми к эффективности кодирования. Так сложилось, что алгоритмы группы LZRW больше не удовлетворяют потребностям реальных приложений, так что в ближайшем будущем следует ожидать полного отказа от их практического использования.

### 5.2.4. Комбинированные алгоритмы семейства LZ77

На примере рассмотренных методик можно было убедиться в том, что в большинстве своем они рассчитаны на использование в некоторых вполне конкретных условиях, невыполнение которых способно привести к совершенно непредсказуемому поведению того или иного алгоритма. В принципе, проблема ограниченной применимости вполне разрешима путем объединения различных готовых схем экономного кодирования (см., например, [66, 67]), однако, во-первых, такой подход связан с необходимостью реализации сразу нескольких алгоритмов, а во-вторых, здесь неизбежно возникает проблема выбора оптимального решения для каждой конкретной ситуации. Предпочтительнее попытаться создать алгоритм, обладающий достоинствами сразу всех существующих методик экономного кодирования, одинаково пригодный для обработки информации произвольного существующего типа. На сегодня создать такой универсальный алгоритм никому не удалось (да и вряд ли удастся в будущем), но все же был предложен ряд частично универсальных схем, позволяющих во многих случаях получать вполне приемлемые результаты и, чаще всего, представляющих собой удачное сочетание (комбинацию) нескольких базовых алгоритмов экономного кодирования. Указанная приемлемость в различных ситуациях трактуется по-разному: в одних случаях это высокая эффективность

кодирования, близкая к максимально достижимой на данном типе информации при полном отсутствии ограничений, в других – высокая эффективность кодирования, близкая к максимально достижимой на данном типе информации за фиксированное время при ограниченных ресурсах. Комбинированные словарные схемы, основу которых составляют словарные алгоритмы, оптимальны как раз в случае наличия определенных ограничений, накладываемых на ресурсы вычислительной системы. До последнего времени такие ограничения были свойственны большинству практических приложений, использующих экономное кодирование, что явилось причиной высокой популярности этих схем.

Алгоритм LZSS оказывается наиболее подходящим на роль базового алгоритма в комбинированных словарных схемах. Сам по себе данный алгоритм малоэффективен, что во многом связано с неоптимальным использованием кодового пространства при кодировании. Умелое устранение этого недостатка позволяет заметно повысить эффективность алгоритма при сохранении приемлемой скорости обработки информации. Для того, чтобы понять, как этого добиться, необходимо глубже разобраться в причинах недостаточной эффективности данного алгоритма.

Как мы знаем, во время работы алгоритма LZSS в задней части скользящего окна производится поиск совпадений, в результате которого на каждой итерации констатируется либо полное отсутствие совпадений, либо их наличие. В случае, когда найдено несколько совпадений, необходимо выбрать одно из них – то, на основе которого будет осуществляться кодирование. Для простоты положим, что каждый раз кодирование осуществляется на основе наиболее длинного найденного совпадения, причем, если имеется несколько совпадений наибольшей длины, выбирается то из них, позиция которого находится ближе всего к текущей обрабатываемой позиции[1]. На рис. 5.13 изображено распределение частот появления наибольшего совпадения по всевозможным позициям в буфере поиска, полученное экспериментально при обработке текстового файла (по горизонтали откладывается смещение позиции в буфере поиска относительно текущей обрабатываемой позиции). Как видно из графика распределения, с увеличением смещения вероятность появления наибольшего совпадения на соответствующей позиции уменьшается. Несмотря на кажущуюся очевидность такой формы распределения частот (она объясняется наличием локальных закономерностей в информации и способом выбора совпадений), она характерна не для всех источников информации. Например, распределение, экспериментально полученное при обработке графической информации, представленной в фор-

[1] В практических реализациях словарных методов не всегда следуют данному правилу выбора.

мате True Color BMP, имеет несколько иной вид (рис. 5.14). Периодичность в данном случае обусловлена построчным представлением изображения в файле (при построчном обходе изображения происходит периодическое возвращение к одним и тем же его фрагментам; наибольшие совпадения чаще всего находятся на расстоянии, кратном размеру строки). При практических реализациях словарных алгоритмов подобные исключительные ситуации можно, конечно, не принимать во внимание, однако нельзя не учитывать общую тенденцию к затуханию в распределении частот, свойственную подавляющему большинству реальных информационных источников.

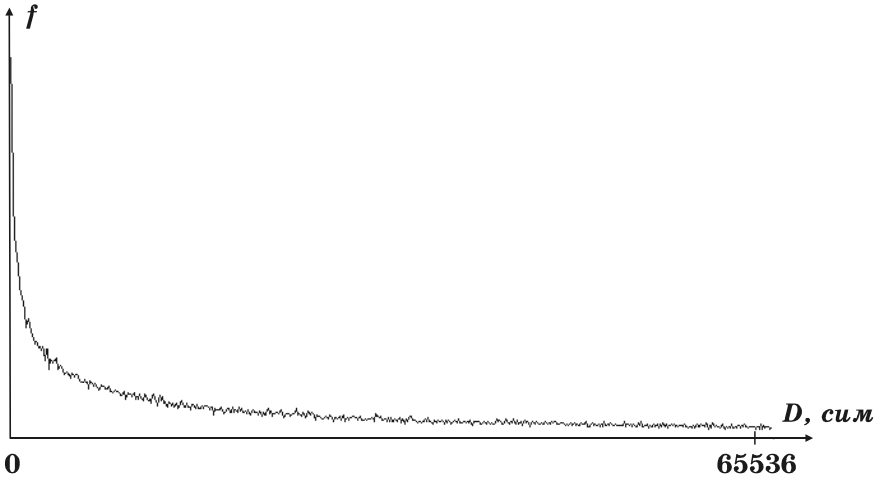



Рис. 5.13. Типичное распределение частот появления наибольшего совпадения по различным позициям в буфере поиска, получаемое при обработке текстовой информации

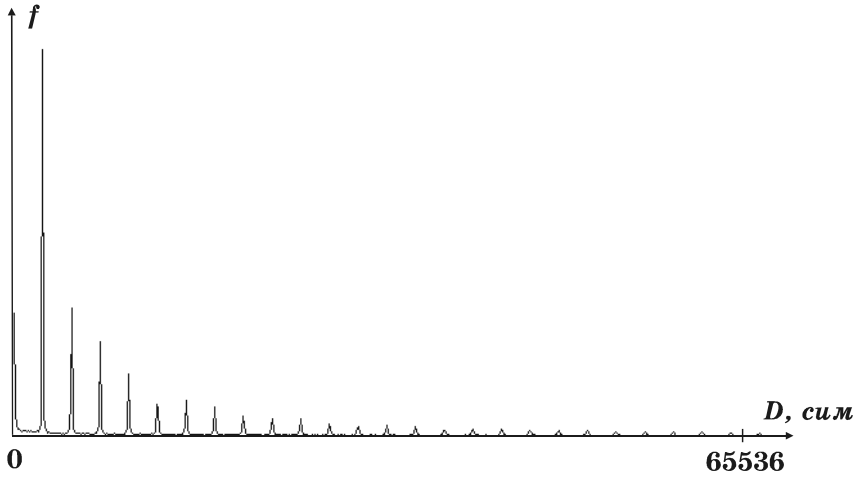



Рис. 5.14. Типичное распределение частот появления наибольшего совпадения по различным позициям в буфере поиска, получаемое при обработке графической информации, представленной в формате True Color BMP

Распределение частот появления различных длин наибольших совпадений также имеет свои особенности (рис. 5.15). При движении в сторону воз-

растания длины совпадения график распределения быстро затухает. Такой характер объясняется более частым появлением коротких совпадений по сравнению с длинными совпадениями и отсутствием очень длинных совпадений.

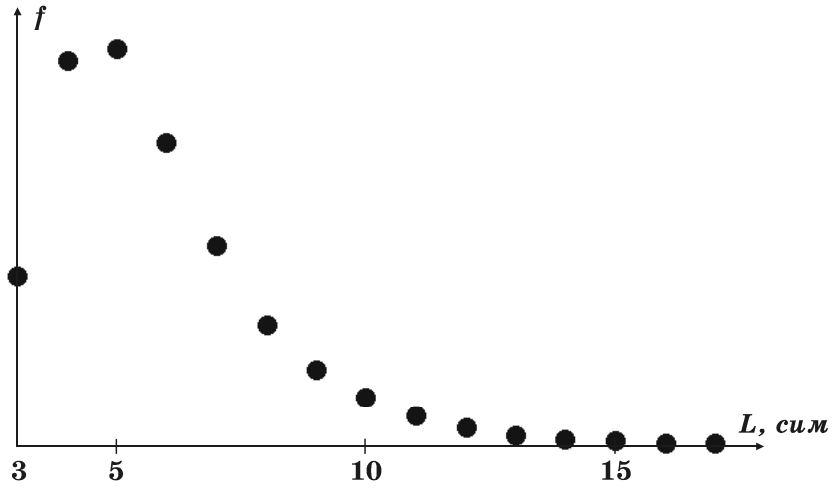



Рис. 5.15. Типичное распределение частот появления различных длин наибольших совпадений, получаемое при обработке текстовой информации

Указанные выше особенности никак не учитываются в алгоритме LZSS. При кодировании в алгоритме используется равномерный код, тогда как в данном случае следовало бы использовать коды переменной длины. Совершенно очевидно, что длина кодовой комбинации должна меняться в зависимости от частоты появления кодируемой позиции и длины совпадения. К сожалению, реализация данного подхода сопряжена с большими трудностями, вызванными необходимостью хранения и обработки колоссального объема статистической информации, содержащей частоты появления различных позиций и длин совпадений.

Для решения проблем, возникающих при реализации метода, чаще всего прибегают к следующему простому приему. Пространство позиций и пространство длин разбивают на небольшие блоки, распределение значений внутри которых условно считается равномерным. Это позволяет сократить объем обрабатываемой статистической информации, так как при получении кода переменной длины учитывается статистика появлений уже не отдельных позиций или длин, а целых блоков соответствующих значений. Безусловно, эффективность данного подхода во многом зависит от оптимальности выбора размеров блоков. При использовании слишком больших блоков статистика оказывается недостаточно точной, что, конечно, отрицательно сказывается на эффективности, а при использовании слишком маленьких блоков существенно возрастает объем статистической информации.

Длины блоков могут различаться. Интуитивно понятно, что чем сильнее меняется статистическое распределение в некотором интервале значений, тем меньше должен быть шаг разбиения в пределах этого интервала. Основываясь на графиках, приведенных выше (см. рис. 5.13, 5.15), можно сделать предположение о том, что оптимальным будет разбиение с экспоненциально меняющейся длиной блока.

Объем статистической информации можно существенно сократить и за счет ограничения общего числа возможных позиций и длин совпадений. Приведенные распределения частот свидетельствуют о том, что подавляющее большинство позиций и длин наибольших совпадений находится вблизи нуля. Поэтому наложение разумных ограничений на дальность (глубину) поиска совпадений и на максимальную длину совпадения, позволяющее уменьшить время поиска и, как следствие, увеличить скорость кодирования, не должно привести к заметному снижению эффективности этого кодирования. Практика показывает, что размер буфера поиска можно ограничить значением, лежащим в пределах от $2^{15}$ до $2^{16}$, а размер буфера просмотра достаточно брать равным нескольким десяткам символов. Возможные потери от таких ограничений обычно не превышают нескольких процентов, хотя во многом все зависит от типа и объема обрабатываемой информации.

