Учреждение образования
«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

П. П. Урбанович, Д. В. Шиман, Н. П. Шутько

ЛАБОРАТОРНЫЙ ПРАКТИКУМ ПО ДИСЦИПЛИНАМ «ЗАЩИТА ИНФОРМАЦИИ И НАДЕЖНОСТЬ ИНФОРМАЦИОННЫХ СИСТЕМ» И «КРИПТОГРАФИЧЕСКИЕ МЕТОДЫ ЗАЩИТЫ ИНФОРМАЦИИ»

В 2-х частях
Часть 1. Кодирование информации

Рекомендовано
учебно-методическим объединением по образованию
в области информатики и радиоэлектроники
в качестве учебно-методического пособия для студентов
учреждений высшего образования по специальностям
1-40 01 01 «Программное обеспечение
информационных технологий»,
1-40 05 01 «Информационные системы и технологии
(по направлениям)» направления специальности
1-40 05 01-03 «Информационные системы и технологии
(издательско-полиграфический комплекс)»,
1-98 01 03 «Программное обеспечение информационной
безопасности мобильных систем»

Минск 2019

УДК [004.056+003.26](075.8)
ББК 32.972.5я73
У69

Р е ц е н з е н т ы:
кафедра управления информационными ресурсами
Академии управления при Президенте Республики Беларусь
(и. о. заведующего кафедрой кандидат физико-математических наук,
доцент *В. К. Шешолко*, рецензент: доцент кафедры
кандидат технических наук, доцент *Н. И. Белодед*);
заведующий кафедрой информационных радиотехнологий
учреждения образования «Белорусский государственный университет
информатики и радиоэлектроники» доктор технических наук,
профессор *Н. И. Листопад*

Все права на данное издание защищены. Воспроизведение всей книги или ее части не может быть осуществлено без разрешения учреждения образования «Белорусский государственный технологический университет».

Урбанович, П. П.

У69 Лабораторный практикум по дисциплинам «Защита информации и надежность информационных систем» и «Криптографические методы защиты информации». В 2 ч. Ч. 1. Кодирование информации : учеб.-метод. пособие для студентов учреждений высшего образования по специальностям 1-40 01 01 «Программное обеспечение информационных технологий», 1-40 05 01 «Информационные системы и технологии (по направлениям)» направления специальности 1-40 05 01-03 «Информационные системы и технологии (издательско-полиграфический комплекс)», 1-98 01 03 «Программное обеспечение информационной безопасности мобильных систем» / П. П. Урбанович, Д. В. Шиман, Н. П. Шутько. – Минск: БГТУ, 2019. – 116 с.

ISBN 978-985-530-764-9.

Издание содержит материалы для подготовки и выполнения лабораторных работ в соответствии с учебными планами дисциплин «Защита информации и надежность информационных систем» и «Криптографические методы защиты информации». Первая часть пособия включает работы, относящиеся к основам теории информации, теории помехоустойчивого кодирования и сжатия информации. Конкретное задание по каждой работе дополнено необходимым минимумом теоретических сведений по объекту исследования, а также вопросами и заданиями для контроля и самоконтроля знаний.

Пособие предназначено для студентов ИТ-специальностей. Может быть полезно магистрантам, изучающим дисциплину «Основы информационных технологий».

УДК [004.056+003.26](075.8)
ББК 32.972.5я73

ISBN 978-985-530-764-9 (Ч. 1)
ISBN 978-985-530-763-2

© УО «Белорусский государственный технологический университет», 2019
© Урбанович П. П., Шиман Д. В., Шутько Н. П., 2019

ПРЕДИСЛОВИЕ

Под защитой информации в широком смысле понимается комплекс мер и мероприятий по предотвращению угроз информационной безопасности и устранению их последствий. Важность и возрастающая актуальность задач, относящихся к обеспечению информационной безопасности различных систем, диктует необходимость постоянного совершенствования и развития методологической и методической базы учебного процесса, получения студентами практических навыков использования современных методов и средств решения указанных задач.