Итак, благодаря применению кодов переменной длины мы можем в определенной степени повысить эффективность представления совпадений. Однако помимо кодовых комбинаций, описывающих совпадения, существует также и другой тип кодовых комбинаций – кодовые комбинации, предназначенные для представления незакодированных символов. Насколько оптимально кодирование в данном случае? Попробуем подробнее разобраться в этом вопросе.

Число кодовых комбинаций, представляющих собой незакодированные символы, можно считать хорошим показателем эффективности кодирования: как правило, чем меньше их доля в общем количестве кодовых комбинаций, тем выше эффективность кодирования. Кодовая последовательность, полученная при обработке информации с очень низкой избыточностью, практически целиком состоит из кодов незакодированных символов, что говорит о малой эффективности алгоритма применительно к такой информации. Более избыточная информация может быть обработана с большей эффективностью, однако определенная ее часть все равно остается в незакодированном виде. Например, при обработке текстовой информации доля незакодированных символов, как правило, составляет 20–30% от общего объема информации. Несложная оценка показывает, что уменьшение объема «незакодированной» части текстовой информации хотя бы на треть приводит к повышению эффективности алгоритма в целом, как минимум, на 5%.

Проблема эффективного представления «незакодированной» информации требует особого подхода, так как не все рассмотренные методы одинаково пригодны для обработки такой информации. Причина кроется в наличии у данной информации особых статистических свойств, отличных от статистических свойств исходной информации. Нетрудно заметить, что словарные алгоритмы при работе косвенно используют межсимвольные корреляционные связи, которыми обладает обрабатываемая информация (здесь мы умышленно прибегаем к вероятностной терминологии). Совпадения являются следствием существования таких связей, тогда как их отсутствие означает либо полное отсутствие этих связей, либо их слабость. На основании этого можно сделать заключение о том, что «незакодированная» информация в большинстве своем обладает очень незначительными корреляционными связями, порядок которых определяется главным образом выбранной минимальной длиной совпадения. Если минимальная длина равна трем символам, что характерно для подавляющего числа используемых на практике модификаций алгоритма LZSS, то средний порядок корреляционных связей между символами в «незакодированной» информации не превысит единицы. Если же выбрать минимальную длину совпадения равной двум символам, то корреляционные связи между различными незакодированными символами будут практически отсутствовать.

Для эффективной обработки «незакодированной» части информации необходим метод кодирования, оптимально учитывающий статистические особенности соответствующих малых порядков. В качестве такого метода целесообразно использовать статистическую схему контекстного спуска с фиксированным порядком первой оценивающей модели (см. метод PPM, п. 5.1.1). В свете вышесказанного порядок первой оценивающей модели должен быть равен единице, однако на практике предпочтение чаще отдается нулевой модели (напомним, что в нулевой модели вероятности появлений символов вычисляются без учета контекстов, в которых эти символы появляются; см. п. 3.1.1). При этом для генерации кода обычно используется кодирование Хаффмана, замена которого на арифметическое кодирование в рассматриваемой ситуации дает очень незначительный выигрыш в эффективности.

Мы рассмотрели способы повышения эффективности представления кодовых компонентов в алгоритме LZSS. Осталось устранить недостаток алгоритма, связанный с несовершенством механизма служебных битов (наличие этого недостатка отмечалось в п. 5.2.1). Распределение вероятностей появления значений служебных битов нередко имеет ярко выраженные статистические закономерности. (Данные закономерности опять же напрямую связаны с особенностями обрабатываемой информации: чем сильнее корреляционные связи между символами в информации, тем больше вероятность

нахождения совпадений в процессе поиска, и, следовательно, тем чаще при кодировании генерируются служебные биты, указывающие на совпадения.) Эффективным способом уменьшения объема последовательности служебных битов могло бы стать применение к данной последовательности арифметического кодирования, построенного на основе некоторой статистической модели[1]. Однако такая побитовая обработка вряд ли приемлема с учетом низкой скорости работы арифметического кодирования. Кроме того, не стоит забывать о наличии уже рассмотренных механизмов повышения эффективности алгоритма LZSS. В совокупности с данными механизмами комбинированная схема экономного кодирования, которую мы хотим построить, становится чересчур громоздкой. К счастью, из сложившегося затруднительного положения существует достаточно простой выход. Соломоновым решением является организация экономного кодирования на основе объединенной статистики появления значений служебных битов, позиций и длин совпадений, а также незакодированных символов.

Вероятно, впервые объединение статистической информации в комбинированной словарной схеме было предложено Х. Окумурой в 1988 году. Возможность такого объединения обусловлена тем фактом, что кодовые комбинации, генерируемые в процессе кодирования с применением алгоритма LZSS, представляют собой своеобразный алфавит информационных сообщений, в качестве источника которых выступает кодер алгоритма. Иначе говоря, кодовая последовательность, поступающая с выхода кодера, может рассматриваться как обычная информационная последовательность. Хотя данная последовательность в принципе может быть обработана произвольным алгоритмом экономного кодирования, наибольший эффект достигается при использовании статистических алгоритмов. При реализации таких алгоритмов нельзя не учитывать определенную специфику информационного алфавита, заключающуюся в наличии особого типа символов – кодовых пар. Основные трудности вызывает их составная структура. Если рассматривать каждую кодовую пару как отдельный символ информационного алфавита, то количество таких символов, равное числу всевозможных сочетаний <позиция, длина>, даже при объединении смежных позиций и близких значений длин совпадений в блоки, оказывается чрезвычайно большим. Поэтому в качестве символа имеет смысл брать только одну из двух составных частей кодовой пары. Оставшуюся часть в этом случае необходимо обрабатывать с использованием отдельной статистической модели[2]. Окумура

[1] Префиксное кодирование в данном случае не может быть использовано из-за ограничения, касающегося минимальной длины кода.

[2] Разделение статистических моделей может привести к снижению эффективности алгоритма, так как данное решение связано с отказом от учета возможной корреляции между компонентами кодовой пары.

предложил объединять незакодированные символы исходного информационного алфавита и допустимые длины совпадения в одну статистическую модель, а возможные позиции совпадений – в другую. На самом деле выбор по большей части произволен, так как он практически не влияет на эффективность кодирования.

Итак, в чем же состоит выигрыш от объединения статистической информации? Оказывается, что данное решение позволяет полностью отказаться от механизма служебных битов. Служебные биты были необходимы в алгоритме LZSS для различения двух типов кодовых комбинаций. При использовании кодов переменной длины, полученных с учетом объединенной статистики, каждый код однозначно интерпретируется либо как отдельный символ, либо как кодовая пара. Таким образом, во время работы алгоритма необходимо поддерживать не три различные статистические модели, а две или, даже, одну модель! Кроме того, объединение статистической информации позволяет использовать более быстрое кодирование Хаффмана взамен теоретически более выгодного арифметического кодирования без заметного снижения эффективности, что было невозможно при обособленном кодировании последовательности служебных битов.

Реализации рассмотренной выше комбинированной словарной схемы, строящейся на базе алгоритма, производного от алгоритма LZ77, и включающей в себя некоторый статистический алгоритм, устраняющий недостатки способа кодирования базового словарного алгоритма, принято называть *комбинированными словарными алгоритмами семейства LZ77*. Комбинированные словарные алгоритмы получили наиболее широкое распространение и являются основой таких известных утилит сжатия и архивации данных, как ACE, ARJ, CABARC, GZIP, HA[1], IMP[2], JAR, LHA, PKZIP и RAR.

В процессе разработки комбинированных словарных алгоритмов особое внимание следует уделять организации поиска совпадений, поскольку от того, насколько грамотно он реализован, в высшей степени зависит производительность алгоритма. Вполне приемлемым решением является использование хэширования с разрешением коллизий методом цепочек (см., например, [2]). Различным индексам в хэш-таблице ставятся в соответствие связанные списки указателей на строки словаря. Индекс хэширования, как обычно, вычисляется по первым нескольким символам кодируемой последовательности. Во время поиска совпадений элементы списка, соответствующего вычисленному индексу, просматриваются с целью нахождения наиболее длинного совпадения (рис. 5.16).

[1] Утилита HA также включает в себя реализацию метода PPMC (см. п. 5.1.1).

[2] Утилита IMP также включает в себя реализацию метода Барроуза-Уилера (см. п. 5.3.2).

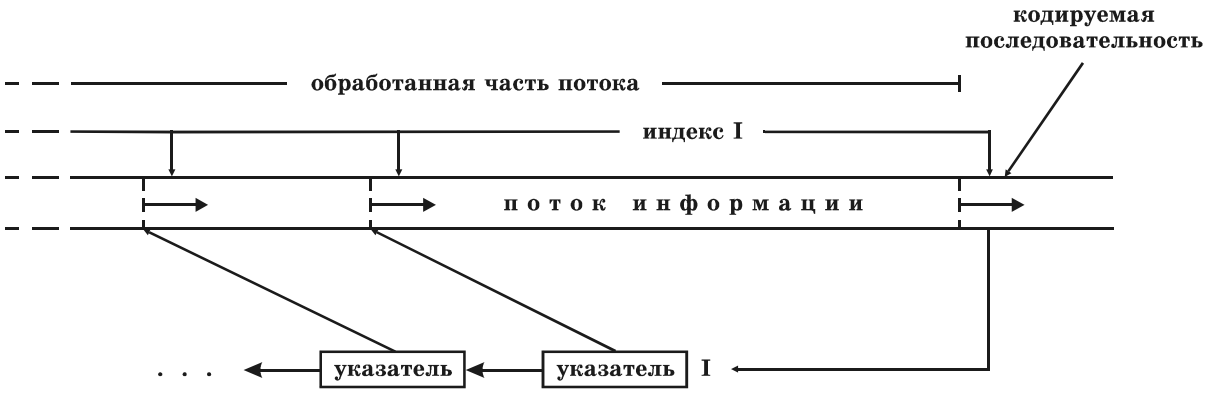



Рис. 5.16. Применение хэширования для поиска совпадений

Указанный алгоритм поиска обладает тремя явными достоинствами: он прост в реализации, нетребователен к ресурсам памяти и удобен с точки зрения организации ограничения глубины поиска. Тем не менее, если требуется добиться максимально возможной скорости поиска, все же целесообразнее прибегнуть к иным решениям. Наибольшая производительность поиска, как правило, достигается за счет использования сложных многоуровневых хэш-структур, а также деревьев цифрового поиска.