В рамках существующих стандартов образования и учебных планов ряда специальностей (в том числе 1-40 01 01 «Программное обеспечение информационных технологий», 1-40 05 01 «Информационные системы и технологии (по направлениям)» направление специальности 1-40 05 01-03 «Информационные системы и технологии (издательско-полиграфический комплекс)», 1-98 01 03 «Программное обеспечение информационной безопасности мобильных систем» предусматривается изучение студентами комплекса специальных дисциплин, направленных на овладение теоретическими знаниями и практическими навыками в анализируемой предметной области. К числу таких дисциплин относятся «Защита информации и надежность информационных систем» (направление специальности 1-40 05 01-03), «Криптографические методы защиты информации» (специальности 1-98 01 03, 1-40 01 01).

Настоящее учебно-методическое пособие охватывает часть тем, предусмотренных учебными программами перечисленных учебных дисциплин. Основу пособия составляют теоретические сведения и практические задания, относящиеся к кодированию данных в информационных системах – одному из основных классов преобразования информации. Здесь рассматриваются вопросы использования избыточного кодирования и сжатия передаваемых сообщений.

Материал пособия включает 11 лабораторных работ. По каждой выполненной работе студент оформляет отчет по установленной форме. Для защиты работы преподавателю представляется электронный вариант отчета. При оформлении отчета полезно ознакомиться с содержанием второй и третьей глав учебно-методического пособия [1].

Приведенный список использованной литературы содержит стандартное библиографическое описание источников на бумажных носителях, а также указатели на соответствующие электронные адреса этих источников, если таковые имеются. По нашему мнению, это поможет заинтересованному читателю углубить теоретические знания и практический опыт при выполнении работ.

Помимо студентов пособие может быть полезным магистрантам, изучающим дисциплину «Основы информационных технологий».

Авторы выражают благодарность рецензентам, чьи замечания и доброжелательная критика способствовали улучшению пособия.

Лабораторная работа № 1

РАЗРАБОТКА И ВНЕДРЕНИЕ ПОЛИТИКИ БЕЗОПАСНОСТИ ОРГАНИЗАЦИИ ИЛИ УЧРЕЖДЕНИЯ

Цель: приобретение практических навыков разработки и внедрения эффективной политики информационной безопасности организации или учреждения.

Задачи:

1. Научиться выделять и классифицировать особенности информационной или информационно-вычислительной системы (ИВС) конкретной организации или учреждения как объекта защиты.

2. Овладеть навыками принятия обоснованных решений по организационному и правовому регулированию проблем, относящихся к состоянию безопасности ИВС, обеспечению необходимого уровня защиты информации в ИВС.

3. Овладеть основными приемами анализа угроз информационной безопасности ИВС.

4. Научиться выявлять все возможные угрозы и их источники информационной безопасности в организации или учреждении, анализировать и оценивать собранные данные.

5. Разработать концепцию, основные элементы политики безопасности для организации или учреждения по указанному преподавателем варианту задания.

6. Разработать мероприятия по внедрению предложенной Вами политики безопасности.

7. Результаты выполнения лабораторной работы оформить в виде описания разработанной политики безопасности, а также плана мероприятий по ее реализации.

1.1. Теоретические сведения

1.1.1. Основные понятия из предметной области

Политика информационной безопасности (***ПИБ***) организации или учреждения – совокупность правил, процедур, практических методов, руководящих принципов, *документированных управленческих решений*, направленных на защиту информации и связан-

ных с ней ресурсов и используемых всеми сотрудниками организации или учреждения в своей деятельности.

Информационная (информационно-вычислительная) система – организационно упорядоченная совокупность документов, технических средств и информационных технологий, реализующая информационные (информационно-вычислительные) процессы.

Информационные процессы – процессы сбора, накопления, хранения, обработки (переработки), передачи и использования информации.

Информационные ресурсы – отдельные документы или массивы документов в информационных системах.

Объект – пассивный компонент системы, хранящий, перерабатывающий, передающий или принимающий информацию; примеры объектов: страницы, файлы, папки, директории, компьютерные программы, устройства (мониторы, диски, принтеры и т. д.).

Субъект – активный компонент системы, который может инициировать поток информации; примеры субъектов: пользователь, процесс либо устройство.