В завершение данного пункта приведем некоторые результаты, получаемые с применением комбинированных словарных алгоритмов. Автором были протестированы утилиты GZIP версии 1.2.4, RAR версии 2.00 и IMP версии 1.12. Данные программные реализации комбинированной словарной схемы отличаются в основном размером буфера поиска. В протестированных версиях утилит GZIP и RAR размер буфера поиска составляет $2^{15}$ и $2^{16}$ символов, а в утилите IMP он равен $2^{20}$ символов. При тестировании в программах были задействованы режимы работы, оптимальные с точки зрения эффективности кодирования. Усредненные результаты, полученные при обработке текстовых файлов book1 и book2 набора Calgary указанными утилитами, равны 2.98, 2.86 и 2.62 бит/сим. Последний результат близок к результатам, получаемым с применением некоторых статистических алгоритмов (см. п. 5.1), однако максимальный проигрыш все же составляет около 0.6 бит/сим. Ситуация несколько меняется в случае, когда обрабатываемая информация менее избыточна. Так, средняя эффективность утилит GZIP, RAR и IMP на объектных файлах obj1 и obj2 набора Calgary равна 3.23, 3.09 и 3.08 бит/сим, тогда как средняя эффективность реализаций статистических методов DMC [26], PPMC [22], PPMD [22], CTW [65] и алгоритма PPMZ2 [17] соответственно составляет 3.86, 3.09, 3.08, 3.14 и 2.95 бит/сим. Учитывая высокую скорость работы практических реализаций

комбинированной словарной схемы, о их применении в данном случае можно говорить как о наиболее обоснованном решении.

### 5.2.5. Алгоритм LZP

Рассматриваемый алгоритм разработан Ч. Блюмом в результате развития идей Р. Уильямса, впервые нашедших свое воплощение в алгоритме LZRW4 (см. п. 5.2.3). В работе [18] Блюм предложил сразу несколько словарных схем, использующих способ кодирования, основанный на знании текущего контекста. Все они были объединены под общим названием «алгоритм LZP» (LZ + Prediction – «словарная схема с предсказаниями»). В данном изложении мы не будем очень подробно останавливаться на каждой из этих схем, а рассмотрим только наиболее значимые приемы, предопределяющие их эффективность.

Как показывает практика, расширение словаря в словарных методах не всегда может компенсировать увеличение размера кодовой комбинации и, как следствие, ведет к снижению эффективности кодирования. Исходя из этого, Блюм приходит к достаточно спорному заключению о том, что при осуществлении кодирования на основе текущего контекста каждый раз достаточно искать совпадение обрабатываемой строки только с последней строкой, встреченной ранее в данном контексте (в алгоритме LZRW4 при поиске анализировались 32 ранее встреченные строки). При этом подразумевается, что тщательным образом будут учитываться статистические особенности появления значений служебных битов (при выбранном подходе для описания совпадений не требуется никакой дополнительной информации), длин совпадений и незакодированных символов. Служебные биты и длины совпадений следует кодировать одновременно с применением арифметического кодирования (данный механизм был подробно описан в предыдущем пункте), а незакодированные символы предлагается обрабатывать методом PPMC. Понятно, что описанную в общих чертах схему (в [18] она представлена как алгоритм LZP4) скорее можно считать статистическим алгоритмом, нежели словарным алгоритмом, так как роль словаря здесь фактически сведена к минимуму.

Отказ от использования сложных статистических подходов ведет, с одной стороны, к снижению эффективности кодирования, а с другой стороны – к увеличению скорости этого кодирования. Блюм предлагает варьировать степень использования таких подходов в алгоритме LZP для получения широкого спектра его вариаций, каждая из которых может стать оптимальным решением для той или иной задачи (в дальнейшем изложении эти вариации будут именоваться *алгоритмами группы LZP*). Блюмом были созданы четыре алгоритма, первый из которых (алгоритм LZP1) является неплохой заменой для одного из самых быстрых алгоритмов экономного кодирования –

алгоритма LZRW1, а третий (алгоритм LZP3) оказывается не менее хорошей альтернативой широко используемым на практике комбинированным словарным алгоритмам семейства LZ77. Алгоритм LZP1 является самым быстрым алгоритмом группы LZP и в то же время самым малоэффективным. Каждый последующий алгоритм группы более эффективен и менее производителен в сравнении со своими предшественниками. Самым выгодным с точки зрения эффективности и, соответственно, самым медленным является алгоритм LZP4.

Основной структурой данных в алгоритмах группы LZP является хэш-таблица, поддерживаемая на этапах кодирования и декодирования. В отличие от хэш-таблицы используемой в алгоритме LZRW4, данная хэш-таблица содержит ровно один указатель в каждой своей ячейке. В алгоритме LZRW4 индекс хэширования вычислялся по двум обработанным символам. В алгоритмах группы LZP при вычислении учитывается от 3 до 5 символов (говорят, что хэш-индекс вычисляется по контексту соответствующего порядка). Особое внимание в алгоритмах группы LZP уделяется коллизиям, возникающим при хэшировании. Если требуется добиться максимальной производительности, то коллизии не избегаются. Это означает, что в процессе поиска совпадения не принимается во внимание возможное различие между текущим контекстом и контекстом, имеющим тот же индекс хэширования (в алгоритме LZRW4 использовался именно такой подход). Разрешение коллизий производится только в случае, когда определяющим фактором является не производительность, а эффективность кодирования. (Разрешение коллизий в терминах алгоритма LZP называется *контекстным подтверждением* (*context confirmation*)).

Главное различие между алгоритмами группы LZP заключается в способе кодирования. В алгоритмах LZP1 и LZP2 незакодированные символы и совпадения различаются служебными комбинациями, которые, вопреки сложившейся традиции, не являются однобитовыми. Служебные комбинации, одна из которых соответствует одиночному совпадению, а другая – незакодированному символу, за которым следует совпадение, включают в себя сразу два бита («00» и «01»), а служебная комбинация, соответствующая двум идущим подряд незакодированным символам, состоит из одного бита («1»). Данный необычный способ различения кодовых комбинаций обосновывается тем, что кодовые комбинации, кодирующие совпадения, как правило, генерируются реже, чем кодовые комбинации, представляющие собой незакодированные символы.

Длины совпадений в алгоритмах LZP1 и LZP2 кодируются с использованием статической системы префиксных кодов переменной длины, в которой каждой длине заранее ставится в соответствие конкретная кодовая комбинация, не меняющаяся в процессе кодирования. Для обработки незакоди-

рованных символов в алгоритме LZP2 дополнительно применяется полуадаптивное кодирование Хаффмана (накопление статистической информации производится в модели нулевого порядка).

В алгоритме LZP3 выбор способа кодирования обусловлен новым подходом к поиску совпадений. При использовании данного подхода действует вышеупомянутый механизм контекстных подтверждений. Если в результате поиска по вычисленному хэш-индексу не удается найти контекст, идентичный текущему контексту некоторого порядка, то производится переход к рассмотрению контекста следующего по убыванию порядка. Процесс начинается с рассмотрения контекста 4-го порядка и при необходимости продолжается вплоть до контекста 2-го порядка. Важным является то, что, в отличие от метода PPM, в алгоритме LZP3 при контекстном спуске не генерируются символы перехода, так как точно такой же «спуск» без какой-либо дополнительной информации может осуществляться и при декодировании. Чтобы понять, почему это становится возможным, следует осознать принципиальную разницу между механизмом, использованным в методе PPM, и механизмом, применяемым в алгоритме LZP3. В методе PPM переход к рассмотрению контекста меньшего порядка осуществляется тогда, когда с помощью этого контекста невозможно оценить обрабатываемый символ (символ не встречался ранее в данном контексте), а в алгоритме LZP3 – тогда, когда при поиске не был найден идентичный контекст.

Еще одной специфической особенностью алгоритма LZP3 является то, что в случае появления совпадения следующий за совпадающей строкой символ в обязательном порядке представляется в незакодированном виде (дополнительный служебный бит при этом не используется). В этом можно заметить явное сходство алгоритма LZP3 с алгоритмом LZ77. Такой подход часто оправдан с учетом малой вероятности появления одного совпадения в контексте другого.

Служебные биты и длины совпадений в алгоритме LZP3 кодируются с применением алгоритма Хаффмана, причем с учетом порядков контекстов, используемых для подтверждения или опровержения факта совпадения. Поэтому для контекстов каждого из трех допустимых порядков (2, 3 и 4) поддерживается отдельная статистическая модель (нулевая модель). Для упрощения процедуры обработки служебные биты кодируются блоками – в каждый блок объединяется 4 служебных бита. Кодирование Хаффмана применяется в алгоритме LZP3 и при обработке незакодированных символов. Для сбора статистической информации здесь используется так называемая *модель порядка 1–0* (подробное описание приведено в [19]).

Исходя из приведенного краткого описания алгоритмов группы LZP, можно произвести их классификацию. Алгоритм LZP1 в целом можно считать самостоятельным словарным алгоритмом. Фактически он представляет

собой модифицированный вариант алгоритма LZRW4. Алгоритмы LZP2 и LZP3 являются комбинированными словарным алгоритмами – они в равной мере сочетают себе и словарный, и статистический подходы (в последнем случае имеется в виду префиксное кодирование, базирующееся на статистической модели). Алгоритм LZP4, как было сказано, стоит скорее рассматривать как статистический алгоритм, хотя изначально он создавался как алгоритм словарной группы.

Так же, как и алгоритм LZRW4, алгоритмы группы LZP наиболее выгодны в ситуации, когда обрабатываемая информации обладает большой избыточностью (к такой информации, в частности, относится текстовая информация). Так, алгоритм LZP3 является одним из лучших среди словарных алгоритмов по соотношению между эффективностью и производительностью кодирования такой информации. (Для сравнения отметим, что при приблизительно равной эффективности данный алгоритм в два-три раза опережает по скорости кодирования алгоритм DEFLATE – комбинированный словарный алгоритм семейства LZ77, лежащий в основе утилиты сжатия GZIP.) К сожалению, помимо достоинств, алгоритмам группы LZP присущ и серьезный недостаток, обычно не свойственный словарным алгоритмам, – низкая скорость декодирования, почти совпадающая со скоростью кодирования. Этот недостаток является наиболее существенным препятствием на пути к практическому распространению алгоритмов рассмотренной группы.

### 5.3. Контекстные методы

В главе 2 было введено одно из базовых понятий теории экономного кодирования *марковская цепь*. Сущность марковской цепи как процесса порождения информации заключается в наличии взаимосвязи между сообщением и его предысторией, то есть, придерживаясь строгой терминологии, между сообщением и его контекстом. Однако, если сообщения зависят от контекстов, в которых они появляются, можно утверждать, что они также зависят и от следующих за ними сообщений. В связи с этим целесообразно несколько расширить понятие контекста и называть контекстом сообщения не только то, что непосредственно предшествует ему, но также и то, что непосредственно следует за ним. Для проведения различия между этими, вообще говоря, неидентичными объектами назовем первый из них *левосторонним контекстом*, а второй – *правосторонним контекстом*. Главное, что отличает два данных типа контекстов – это хронологический порядок их следования: левосторонний контекст некоторого сообщения всегда предшествует его правостороннему контексту.

Итак, информационное сообщение может зависеть в вероятностном (комбинаторном) смысле как от своего левостороннего контекста, так и от своего правостороннего контекста. О такой же зависимости можно говорить и применительно к левостороннему и правостороннему контекстам, являющимся контекстами друг относительно друга. Информационный поток представим в виде конкатенации двух таких примыкающих друг к другу контекстов, каждый из которых мы будем называть *контекстным дополнением* для другого (рис. 5.17). Такое представление, очевидно, не единственно, так как оно может быть произведено относительно произвольной позиции в потоке.

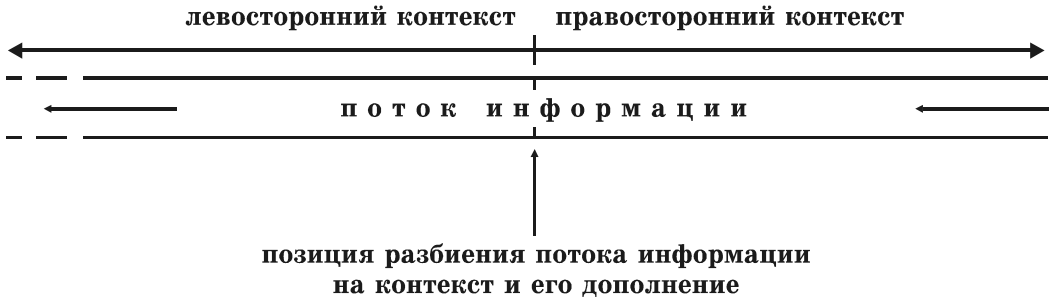



Рис. 5.17. Представление потока информации в виде конкатенации двух контекстов

Данная глава посвящена рассмотрению двух методов экономного кодирования, в основу которых был положен учет зависимости между символьным контекстом и его дополнением. Несмотря на идейную общность, мы имеем дело с совершенно разными подходами, в одном из которых рождение (генерация) обрабатываемой информации рассматривается как последовательный процесс, а в другом – как единовременный акт. Метод ACB, представляющий первый подход, предназначен для последовательной обработки информации, поступающей на вход обработчика в соответствии с условной хронологией ее рождения. Так же, как и большинство рассмотренных методов, метод ACB относится к *поточным* методам экономного кодирования. Представителем второго, менее стандартного подхода является метод Барроуза-Уилера. Особенность данного метода – непоследовательная обработка информации, которая возможна только при наличии на любом этапе работы всего объема обрабатываемой информации. Указанные методы являются современными высокоэффективными методами экономного кодирования, уже сегодня применяемыми в реальных приложениях и имеющими, по мнению автора данной книги, неплохую перспективу для дальнейшего практического использования.

### 5.3.1. Ассоциативное кодирование (метод ACB)

В статье [1] Г. Буяновским был предложен алгоритм экономного кодирования, получивший название *ассоциативное кодирование*. Всевозможные модификации данного алгоритма принято рассматривать в рамках единого метода ACB (Associative Coding of Buyanovsky – «ассоциативное кодирование Буяновского»).

Принцип работы метода ACB достаточно прост. На каждом этапе кодирования обрабатывается некоторая часть текущего правостороннего контекста (текущий контекст начинается с первой необработанной позиции). Длина кодируемой части контекста соответствует длине наибольшего совпадения[1] данного контекста с уже обработанными правосторонними контекстами (начальные позиции этих контекстов находятся левее первой необработанной позиции). Код генерируется с учетом следующих параметров:

- длины наибольшего совпадения;
- длины второго по величине совпадения;
- длин совпадений контекстного дополнения обрабатываемого контекста с различными контекстными дополнениями обработанных правосторонних контекстов.

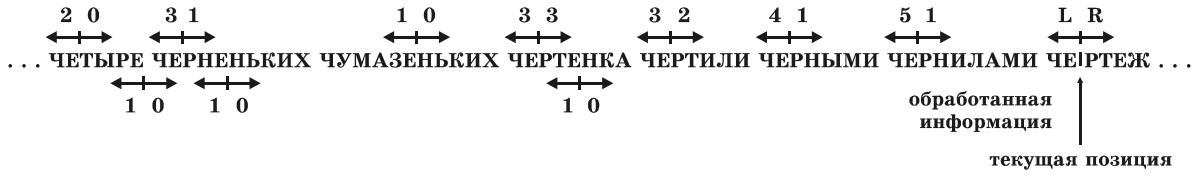
На рис. 5.18 приведен пример, иллюстрирующий процесс обработки потока информации методом ACB. Как видно, здесь приведены два лексикографически упорядоченных списка, содержащие различные варианты разбиения информационного потока на контекст и его дополнение. Для удобства рассмотрения в списки включены только те разбиения, которым соответствуют левосторонние или правосторонние контекстные совпадения ненулевой длины.

Один из списков упорядочен по левостороннему контексту, а другой – по правостороннему. Каждый из списков содержит разбиения информационного потока на контексты определенной направленности, частично совпадающие (можно сказать, ассоциирующиеся) с текущим обрабатываемым контекстом той же направленности (текущее контекстное разбиение заключено в рамку). Будем называть эти списки соответственно *левосторонним* и *правосторонним ассоциативными списками*[2]. Отметим, что реальное построение ассоциативного списка, вообще говоря, не является обязательным атрибутом метода ACB.

---

[1] Длина совпадения определяется как длина идентичной части сравниваемых контекстов (идентичная часть находится в начале контекстов).

[2] В [1] эти списки именуются *воронками аналогий.*




| Длина левостороннего совпадения (L) | Левосторонний ассоциативный список | Правосторонний ассоциативный список | Длина правостороннего совпадения (R) |
|---|---|---|---|
| 1 | УМАЗЕ\|НЬК | ЕТЫРЕ\| ЧЕ | 0 |
| 1 | ЧЕРНЕ\|НЬК | ЧЕРТЕ\|НКА | 0 |
| 1 | ЕТЫРЕ\| ЧЕ | УМАЗЕ\|НЬК | 0 |
| 1 | ЧЕРТЕ\|НКА | ЧЕРНЕ\|НЬК | 0 |
| 2 | . . . ЧЕ\|ТЫР | РЕ ЧЕ\|РНЕ | 1 |
| 3 | КА ЧЕ\|РТИ | МИ ЧЕ\|РНИ | 1 |
| 3 | РЕ ЧЕ\|РНЕ | ЛИ ЧЕ\|РНЫ | 1 |
| 4 | ЛИ ЧЕ\|РНЫ | ИХ ЧЕ\|РТЕ | 3 |
| | МИ ЧЕ\|РТЕ | МИ ЧЕ\|РТЕ | |
| 5 | МИ ЧЕ\|РНИ | КА ЧЕ\|РТИ | 2 |
| 3 | ИХ ЧЕ\|РТЕ | . . . ЧЕ\|ТЫР | 0 |

Рис. 5.18. Иллюстрация работы ассоциативного кодирования

Код обрабатываемой части контекста в методе ACB, как правило, состоит из двух компонентов. Первый из них отвечает за местоположение в том или ином ассоциативном списке правостороннего контекста, имеющего наибольшее совпадение с текущим правосторонним контекстом. Второй компонент предназначен для описания длины этого совпадения. Представление позиции совпадения, как правило, осуществляется с использованием кодов переменной длины, строящихся на основе статистической модели, в которой вероятности заменяются на длины левосторонних контекстных совпадений (для корректной замены длины необходимо нормировать). Однако можно воспользоваться и более простым методом, указав в качестве местоположения наиболее длинного правостороннего совпадения смещение позиции соответствующего ему разбиения в левостороннем ассоциативном списке относительно позиции разбиения, которое соответствует наиболее длинному левостороннему совпадению (в рассматриваемом примере смещение разбиения – «ИХ ЧЕ|РТЕ» относительно позиции разбиения – «МИ ЧЕ|РНИ» равно −1). Эффективность данного метода существенно повышается, если учитывается неравномерность в распределении смещений (при кодировании источника с памятью маленькие по абсолютному значению смещения встречаются чаще больших смещений, так как наиболее длинные левосторонние совпадения очень часто являются контекстными дополнениями для достаточно длинных правосторонних совпадений). Отметим, что использование последнего подхода сопряжено с необходимостью реального построения левостороннего ассоциативного списка.

Специфика метода ACB позволяет эффективно кодировать длины совпадений. Текущее разбиение всегда находится в правостороннем ассо-

циативном списке между такими двумя разбиениями, правосторонние контексты в которых имеют самые большие совпадения с обрабатываемым контекстом. Ясно, что если данные правосторонние контексты частично совпадают друг с другом, то их общая часть является началом обрабатываемого правостороннего контекста (в нашем примере общая часть контекстов «РТЕ ...» и «РТИ ...» – «РТ» является началом обрабатываемого контекста «РТЕ ...») Таким образом, все три контекста могут иметь ненулевое совпадение. Длину этого совпадения нет смысла включать в кодовую комбинацию, так как она может быть получена в процессе декодирования на основании знания двух разбиений, соседствующих в правостороннем ассоциативном списке с текущим разбиением. Одно из этих разбиений становится известным благодаря включаемой в кодовую комбинацию позиции наиболее длинного правостороннего совпадения. Второе разбиение находится в ассоциативном списке либо до, либо после первого разбиения, и поэтому для его идентификации достаточно одного различающего бита.
На данном этапе нам удалось закодировать только часть длины наибольшего совпадения. Оставшуюся часть (остаток в нашем примере равен одному символу) следует непосредственно включить в кодовую комбинацию[1]. (При этом опять же целесообразно учесть статистические особенности появления различных остатков длин.)

Выше рассмотрен один из возможных подходов к реализации метода АСВ, но возможны и другие подходы. Так, например, можно вообще не кодировать остаток длины совпадения, ограничившись кодированием только общей части. При таком подходе имеет смысл несколько изменить весь способ кодирования и вместо позиции правостороннего контекста, имеющего наибольшее совпадение с текущим правосторонним контекстом, включать в кодовую комбинацию местоположение интервала между двумя элементами правостороннего ассоциативного списка, в котором должно находиться текущее разбиение (в данном случае местоположение правостороннего контекста, имеющего наибольшее совпадение с кодируемым контекстом, не имеет значения). Местоположение интервала может быть эффективно представлено кодом переменной длины, получаемым с учетом длин совпадений левосторонних контекстов в примыкающих к интервалу разбиениях с текущим левосторонним контекстом (в [1] в качестве параметра, по которому производится учет, предлагается брать среднее арифметическое этих длин). Преимущество нового способа кодирования состоит в полном отказе от использования битов различия, а недостаток – в необходимости реального построения правостороннего ассоциативного списка.

---

[1] Если обрабатываемая информационная последовательность является бинарной, остаток длины совпадения может быть представлен более эффективно (см. [1]).

При рассмотрении процесса кодирования до сих пор не затрагивался случай полного отсутствия правосторонних совпадений. Специально для данного случая в множестве допустимых позиций контекстов (интервалов) предусматривается некоторая реально не существующая служебная позиция. Использование этой позиции как части кода указывает на отсутствие совпадения (в данном случае для обработки очередной порции информации необходимо использовать какой-то другой метод кодирования). Формально служебная позиция может соответствовать местоположению обрабатываемого контекста.

В практических реализациях метода ACB приходится прибегать к некоторым ограничениям, позволяющим снизить системные требования и повысить производительность. Во-первых, ограничивается объем обработанной информации, анализируемой при кодировании каждого контекста, а во-вторых, из рассмотрения полностью исключаются те разбиения, в которых левосторонний контекст совпадает по длине с текущим левосторонним контекстом на величину, меньшую некоторого наперед заданного значения. Если условиям последнего ограничения не удовлетворяет ни один из обработанных левосторонних контекстов, первый символ обрабатываемого правостороннего контекста либо записывается в его исходном (незакодированном) представлении, либо кодируется некоторым статистическим методом, как это обычно делается в комбинированных словарных схемах.