Доступ – специальный тип взаимодействия между объектом и субъектом, в результате которого создается поток информации от одного к другому.

Атака – попытка несанкционированного преодоления защиты системы.

Несанкционированный доступ (НСД) – доступ к информации, устройствам ее хранения и обработки, а также к каналам передачи, реализуемый без ведома (санкции) владельца и нарушающий тем самым установленные правила доступа.

Защита информации – организационные, правовые, программно-технические и иные меры по предотвращению угроз информационной безопасности и устранению их последствий.

Безопасность информации – защищенность информации от нежелательного (для соответствующих субъектов информационных отношений) ее разглашения (нарушения конфиденциальности), искажения (нарушения целостности), утраты или снижения степени доступности информации, а также незаконного ее тиражирования.

Безопасность любого ресурса информационной системы складывается из обеспечения трех его характеристик: конфиденциальности, целостности и доступности, также могут быть включены другие, такие как аутентичность, подотчетность, надеж-

ность; или иначе: ***информационная безопасность*** – все аспекты, связанные с определением, достижением и поддержанием конфиденциальности, целостности, доступности информации или средств ее обработки:

конфиденциальность (англ. *confidentiality*) компонента системы заключается в том, что он доступен только тем субъектам доступа (пользователям, программам, процессам), которым предоставлены на то соответствующие полномочия;

целостность (англ. *integrity*) компонента предполагает, что он может быть модифицирован только субъектом, имеющим для этого соответствующие права; целостность является гарантией корректности (неизменности, работоспособности) компонента в любой момент времени;

доступность (англ. *availability*) компонента означает, что имеющий соответствующие полномочия субъект может в любое время без особых проблем получить доступ к необходимому компоненту системы (ресурсу).

1.1.2. Элементы эффективной системы информационной безопасности

Для создания эффективной системы информационной безопасности организации или учреждения целесообразно разработать:

- *концепцию* информационной безопасности, которая определяет в целом цели политики и основные ее принципы в увязке со статусом, целями и задачами организации или учреждения;
- *стандарты* (менеджмента качества) – правила и принципы защиты информации по каждому конкретному направлению деятельности;
- *процедуры* – описание конкретных действий по защите информации при работе с ней: персональных данных, порядка доступа к информационным носителям, системам и ресурсам;
- *инструкции*, содержащие подробное описание (алгоритмы) действий по организации информационной защиты и обеспечению разработанных стандартов и процедур;
- *план мероприятий* по обучению персонала и тестированию знаний сотрудников, имеющих доступ к информационным ресурсам.

Все вышеуказанные элементы должны быть взаимосвязанными и непротиворечивыми.

Для эффективной организации системы информационной безопасности целесообразно разработать *аварийные планы*. Они необходимы на случай восстановления информационных систем при возникновении форс-мажорных обстоятельств: аварий, катастроф и т. д.

1.1.3. Концепция политики безопасности для организации или учреждения

Концепция политики информационной безопасности (ИБ) разрабатывается в соответствии с законодательством по информационной безопасности Республики Беларусь, соответствующими нормативными документами министерства или ведомства, к которому относится организация или учреждение, а также решениями Оперативно-аналитического центра при Президенте Республики Беларусь (см. п. 2.2 в книге [1]).

Обеспечение ИБ на предприятиях и в учреждениях, как правило, является неотъемлемой частью общей системы управления, необходимой для достижения уставных целей и задач. Значимость систематической целенаправленной деятельности по обеспечению ИБ становится тем более высокой, чем выше степень автоматизации бизнес-процессов. Значимость обеспечения ИБ в некоторых случаях может определяться наличием в общей системе информационных потоков предприятия сведений, составляющих не только коммерческую, но и государственную тайну, а также другие виды конфиденциальной информации: сведения, составляющие банковскую тайну, различные виды *персональных данных*, в том числе врачебная тайна, интеллектуальная собственность компаний-партнеров и т. п. Обеспечение ИБ в этой сфере и, в частности, основные требования, организационные правила и процедуры непосредственно регламентируются указанными в начале данного подраздела документами.