На первый взгляд, метод ACB представляет собой достаточно неординарное решение, что, впрочем, неудивительно с учетом не совсем обычной терминологии, используемой при его описании. В действительности же мы имеем дело с хорошо известным подход. Метод ACB является не более чем грамотной реализацией контекстно-зависимого кодирования в рамках словарной модели. Конечно, здесь может возникнуть вполне резонный вопрос: где же все атрибуты, присущие словарному подходу? Попытаемся на него ответить. Основным атрибутом словарного алгоритма, как известно, является словарь. В качестве словаря в методе ACB всегда можно рассматривать всю обработанную информацию. В процессе кодирования текущий правосторонний контекст, то есть обрабатываемая последовательность символов, кодируется ссылками на уже обработанные правосторонние контексты, то есть на обработанные символьные последовательности (вспоминая словарный подход, – строки), которые составляют обработанную часть информации, то есть словарь. Таким образом, метод ACB обладает всеми атрибутами словарного метода, и, с учетом используемого способа кодирования, его вполне можно поставить в один ряд с такими словарными алгоритмами, как LZRW4 и LZP.

Метод ACB является одним из самых эффективных методов экономного кодирования информации. По данному показателю он редко уступает, а

подчас и опережает методы статистической группы. Для примера, усредненный результат, полученный при обработке текстовых файлов book1 и book2 набора Calgary утилитой сжатия и архивации ACB версии 2.00 (усовершенствованная версия алгоритма, описанного в [1]), составил 2.13 бит/сим, а усредненный результат, полученный при обработке данной утилитой файлов obj1 и obj2, составил 2.85 бит/сим. К сожалению, высокая эффективность в данном случае достигается с привлечением существенных вычислительных ресурсов. Прежде всего это связано с трудоемкостью поиска контекстных совпадений разной направленности, а также необходимостью построения ассоциативных списков. На сегодняшний день по производительности метод ACB не может сравниться ни со статистическими методами, ни, тем более, с методами словарной группы. Остается надеяться, что в большей степени данная ситуация обусловлена недостатками существующих реализаций рассмотренного метода.

### 5.3.2. Метод Барроуза-Уилера

Мы приступаем к рассмотрению, вероятно, самого необычного метода экономного кодирования – метода Барроуза-Уилера (метод был впервые описан М. Барроузом и Д. Уилером в работе [23]). В основе метода Барроуза-Уилера лежит одноименное преобразование (Burrows-Wheeler Transformation – BWT), которому в методе отводится роль главной информационной модели. Преобразование позволяет представлять обрабатываемую информацию в особой форме, идеально подходящей для последующего эффективного кодирования. Необычность подхода состоит в наличии фактически двух этапов моделирования: первый этап – это работа преобразования, направленная на получение «удобного» информационного представления, а второй – построение вспомогательной модели, на основе которой будет закодировано данное представление.

Первая часть изложения метода Барроуза-Уилера посвящена рассмотрению самого преобразования. В дальнейшем затрагиваются вопросы, непосредственно касающиеся кодирования, а также некоторые аспекты программной реализации метода.

Предположим, что нам требуется эффективно закодировать некоторый блок информации длины $N$. Поместим данный блок в циклический буфер размера $N$. Если на время «забыть» о том, где находится начало исходного информационного блока в этом буфере, информационный блок превратится в замкнутую символьную последовательность – так называемое *информационное кольцо*. Информационное кольцо обладает тем свойством, что любой контекст внутри него всегда состоит из объединения бесконечного числа одинаковых последовательностей длины $N$. Так как уникальность контекста определяется первыми $N$ символами, то имеет смысл под контекстом в

информационном кольце понимать последовательность из первых $N$ символов. Контекст в новом определении имеет следующую важную особенность: его последний символ является одновременно первым символом его контекстного дополнения.

На рис. 5.19 приведено информационное кольцо для слова «ОБОРОНОСПОСОБНОСТЬ», а также два лексикографически упорядоченных контекстных списка, один из которых содержит левосторонние контексты, а другой – правосторонние. Особый интерес в каждом из этих списков для нас представляет столбец, состоящий из последних символов контекстов, обозначенный как **L**. Покажем, что каждый список может быть полностью восстановлен по данному столбцу.


Рис. 5.19. Преобразование Барроуза-Уилера

Столбец, состоящий из первых символов контекстов, получается из столбца **L** простой сортировкой символов. В результате мы знаем все пары идущих подряд символов, встречающиеся в информационном кольце. Если отсортировать эти пары по старшему символу, которым является символ, принадлежащий столбцу **L**, то столбец из младших символов пар после сортировки будет соответствовать столбцу вторых символов контекстов. На данном этапе становятся известны все триады идущих подряд символов, встречающиеся в информационном кольце. Для получения следующего столбца списка триады также необходимо отсортировать по старшему символу. Искомым столбцом будет опять же являться столбец из младших сим-

волов триад после сортировки. Действуя и далее в соответствии с описанной процедурой[1], мы рано или поздно восстановим весь список. Заметим, что на любом шаге данной процедуры строки переставляются одним и тем же способом, так как каждый раз сортируются символы одного и того же столбца **L**.

Итак, по столбцу **L** в любом из списков можно восстановить сам список, а следовательно, и информационное кольцо, представленное в любой из строчек данного списка. Однако оказывается, что для восстановления информационного кольца по столбцу **L** совсем не обязательно производить описанную сортировку. Для этого потребуется воспроизвести только один столбец, состоящий из первых символов контекстов (получение столбца фактически сводится к подсчету числа появлений каждого символа информационного алфавита в столбце **L**). Как было сказано, данный столбец и столбец **L** в совокупности порождают все пары идущих подряд символов. Для восстановления информационного кольца достаточно определить порядок следования этих пар в кольце. Как определяется порядок следования, становится понятно на следующем примере.

Попытаемся восстановить информационное кольцо, соответствующее слову «ОБОРОНОСПОСОБНОСТЬ», по двум указанным столбцам в левостороннем контекстном списке (имеется в виду упорядоченный список, содержащий левосторонние контексты)[2]. В столбце первых символов контекстов символ «Т» встречается один раз. В столбце **L** на той же позиции находится символ «Ь». Это означает, что в восстанавливаемом информационном кольце за символом «Т» следует символ «Ь». Символ «Ь» в столбце первых символов контекстов встречается также один раз, и ему соответствует символ «О». Теперь мы знаем, что частью информационного кольца является последовательность «ТЬО». На данном этапе возникает проблема, связанная с определением следующего символа последовательности. Дело в том, что символ «О» встречается в столбце первых символов 7 раз. Таким образом, следующий символ последовательности может находиться на одной из 7 возможных позиций. В зависимости от выбора позиции этим символом может оказаться один из следующих символов: «Р», «С», «Н» и «Б». Как определить нужную позицию? Если посмотреть на уже составленный контекстный список, становится понятно, что искомой позицией будет та, которая соответствует левостороннему контексту «...СТЬО|Б...», поскольку только в этом контексте перед символом «О» находится символ «Ь». Однако

---

[1] Данная процедура носит название *поразрядная сортировка* (см. [2]).

[2] Описываемый алгоритм работает и в случае, когда восстановление производится по двум столбцам правостороннего списка. При этом в него необходимо внести соответствующие корректировки, связанные с изменением направления восстановления.

на данном этапе восстановления список, очевидно, не может быть составлен. Что же делать в подобной ситуации? Как оказывается, существует достаточно простое решение. Заметим, что взаимное расположение контекстов, оканчивающихся на символ «О», совпадает со взаимным расположением контекстов, полученных циклическим сдвигом исходных контекстов на один символ влево (при таком сдвиге последний символ контекста становится его первым символом). Следовательно, так как контекст «...ОСТЬ|О...» находится в контекстном списке ниже других контекстов, оканчивающихся на символ «О», то среди контекстов, начинающихся с символа «О», ему будет соответствовать также самый нижний контекст списка – «...СТЬО|Б...». В свою очередь, контекст «...СТЬО|Б...» лежит ниже контекста «...ПОСО|Б...», и поэтому в качестве следующего символа надо брать символ «О», являющийся последним символом контекста «...ТЬОБ|О...», а не символ «Н», являющийся последним символом контекста «...ОСОБ|Н...» (восстанавливаемая последовательность теперь приобретает вид «ТЬОБО»). Легко проверить, что, если и далее следовать выбранной методике восстановления, в конце концов восстанавливаемая последовательность превратится в искомое информационное кольцо[1].

Для полного восстановления исходного блока информации достаточно запомнить местоположение какой-либо позиции блока в информационном кольце. Лучше всего хранить позицию первого символа исходного блока информации в столбце **L** контекстного списка. Благодаря такому подходу, исходный информационный блок может быть последовательно восстановлен, начиная с первой позиции. (На рис. 5.19 позиции первого символа слова «ОБОРОНОСПОСОБНОСТЬ» в столбцах **L** левостороннего и правостороннего контекстных списков отмечены стрелками.)

Получение столбца **L** и определение в данном столбце позиции первого символа исходного информационного блока есть суть преобразования Барроуза-Уилера. Как следует из приведенных выше рассуждений, данное преобразование является обратимым. Остается выяснить, в чем же состоит преимущество нового представления информации – столбца **L** – по сравнению с ее исходным представлением.

Как мы знаем, особенность информации, порождаемой источником с памятью, состоит в том, что символы в такой информации зависят от кон-

[1] Данная методика восстановления информационного кольца применима во всех ситуациях за одним исключением: информационное кольцо не может быть правильно восстановлено описанным способом в случае, когда обрабатываемая информация состоит из объединения нескольких одинаковых информационных блоков (примером может служить слово «МАМА»). Указанное ограничение, естественно, легко устранимо: в обрабатываемую информацию всегда можно внести изменения, позволяющие избежать неприятностей при восстановлении.

текстов, в которых они появляются. Отсюда, в частности, следует, что вероятность совпадения символов, имеющих частично совпадающие контексты, достаточно велика. (Как правило, эта вероятность тем выше, чем больше длина совпадения между контекстами.) Преобразование Барроуза-Уилера позволяет группировать символы исходного информационного блока по схожести их контекстов. Как результат, представление информации, полученное с помощью преобразования Барроуза-Уилера (в дальнейшем – *S-представление*) обладает свойством *локальной идентичности*, которое заключается в том, что с уменьшением расстояния между позициями в представлении повышается вероятность совпадения символов, находящихся на этих позициях.

Обратимся к рассмотренному примеру (рис. 5.19). Даже несмотря на небольшой размер исходного блока информации, его S-представлениям присущи определенные особенности, являющиеся проявлением указанного свойства. Так, в столбцах **L** каждого контекстного списка почти все символы «О» расположены группами. То же заключение, но применительно к каждому списку в отдельности, можно сделать и касательно некоторых других символов («С», «Б», «Н»). Эти особенности можно объяснить закономерным появлением отдельных букв в слове «ОБОРОНОСПОСОБНОСТЬ». Например, буква «С» всегда следует за буквой «О», а буква «Н» всегда предшествует букве «О».