Мероприятия по разработке и внедрению политики информационной безопасности в соответствии со стандартом BS ISO/IEC 27001:2005, на основе которого разработан национальный стандарт России ГОСТ Р ИСО/МЭК 27001–2006 [2], должны начинаться с определения области действия *системы управления информационной безопасностью* (СУИБ). Определение области действия СУИБ полностью зависит от организации. Областью действия СУИБ может являться вся организация в целом либо конкретный бизнес-процесс или информационная система.

Решение относительно области действия СУИБ должно учитывать интерфейсы и взаимозависимости этой СУИБ с другими частями организации (находящимися вне области действия СУИБ), другими организациями, поставщиками третьей стороны или любыми другими субъектами, не входящими в СУИБ. Примером является СУИБ, состоящая только из одного конкретного бизнес-процесса. В этом случае другие части организации, которые необходимы СУИБ для повседневного функционирования (например, кадровые ресурсы, финансы, продажи и маркетинг или коммунальные службы), являются интерфейсами и зависимостями, в дополнение к любым другим интерфейсам и зависимостям, которые могут существовать.

Область действия СУИБ должна быть подходящей и соответствовать как возможностям организации, так и ее ответственности за обеспечение информационной безопасности в соответствии с требованиями, определяемыми *оценкой рисков* и применимыми законодательными и нормативными механизмами контроля. Для того чтобы заявить об этом соответствии, из области действия СУИБ не должно быть исключено ничего, что оказывает влияние на способность и/или ответственность организации за обеспечение информационной безопасности в соответствии с требованиями, определяемыми оценкой рисков, и соответствующими нормативными требованиями.

Основными разделами *концепции информационной безопасности* могут быть следующие:

- определение ИБ (или СУИБ);
- структура информационной системы организации (учреждения) и вытекающая из этого *структура системы обеспечения информационной безопасности*;
- безопасность информации: принципы и стандарты;
- оценка рисков информационным ресурсам в организации (учреждении);
- описание основных механизмов контроля безопасности;
- обязанности и ответственность каждого отдела, управления или департамента, каждого сотрудника в реализации разработанной и утвержденной политики безопасности;
- обязанности лица (администратора безопасности), ответственного за организацию оперативного контроля и управления политикой безопасности;

• ссылки на документы об информационной безопасности, действующие на территории Республики Беларусь.

Помимо упомянутых выше законодательных и нормативных актов, в общем плане структура системы обеспечения ИБ должна базироваться на *организационно-технических и режимных мерах и методах*. Для построения политики ИБ рассматривают следующие направления защиты ИВС:

• защита объектов ИВС;

• защита процессов, процедур и программ обработки информации;

• защита каналов связи;

• подавление побочных электромагнитных излучений;

• управление системой защиты.

Организационная защита обеспечивает:

• организацию охраны, режима, работу с кадрами и с документами;

• использование технических средств безопасности (например, простейших дверных замков, магнитных или иных карт и др.), информационно-аналитическую деятельность по выявлению внутренних и внешних угроз.

Оперативно-аналитический центр при Президенте Республики Беларусь требует, например, от государственных организаций и учреждений выполнения следующих «рекомендаций по обеспечению безопасности информации в локальных сетях, подключенных к сети Интернет» [3]:

• осуществлять предоставление доступа сотрудникам органа (организации) к сервисам сети Интернет (электронная почта, передача файлов, информационные ресурсы и др.) в соответствии с определенным в государственном органе порядком;

• определять правила работы сотрудников с сервисами сети Интернет (электронная почта, передача файлов, доступ к информационным ресурсам, IP-телефонии, социальным сетям и публичным системам мгновенных сообщений);

• определять администраторов сети, их права и обязанности;

• определять права и обязанности пользователей;

• определять ответственность сотрудников и должностных лиц за обеспечение защиты информации;

• обеспечивать контроль использования сотрудниками в глобальных сетях: IP-телефонии, социальных сетей и публичных систем мгновенных сообщений;

• определять порядок и перечень используемого программного обеспечения на средствах вычислительной техники сотрудников;

• определять порядок применения средств защиты информации, установленных в локальной вычислительной сети;

• определять необходимые мероприятия по разграничению доступа к средствам защиты информации и обработки информации;

• определять регламент смены атрибутов безопасности (паролей) пользователей;

• определять порядок действий при возникновении нештатной ситуации (сбои, повреждения и отказы) с информационными ресурсами;

• определять регламенты резервирования и уничтожения информации;

• определять порядок контроля, учета использования ресурсов сети Интернет пользователями, формирования и предоставления руководству организации отчетных документов.

Перечисленные требования можно рассматривать как элементы (процедуры и инструкции) *эффективной системы ИБ*.

При этом использование технических, программно-аппаратных и программных средств должно обеспечивать:

• межсетевое экранирование с использованием собственных возможностей и (или) возможностей уполномоченных поставщиков Интернет-услуг;

• идентификацию абонентских устройств в локальной сети;

• блокирование неконтролируемого обмена информацией между рабочими местами пользователей в локальной сети;

• исключение использования на рабочих местах в локальной сети постороннего программного обеспечения, ресурсов сети Интернет, предназначенных для сокрытия действий пользователя;

• невозможность подключения рабочего места в локальной сети к сетям связи общего пользования через другие каналы доступа (сотовый телефон, модем);

• синхронизацию системного времени от единого (общего) источника (в качестве источника использовать службу единого времени Белорусского государственного института метрологии);

• осуществление сбора и хранения данных авторизации и статистики использования сети Интернет пользователями в течение 1 года;

• возможность анализа использования сети Интернет пользователями (с использованием собственных возможностей или поставщиков Интернет-услуг);

• применение криптографических протоколов для защиты данных авторизации при работе с сервисами сети Интернет.

Британский стандарт BS 7799-3:2006 «Руководство по управлению рисками информационной безопасности» (специалисты часто ссылаются на него при изучении и анализе вопросов разработки политики безопасности; его перевод можно найти в источнике [4]) рекомендует в основу концепции политики ИБ положить:

• идентификацию (описание) ресурсов;

• идентификацию требований законодательства и бизнеса, применимых к идентифицированным ресурсам;

• оценивание идентифицированных ресурсов с учетом идентифицированных требований законодательства и бизнеса, а также последствий нарушения конфиденциальности, целостности и доступности.

1.1.4. Оценка рисков для информационных ресурсов

Общая характеристика факторов, влияющих на безопасность ИВС. *Фактор*, *воздействующий на ИВС*, – это явление, действие или процесс, результатом которых может быть утечка, искажение, уничтожение данных, блокировка доступа к ним, повреждение или уничтожение системы защиты.

Все многообразие дестабилизирующих факторов можно разделить на два класса: внутренние и внешние.

Внутренние дестабилизирующие факторы влияют:

1) на программные средства (ПС):

• некорректный исходный алгоритм;

• неправильно запрограммированный исходный алгоритм (первичные ошибки);

2) на аппаратные средства (АС):

• системные ошибки при постановке задачи проектирования;

• отклонения от технологии изготовления комплектующих изделий и АС в целом;

• нарушение режима эксплуатации, вызванное внутренним состоянием АС.

Внешние дестабилизирующие факторы влияют:

1) на программные средства:

- неквалифицированные пользователи;
- несанкционированный доступ к ПС с целью модификации кода;

2) на аппаратные средства:

- внешние климатические условия;
- электромагнитные и ионизирующие помехи;
- перебои в электроснабжении;
- недостаточная квалификация обслуживающего персонала.

Риски и их оценка. В соответствии с [2, 4] оценка рисков включает в себя следующие действия и мероприятия:

- идентификация значимых угроз и уязвимостей для идентифицированных ресурсов;
- оценка вероятности возникновения угроз и уязвимостей;
- вычисление рисков; оценивание рисков по заранее определенной шкале риска.

Все многообразие потенциальных угроз безопасности информации по природе их возникновения разделяются на два класса: *естественные* (объективные) и *искусственные* (субъективные).

Естественные угрозы – это угрозы, вызванные воздействиями на информационную систему и ее компоненты объективных физических процессов техногенного характера или стихийных природных явлений, независящих от человека.

Искусственные угрозы – это угрозы, вызванные деятельностью человека.

Источники угроз по отношению к самой информационной системе могут быть как *внешними*, так и *внутренними* (о чем мы вспоминали выше).