В более явном виде свойство локальной идентичности проявляется у S-представлений достаточно больших информационных блоков. В следующем примере в качестве исходной информации выступает приводившийся в главе 1 отрывок стихотворения Есенина.

**Исходная информация:**
*Лицом_к_лицу↓Лица_не_увидать.↓Большое_видится_на_расстоянье.*
*↓Когда_кипит_морская_гладь,↓Корабль_в_плачевном_состоянье.*

**S-представление (сортировка по левостороннему контексту):**
*↓Л↓↓крнпувлскмвгноооии_тядч_сб_ли_нилдьаиа__в..цддтпцтци_аиьаьа*
*о__еаоььлргрмсяямелиаасостктяьс_оов↓ауоео,_шее._нн_ККБЛ*
**(позиция начала исходной информации в S-представлении – 2)**

**S-представление (сортировка по правостороннему контексту):**
*ььееьеямактяавме↓↓↓↓дцнрлрдлка_у_ео_гииаььончввкпдЛЛл_с_гп_бооо*
*___вяяКшБцнКмстти_о_ор_асотиссиа_цииаьдтлннласоо..,у.*
**(позиция начала исходной информации в S-представлении – 53)**

В данном случае мы имеем примеры классических S-представлений, включающих в себя как относительно стационарные участки, на которых симво-

лы почти не меняются, так и участки с более или менее заметной динамикой изменения. Примечательно, что преобразование Барроуза-Уилера позволило наглядно выявить многие закономерности, отмеченные в гл. 1. К примеру, соседнее расположение нескольких заглавных букв в первом представлении отражает обязательность их следования вслед за символом смены строки «↓».

Эффективное кодирование S-представлений может быть осуществлено с использованием локально адаптивного статистического кодирования (слово «локально» использовано для того, чтобы подчеркнуть необходимость адаптации статистической модели к информационным особенностям небольших участков обрабатываемой информации). Однако перед применением такого кодирования целесообразно подвергнуть S-представление какому-либо дополнительному преобразованию (или преобразованиям). В качестве такого преобразования чаще всего используется преобразование MTF[1].

Работа преобразования MTF заключается в замещении символа номером его позиции в пронумерованной таблице, содержащей символы информационного алфавита. Символы замещаются последовательно, начиная с начала или с конца S-представления. Замещение сопровождается перестройкой таблицы, производимой следующим образом: замещенный символ перемещается на первую позицию в таблице, а все символы, находившиеся в момент замещения на позициях с номерами, меньшими номера позиции замещаемого символа, сдвигаются на одну позицию в направлении от начала таблицы. Таким образом, если какой-то символ недавно выступал в качестве замещаемого символа, то он будет находиться близко к началу таблицы. Данная особенность преобразования MTF позволяет в численной форме представлять закономерности S-представлений. Для пояснения применим преобразование MTF к последнему S-представлению отрывка стихотворения. В результате получается следующая последовательность номеров позиций (начальное распределение символов по позициям в таблице соответствует стандартной кодировке, используемой в операционной системе MS-DOS): «237, 1, 167, 1, 2, 2, 240, 175, 165, 174, 229, 5, 4, 169, 2, 6, 7, 245, 1, 1, 1, 172, 234, 179, 230, 180, 2, 5, 3, 13, 10, 110, 232, 2, 11, 182, 3, 178, 181, 1, 7, 19, 1, 6, 13, 235, 18, 1, 12, 184, 15, 160, 1, 16, 14, 233, 2, 15, 7, 3, 182, 13, 1, 1, 3, 1, 1, 11, 23, 1, 162, 236, 155, 24, 18, 5, 26, 14, 27, 1, 23, 12, 13, 2, 2, 26, 3,

[1] Преобразование MTF является частью метода, впервые предложенного Б. Я. Рябко [6]. (Метод получил название «Сжатие с помощью стопки книг».) Общепринятая аббревиатура MTF (Move To Front - «двигай вперед») была изначально использована в [15] для обозначения указанного метода, повторно открытого Д. Бентли, Д. Слейтером, Р. Тарьяном и В. Веи. Наиболее удачной альтернативой преобразованию MTF является так называемое *дистанционное кодирование* (Э. Биндер, 1999 г).

24, 7, 5, 7, 7, 4, 1, 2, 5, 6, 11, 4, 1, 1, 4, 24, 22, 8, 21, 14, 1, 2, 6, 10, 11, 1, 74, 1, 73, 28, 3».

Как видно, маленькие номера встречаются в данной последовательности значительно чаще, чем большие. Неравномерность в распределении номеров позиций является прямым следствием наличия у S-представления свойства локальной идентичности. Наилучшим образом учитывать данную неравномерность позволяет статистическое кодирование, построенное на основе нулевой модели.

Описанный метод экономного кодирования был открыт Д. Уилером в 1983 году. По странному стечению обстоятельств первая публикация данного по сути революционного подхода датируется аж 1994 годом! Трудно вообразить, как бы происходило развитие теория экономного кодирования, если бы совместная работа М. Барроуза и Д. Уилера [23] появилась не в середине 90-х, а в первой половине 80-х, в то время, когда еще не был разработан практически ни один из хорошо известных на сегодня методов (за исключением нескольких простейших алгоритмов словарной группы).

С момента своего появления метод Барроуза-Уилера привлекает повышенное внимание со стороны исследователей, занимающихся проблемами экономного кодирования. Следствием их усилий стало появление множества весьма эффективных реализаций данного метода. Оставшаяся часть пункта посвящена рассмотрению наиболее важных моментов этих реализаций.

Главной проблемой, возникающей при практическом воплощении метода Барроуза-Уилера , является проблема организации быстрого лексикографического упорядочивания контекстов. Здесь чаще всего применяются алгоритмы двух типов[1].

Алгоритмы первого типа базируются на «быстрой сортировке» Хоара или сортировке Шелла (см. [2, 23, 16]). Как правило, такие алгоритмы требуют для работы $5N$ единиц хранения информации, где $N$ – размер исходного блока информации (размер машинного слова считается равным 4 единицам хранения). Они обладают достаточно высокой производительностью, будучи примененными ко многим распространенным типам информации (например, к текстовой информации), однако в худшем случае их сложность может оцениваться как $O(N^2)$. Заметное падение производительности наблюдается, как правило, при обработке информации, содержащей повторяющиеся информационные последовательности большой длины. В подобных случаях для увеличения скорости сортировки полезно предварительно

[1] Другие существующие алгоритмы (см., например, [40, 55]) на практике обычно не используются.

обработать исходную информацию каким-либо специальным словарным алгоритмом, позволяющим «устранить» достаточно длинные совпадения[1].

В алгоритмах упорядочивания второго типа во время работы производится построение так называемого *суффиксного дерева*, представляющего собой дерево цифрового поиска, листовые узлы которого однозначно соответствуют различным контекстам, имеющимся в обрабатываемой информации (см. [11, 41]). Достоинством этих алгоритмов является их линейная сложность, а недостатком – повышенные требования к объему оперативной памяти (для работы требуется в среднем $10N$ единиц хранения информации, а в худшем случае – $12N$). Вследствие последнего, данные алгоритмы обычно не используются в коммерческих программных продуктах, а применяются в основном только в исследовательских целях.

Трудоемкость сортировки в алгоритмах обоих типов обусловлена необходимостью обработки большого количества достаточно длинных контекстов. В работе [58] М. Шиндлер показал, что блочное преобразование Барроуза-Уилера является обратимым, даже если контексты сортируются по нескольким первым символам. Это открытие позволяет использовать для упорядочивания контекстов высокопроизводительную поразрядную сортировку, применявшуюся нами при доказательстве обратимости преобразования Барроуза-Уилера. К сожалению, сложность обратного преобразования в данном случае оказывается сравнимой со сложностью прямого преобразования (сортировки)[2]. Кроме того, S-представление, полученное с помощью преобразования, предложенного Шиндлером, часто не может быть так же эффективно закодировано, как S-представление, полученное с помощью оригинального преобразования Барроуза-Уилера. Проигрыш в эффективности кодирования может составлять до нескольких процентов – в зависимости от глубины сортировки.

Помимо алгоритмов упорядочивания контекстов, особого внимания при рассмотрении метода Барроуза-Уилера, безусловно, заслуживают методики повышения эффективности кодирования S-представлений. Рассмотрим вкратце некоторые из них.

В работе П. Фенвика [28] предлагается разделить 256 номеров позиций, которые могут быть получены на выходе преобразования MTF (предполагается, что информационный алфавит содержит 256 символов), на 9 групп следующим образом: {0}, {1}, {2, 3}, {4,...,7}, {8,...,15}, {16,...,31}, {32,...,63}, {64,...,127}, {128,...,255}. Согласно Фенвику, каждый номер, принадлежащий последовательности номеров позиций, следует кодировать в

[1] Естественно, словарный алгоритм должен быть совместим с реализацией метода Барроуза-Уилера.

[2] При использовании полной сортировки скорость обратного преобразования в несколько раз превышает скорость прямого преобразования.

два этапа с применением так называемой *структурной модели*. На первом этапе необходимо кодировать индекс группы, в которой находится номер, а на втором, заключительном этапе – смещение номера в этой группе относительно ее начала (последний этап не распространяется на номера 0 и 1, так как они являются единственными номерами в своих группах). При кодировании индексов групп и смещений в различных группах предлагается использовать различные статистические модели. Модель, используемая при кодирования индексов групп, называется *моделью 1-го уровня*, а 7 моделей, используемые при кодирования смещений, соответственно именуются *моделями 2-го уровня*. Преимущество подхода, предложенного Фенвиком, заключается в том, что он учитывает степень репрезентативности статистической выборки: появление позиций с большими номерами, как правило, носит более случайный характер, нежели появление позиций с маленькими номерами; поэтому статистика появления маленьких номеров является более репрезентативной.

При применении для обработки последовательности, поступающей с выхода преобразования MTF, префиксного кодирования (обычно это кодирование Хаффмана) некоторого увеличения эффективности можно добиться, комбинируя его с алгоритмом RLE (Run-Length Encoding – «групповое кодирование»)[1]. Алгоритм RLE на самом деле является самостоятельным алгоритмом экономного кодирования, часто используемым для получения эффективных представлений графической информации. Работа алгоритма заключается в замене групп одинаковых символов (такие группы, как правило, составляют основу S-представлений) кодовыми парами типа <символ, количество символов в группе>. Алгоритм RLE необходимо применять либо непосредственно перед преобразованием MTF, либо непосредственно после него. Не вдаваясь в подробности реализации, отметим, что для представления компонентов кодовых пар здесь, как обычно, имеет смысл использовать коды переменной длины.

Несколько необычный метод кодирования S-представлений описывается в работе [12]. Авторы отмечают наличие особых зависимостей в появлении номеров позиций на выходе преобразования MTF. Фактически утверждается, что последовательность номеров позиций обладает памятью. Исходя из этого, для обработки данной последовательности предлагается использовать статистическую схему, основанную на использовании контекстной модели 3-го порядка.

В зависимости от выбора направленности сортируемых контекстов преобразование Барроуза-Уилера исходного информационного блока может

[1] Экспериментально установлено, что использование алгоритма RLE совместно с арифметическим кодированием не приводит к увеличению эффективности, но позволяет ускорить процесс кодирования.

быть осуществлено двумя различными способами. В связи с этим возникает естественный вопрос – какое из двух возможных S-представлений должно быть выбрано для последующей обработки из расчета на достижение максимальной эффективности? К сожалению, хоть сколько-нибудь полной теории на этот счет не существует. С уверенностью можно утверждать только то, что для получения S-представлений большинства разновидностей текстовой информации следует использовать сортировку, в которой участвуют правосторонние контекстам.

Помимо проблемы выбора направленности сортируемых контекстов существует еще одна немаловажная проблема, непосредственно связанная с особенностями преобразования Барроуза-Уилера. Преобразование Барроуза-Уилера является блочным преобразованием, что обусловливает необходимость разбиения обрабатываемой информации на блоки. Способ разбиения, как оказывается, во многом предопределяет эффективность кодирования. Поясним это на примере.

Предположим, что исходная информация получена конкатенацией нескольких информационных последовательностей, порожденных источниками с сильно различающимися свойствами. Следствием данного различия может стать то, что в одних и тех же контекстах, но в разных последовательностях будут появляться абсолютно разные символы. Объем закодированного S-представления такой неоднородной информации, вероятнее всего, превысит объем закодированной конкатенации S-представлений последовательностей, составляющих эту информацию, так как в среднем вероятность совпадения близко стоящих символов в S-представлении всей исходной информации будет меньше вероятности совпадения символов, находящихся по соседству в S-представлениях каждой последовательности. С другой стороны, совершенно очевидно, что в случае однородности информации ее нет смысла разделять на составляющие, поскольку дробление информации ведет к уменьшению длин стационарных и близких к стационарным участков в ее S-представлениях, что, естественно, совсем не способствует повышению эффективности кодирования. Таким образом, выбор разбиения должен зависеть прежде всего от особенностей обрабатываемой информации. К сожалению, на практике ограниченность машинных ресурсов не всегда позволяет реализовывать оптимальные подходы к разбиению.

Как можно было убедиться, метод Барроуза-Уилера являет собой весьма нестандартный подход к проблеме экономного кодирования. Неудивительно, что он отличается и достаточно необычными характеристиками. По скорости кодирования и по эффективности метод находится на уровне наиболее высокопроизводительных методов статистической группы, а по скорости декодирования – в два-три раза опережает эти методы. Для количественной оценки эффективности метода Барроуза-Уилера была выбрана

свободно распространяемая утилита DC версии 0.99, являющаяся его программной реализацией. Так же, как и в предыдущих тестах, кодированию были подвергнуты текстовые файлы book1 и book2 и объектные файлы obj1 и obj2, принадлежащие стандартному набору Calgary. Усредненные результаты, полученные при обработке файлов каждого типа, оказались равными 2.09 и 3.21 бит/сим соответственно. В первом случае проигрыш по сравнению с лучшим результатом составил 0.09 бит/сим, а во втором – 0.36 бит/сим. Приведенные результаты вполне отражают реальную ситуацию: метод Барроуза-Уилера один из лучших по соотношению между эффективностью и производительностью кодирования применительно к информации, обладающей большой избыточностью; текущие реализации метода пока не позволяют говорить о его оптимальности по отношению к другим типам информации.

## Заключение

Мы рассмотрели практически все основные методы экономного кодирования без потерь последовательной дискретной информации. Без внимания остались некоторые узкоспециализированные приемы, предназначенные для решения достаточно частных задач, и малоприменимые в более общих ситуациях.

Основываясь на рассмотренном материале, можно сделать вывод, что в основе большинства существующих методик экономного кодирования лежат два по сути весьма схожих подхода. Речь идет о вероятностном и комбинаторном подходах к описанию свойств информационных источников. Наибольшее предпочтение среди этих подходов, безусловно, следует отдать вероятностному подходу, так как комбинаторный подход, как мы могли убедиться на примере простейших алгоритмов словарной группы (см. п. 5.2.1, 5.2.2, 5.2.3), предоставляет значительно меньше возможностей для построения адекватных информационных моделей. В действительности комбинаторный подход является приближением к вероятностному подходу, в связи с чем вероятностный подход можно в *некоторой степени* рассматривать как обобщение комбинаторного подхода.

Последняя оговорка отнюдь не случайна. При всем вышесказанном надо четко осознавать, что вероятностная модель все же полностью никогда не заменит комбинаторную модель. Комбинаторная модель может быть в принципе выделена как вполне самостоятельная информационная модель. Таким образом, вероятностный и комбинаторный подходы, хотя и имеют много общего, существуют скорее независимо друг от друга. Очевидное преимущество такого положения состоит в возможности выбора. Однако, как оказывается, эта возможность не является тем фундаментом, на котором может базироваться решение любой проблемы в области экономного кодирования.

На примере возрастающей последовательности натуральных чисел «1, 2, 3, ... » легко убедиться в несостоятельности как вероятностного, так и комбинаторного подходов. В основе этих подходов строгая ориентация на повторение, причем не на *функциональное* повторение, которое мы имеем в данном случае, а на более простой тип повторения – на *номинальное* повторение. Номинальное повторение является всего лишь повторением информационной выборки, тогда как здесь наиболее существенен способ получения этой выборки. Совершенно очевидно, что для моделирования подобных ситуаций необходим некоторый альтернативный подход к описанию свойств обрабатываемой информации, более общий по сравнению с первыми двумя. В этом подходе должны учитываться не только вероятностные и комбинаторные закономерности, но и закономерности более высокого уров-

ня – *функциональные закономерности*. Этот третий, универсальный подход принято называть *алгоритмическим подходом*.

Алгоритмический подход является достаточно абстрактным подходом, поскольку его представителем по определению может быть любой метод экономного кодирования. Кодирование в рамках алгоритмического подхода всегда сводится к построению алгоритма генерации информационной последовательности. На практике такой алгоритм представляет собой программу, которая фактически выступает в роли кода этой последовательности.

Построение кодовых программ возможно только при наличии некоторого метода программирования. Такой метод, очевидно, должен быть максимально эффективен с точки зрения экономности представления различных программ. Однако из теории, развитой А. Н. Колмогоровым, следует, что эффективность метода программирования никогда не носит универсального характера – метод программирования может быть эффективен только по отношению к другому методу или по отношению к тому или иному классу информационных источников[1]. Так называемые *асимптотически оптимальные методы программирования* (см. [3]) на самом деле не являются таковыми в общем случае – их оптимальность определяется только потенциальной возможностью получения асимптотически оптимальных представлений информации в сравнении с представлениями, получаемыми в рамках других *отдельно взятых* методов программирования. Таким образом, проблема построения эффективных методов программирования не имеет общего решения.

Помимо указанной проблемы существует не менее серьезная проблема – нахождение оптимальных или близких к оптимальным программам в рамках различных методов программирования. С этой проблемой мы уже неоднократно сталкивались в предыдущем изложении. При рассмотрении различных методов экономного кодирования очень часто встречаются ситуации, в которых возникает необходимость в выборе того или иного конкретного алгоритмического решения. Как правило, мы пытаемся достичь оптимального результата по соотношению между эффективностью, производительностью и возможными затратами. Выбор наилучшего алгоритма в данном случае представляет собой не что иное, как поиск оптимальной программы. При этом в качестве метода программирования выступает метод экономного кодирования.

К сожалению, на сегодняшний день теория экономного кодирования практически не выходит за рамки простейшего комбинаторно-

[1] Это доказывает несостоятельность расхожих утверждений о существовании полностью универсальных методов экономного кодирования.

вероятностного подхода. Примечательно, что до сих пор не известно, насколько оправданно такое положение. С одной стороны, может оказаться, что комбинаторно-вероятностные методы являются достаточными в том смысле, что они идеально подходят для обработки большинства существующих информационных источников. С другой стороны, не исключено и обратное – ограниченность общепринятых методик не позволяет учитывать особенности этих источников в полной мере. Проверить последнюю догадку можно только путем дальнейшего совершенствования теории экономного кодирования, основой для которого должно стать развитие алгоритмического подхода к описанию свойств реальных источников информации.

## Литература


1. **Буяновский Г.** Ассоциативное кодирование // Монитор. – 1994. – № 8. – С. 10–15.
2. **Кнут Д.** Искусство программирования для ЭВМ В 3 т. Т. 3. Сортировка и поиск: Пер. с англ. – М.: Мир, 1978.
3. **Колмогоров А. Н.** Три подхода к определению понятия «количество информации» // Теория информации и теория алгоритмов. – М.: Наука, 1987. – С. 213–223.
4. **Кричевский Р. Е.** Сжатие и поиск информации. – М.: Радио и связь, 1989.
5. **Рябко Б. Я.** Дважды универсальное кодирование // Проблемы передачи информации. – 1984. – Т. 20, № 3. – С. 24–28.
6. **Рябко Б. Я.** Сжатие информации с помощью стопки книг // Проблемы передачи информации. – 1980. – Т 16, № 4. – С. 16–21.
7. **Хаффмен Д. А.** Метод построения кодов с минимальной избыточностью: Пер. с англ. // Кибернетический сборник. – М.: ИЛ, 1961. –Вып. 3. – С. 79–87.
8. **Шеннон К.** Математическая теория связи: Пер. с англ. // Работы по теории информации и кибернетике. – М.: ИЛ, 1963. – С. 243–332.
9. **Шеннон К.** Предсказание и энтропия печатного английского текста: Пер. с англ. // Работы по теории информации и кибернетике. – М.: ИЛ, 1963. – С. 669–686.
10. **Шкарин Д.** Практическая реализация алгоритма PPM. – http://sochi.net.ru/~maxime/doc/PracticalPPM.doc.gz.
11. **Balkenhol B., Kurtz S.** Universal Data Compression Based on the Burrows and Wheeler Transformation: Theory and Practice: Tech. rept. N 98-069 – Sonderforschungsbereich: Discrete Structuren int der Mathematik, Universitat Bielefeld, Bielefeld, Germany, 1998.
12. **Balkenhol B., Kurtz S, Shtarkov Y. M.** Modifications of the Burrows and Wheeler Data Compression Algorithm // Proc. IEEE Data Compression Conf. – Snowbird, Utah, USA, Mar. 29–31, 1999. – P. 188–197.
13. **Bell T. C.** A Unifying Theory and Improvements for Existing Approaches to Text Compression: Ph.D. dissertation. – Dept. of Computer Science, University of Canterbury, New Zealand, 1987.
14. **Bell T. C.** Better OPM/L Text Compression // IEEE Trans. Commun. – 1986. – Vol. 34, N 12. – P. 1176–1182.
15. **Bentley J. L., Sleator D. D., Tarjan R. E., Wei V. K.** A Locally Adaptive Data Compression Scheme // CACM. – 1986. – Vol. 29, N 4. – P. 320–330.

16. **Bentley J., Sedgewick R.** Fast Algorithms for Sorting and Searching Strings // Proc. ACM–SIAM Symp. on discr. algorithms, New Orleans, USA, Jan., 1997. – P. 360–369.
17. **Bloom C.** Algorithm PPMZ2. – http://www.cbloom.com/src/ppmz.html.
18. **Bloom C.** LZP: A New Data Compression Algorithm. – http://www.cbloom.com/papers/lzp.zip.
19. **Bloom C.** New Techniques in Context Modeling and Arithmetic Encoding. – http://www.cbloom.com/papers/dcc_cntx.zip.
20. **Bloom C.** Solving the Problems of Context Modeling. – http://www.cbloom.com/papers/ppmz.zip.
21. **Brent R. P.** A Linear Algorithm for Data Compression // Aust. Comput. J. – 1987. – Vol. 19, N 2. – P. 64–68.
22. **Bunton S.** On-Line Stochastic Processes in Data Compression: Ph.D. dissertation. – Dept. of Computer Science and Engineering, University of Washington, Seattle, Washington, USA, 1996.
23. **Burrows M., Wheeler D. J.** A Block-Sorting Lossless Data Compression Algorithm: Res. rept. 124. – DIGITAL Systems Research Center, 1994.
24. **Cleary J., Teahan W., Witten I.** Unbounded Length Contexts for PPM // Proc. IEEE Data Compression Conf. – Snowbird, Utah, USA, Mar. 28–30, 1995. – P. 52–61.
25. **Cleary J. G., Witten I. H.** Data Compression Using Adaptive Coding and Partial String Matching // IEEE Trans. Commun. – 1984. – Vol. 32, N 4. – P. 396–402.
26. **Cormack G. V., Horspool R. N.** Data Compression Using Dynamic Markov Modeling // Comput. J. – 1987. – Vol. 30, N 6. – P. 541–550.
27. **Fano R. M.** Technical N65. – The Research Laboratory of Electronics, MIT, Mar. 17, 1949.
28. **Fenwick P. M.** Block Sorting Text Compression // Proc. 19th Australian Computer Science Conf., Melbourne, Australia, Jan. 31 – Feb. 2, 1996.
29. **Fiala E. R., Greene D. H.** Data Compression with Finite Windows // CACM. – 1989. – Vol. 32, N 4. – P. 490–505.
30. **Gallager R. G.** Variations on a Theme by Huffman // IEEE Trans. Inform. Theory. – 1978. – Vol. 24, N 6. – P. 668–674.
31. **Golomb S. W.** Run-Length Encoding // IEEE Trans. Inform. Theory. – 1966. – Vol. 12, N 4. – P. 399–401.
32. **Guazzo M.** A General Minimum-Redundancy Source-Coding Algorithm // IEEE Trans. Inform. Theory. – 1980. – Vol. 26, N 1. – P. 15–25.
33. **Horspool R. N. Cormack G. V.** Dynamic Markov Modeling – A Prediction Technique // Proc. Int. Conf. on the System Sciences. – Honolulu, Hawaii, USA, 1986. – P. 700–707.

34. **Howard P. G.** The Design and Analysis of Efficient Lossless Data Compression Systems: Tech. rept. N CS-93-28. – Dept. of Computer Science, Brown University, Providence, Rhode Island, USA, 1993.
35. **Howard P. G., Vitter J. S.** Practical Implementations of Arithmetic Coding // **Storer A.** Image and text compression. – Kluwer Academic Publishers, Massachusetts, USA, 1992. – P. 85–112.
36. **Jakobson M.** Compression of Character String by an Adaptive Dictionary // BIT. – 1985. – Vol. 25, N 4. – P. 593–603.
37. **Jones D. W.** Application of Splay Trees to Data Compression // CACM. – 1978. – Vol. 31, N 8. – P. 996–1007.
38. **Krichevsky R. E., Trofimov V. K.** The Performance of Universal Coding // IEEE Trans. Inform. Theory. – 1981. – Vol. 27, N 1. – P. 199–207.
39. **Long P. M., Natsev A. I., Vitter J. S.** Text Compression via Alphabet Re-Representation // Proc. IEEE Data Compression Conf. – Snowbird, Utah, USA, Mar. 25–27, 1997. – P. 161–170.
40. **Manber U., Myers E. W.** Suffix Arrays: A New Method for On-Line String Searches // SIAM J. of Computing. – 1993. – Vol. 22, N 5. – P. 935–948.
41. **McCreight E. M.** A Space-Economical Suffix Tree Construction Algorithm // J. ACM. – 1976. – Vol. 23, N 2. – P. 262–272.
42. **Miller V. S., Wegman M. N.** Variations on a Theme by Ziv and Lempel // **Apostolico A., Galil Z.** Combinatorial Algorithms on Words. – Springer-Verlag, NY, USA, 1985. – P. 131–140.
43. **Moffat A. M.** A Note on the PPM Data Compression Algorithm: Res. rept. 88/7. – Dept. of Computer Science, University of Melbourne, Victoria, Australia, 1988.
44. **Moffat A. M.** Implementing the PPM Data Compression Scheme // IEEE Trans. Commun. – 1990. – Vol. 38, N 11. – P. 1917–1921.
45. **Moffat A., Neal R., Witten I. H.** Arithmetic Coding Revisited // Proc. IEEE Data Compression Conf. – Snowbird, Utah, USA, Mar. 28–30, 1995. – P. 202–211.
46. **Nelson M.** Data Compression Book. – M&T Books, Redwood City, California, USA, 1991.
47. **Pasco R.** Source Coding Algorithms for Fast Data Compression: Ph.D. dissertation. – Dept. of Electrical Engineering, Stanford University, California, USA, 1976.
48. **Pennebaker W. B., Mitchell J. L., Langdon G. G., Arps R. B.** An Overview of the Basic Principles of the Q-Coder Adaptive Binary Arithmetic Coder // IBM J. Res. Develop. – 1988. – Vol. 32, N 6. – P. 717–726.
49. **Rice R. F.** Some Practical Universal Noiseless Coding Techniques: JPL Publication 79-22. – Jet Propulsion Laboratory, Pasadena, California, USA, Mar., 1979.

50. **Rissanen J. J.** Generalized Kraft Inequality and Arithmetic Coding // IBM J. Res. Develop. – 1976, Vol. 20, N 3. – P. 198–203.
51. **Rissanen J. J., Langdon G. G.** Arithmetic Coding // IBM J. Res. Develop. – 1979. – Vol. 23, N 2. – P. 146–162.
52. **Rodeh M., Pratt V. R., Even S.** Linear Algorithm for Data Compression via String Matching // J. ACM. – 1981. – Vol. 28, N 1. – P. 16–24.
53. **Rubin F.** Arithmetic Stream Coding Using Fixed Precision Registers // IEEE Trans. Inform. Theory. – 1979. – Vol. 25, N 6. – P. 672–675.
54. **Rubin F.** Experiments in Text File Compression // CACM. – 1976. –Vol. 19, N 11. – P. 617–623.
55. **Sadakane K.** A Fast Algorithm for Making Suffix Arrays and for Burrows-Wheeler Transformation // Proc. IEEE Data Compression Conf. – Snowbird, Utah, USA, Mar. 30 – Apr. 1, 1998. – P. 129–138.
56. **Schmidhuber J., Heil S.** Sequential Neural Text Compression // IEEE Trans. Neural Networks. – 1996. – Vol. 7, N. – P. 142–146.
57. **Sedgewick R.** Algorithms. – Addison-Wesley Reading, Massachusetts, USA, 1988.
58. **Schindler M.** A Fast Block-Sorting Algorithm for Lossless Data Compression. – http://www.compressconsult.com/st/dcc97eab.ps.gz.
59. **Schindler M.** Range Coder. – http://www.compressconsult.com/rangecoder/.
60. **Storer J. A., Szymanski T. G.** Data Compression via Textual Substitution // J. ACM. – 1982. – Vol. 29, N 4. – P. 928–951.
61. **Teahan W. J.** Probability Estimation for PPM. – http://www.cs.waikato.ac.nz/~wjt/papers/NZCSRSC.ps.gz.
62. **Thomas S. W., McKie J., Davies S., Turkowski K., Woods J. A., Orost J. W.** Compress (Version 4.0): Program and Documentation. – 1985.
63. **Tischer P.** A modified Lempel-Ziv-Welch Data Compression Scheme // Aust. Comput. Sci. Commun. – 1987. – Vol. 9, N 1. – P. 262–272.
64. **Tjalkens Tj. J., Volf P. A. J., Willems F. M. J.** A Context-Tree Weighting Method for Text-Generating Sources // Proc. IEEE Data Compression Conf. – Snowbird, Utah, USA, Mar. 25–27, 1997. – P. 472. (см. также http://ei1.ei.ele.tue.nl/~paul/p_and_w/dcc97.html)
65. **Volf P. A. J.** Text Compression Methods Based on Context Weighting: Tech. rept. – Stan Ackermans Institute, Eindhoven University of Technology, Netherlands, June, 1996.
66. **Volf P. A. J., Willems F. M. J.** Switching between Two Universal Source Coding Algorithms // Proc. IEEE Data Compression Conf. – Snowbird, Utah, USA, Mar. 30 – Apr. 1, 1998. – P. 491–500.
67. **Volf P. A. J., Willems F. M. J.** Switching Method: Elaborations // Proc. 19-th Symp. Inform. Theory in the Benelux. – Veldhoven, Netherlands, May 28–29, 1998. – P. 13–20.

68. **Welch T. A.** A technique for High-Performance Data Compression // IEEE Computer. – 1984. – Vol. 17, N 6. – P. 8–19.
69. **Willems F. M. J.** The Context-Tree Weighting Method: Extensions // IEEE Trans. Inform. Theory. – 1998. – Vol. 44, N 2. – P. 792–798.
70. **Willems F. M. J., Shtarkov Y. M., Tjalkens Tj. J.** Context Tree Weighting: A Sequential Universal Source Coding Procedure for FSMX Sources // Proc. IEEE Int. Symp. Inform. Theory. – San Antonio, Texas, USA, 1993. – P. 59.
71. **Willems F. M. J., Shtarkov Y. M., Tjalkens Tj. J.** Context Tree Weighting: Basic Properties // IEEE Trans. Inform. Theory. – 1995. – Vol. 41, N 3. – P. 653–664.
72. **Willems F. M. J., Shtarkov Y. M., Tjalkens Tj. J.** Context Weighting for General Finite Context Sources // IEEE Trans. Inform. Theory. – 1996. – Vol. 42, N 5. – P. 1514–1520.
73. **Witten J. S., Bell T. C.** The Zero Frequency Problem: Estimating the Probabilities of Novel Events in Adaptive Text Compression // IEEE Trans. Inform. Theory. – 1987. – Vol. 37, N 6. – P. 1085–1094.
74. **Witten I. H., Neal R. M., Cleary J. G.** Arithmetic Coding for Data Compression // CACM. – 1987. – Vol. 30, N 6. – P. 520–540.
75. **Ziv J., Lempel A.** A Universal Algorithms for Sequential Data Compression // IEEE Trans. Inform. Theory. – 1977. – Vol 23, N 3. – P. 337–343.
76. **Ziv J., Lempel A.** Compression of Individual Sequences via Variable-Rate Coding // IEEE Trans. Inform. Theory. – 1978. – Vol. 24, N 5. – P. 530–536.
77. **CCITT** Data Compression Procedures for Data Circuit Terminating Equipment (DCE) Using Error Correction Procedures: Recommendation V.42bis. – Jan., 1990.
78. Calgary corpus. – ftp://ftp.cpsc.ucalgary.ca/pub/projects/text.compression.corpus/.
79. Canterbury corpus. – http://corpus.canterbury.ac.nz/.