Основные источники угроз безопасности информации можно классифицировать следующим образом:

- *непреднамеренные* (ошибочные, случайные, без злого умысла и корыстных целей) нарушения установленных регламентов сбора, обработки и передачи информации, а также требований безопасности информации и другие действия пользователей ИВС (в том числе сотрудников, отвечающих за обслуживание и администрирование компонентов корпоративной информационной системы), приводящие к непроизводительным затратам времени и ресурсов, разглашению сведений ограниченного распространения,

потере ценной информации или нарушению работоспособности компонентов ИВС;

• *преднамеренные* (в корыстных целях, по принуждению третьими лицами, со злым умыслом и т. п.) действия легально допущенных к информационным ресурсам пользователей (в том числе сотрудников, отвечающих за обслуживание и администрирование компонентов корпоративной информационной системы), которые приводят к непроизводительным затратам времени и ресурсов, разглашению сведений ограниченного распространения, потере ценной информации или нарушению работоспособности компонентов информационной системы:

– *деятельность преступных групп и формирований*, политических и экономических структур, разведок иностранных государств, а также отдельных лиц по добыванию информации, навязыванию ложной информации, нарушению работоспособности ИВС в целом и ее отдельных компонентов;

– *удаленное несанкционированное вмешательство посторонних лиц* из территориально удаленных сегментов корпоративной информационной системы и внешних информационно-телекоммуникационных сетей общего пользования (прежде всего сеть Интернет) через легальные и несанкционированные каналы подключения к таким сетям, используя недостатки протоколов обмена, средств защиты и разграничения удаленного доступа к ресурсам;

• *ошибки, допущенные при разработке* компонентов информационной системы и системы ее защиты, ошибки в программном обеспечении, отказы и сбои технических средств (в том числе средств защиты информации и контроля эффективности защиты).

В заключение целесообразно отдельно отметить «человеческий фактор», классифицировав физических лиц, которые могут получить (а часто и реализуют) несанкционированный доступ к информации. К ним следует отнести:

1) сотрудников организации (учреждения):

• программисты, системные администраторы и даже администраторы информационной безопасности;

• технический персонал;

2) лиц, не являющихся сотрудниками:

• посетители офиса;

• ранее уволенные сотрудники (особенно «обиженные» увольнением);

• хакеры.

С учетом изложенного *основные факторы* (*угрозы*) ресурсам можно идентифицировать следующим образом:

1) действия внутреннего или внешнего злоумышленника (несанкционированный, в том числе удаленный доступ с целью нарушения работоспособности ИВС, кражи, удаления или модификации информации, несанкционированного распространения материальных носителей за пределами организации);

2) наблюдение за источниками информации;

3) подслушивание конфиденциальных разговоров и акустических сигналов работающих механизмов;

4) перехват электрических, магнитных и электромагнитных полей, электрических сигналов и радиоактивных излучений;

5) разглашение информации компетентными людьми;

6) утеря носителей информации;

7) несанкционированное распространение информации через поля и электрические сигналы, случайно возникшие в аппаратуре;

8) воздействие стихийных сил (наводнения, пожары и т. п.);

9) сбои и отказы в аппаратуре сбора, обработки и передачи информации;

10) отказы системы электроснабжения;

11) воздействие мощных электромагнитных и электрических помех (промышленных и природных).

Несанкционированный доступ с помощью *деструктивных программных средств* осуществляется, как правило, через компьютерные сети.

Цель оценивания рисков состоит в определении характеристик рисков для информационной системы и ее ресурсов. На основе таких данных могут быть выбраны необходимые средства управления ИБ.

При оценивании рисков учитываются:

- ценность ресурсов;
- оценка значимости угроз;
- эффективность существующих и планируемых средств защиты.

Показатели ресурсов или потенциальное негативное воздействие на деятельность организации можно определять несколькими способами:

- количественными (например, стоимостными);
- качественными (могут быть построены на использовании таких понятий, как «умеренный» или «чрезвычайно опасный»);
- их комбинацией.

Рассмотрим пример создания шкалы для численной оценки рисков от несанкционированного доступа (НСД) к информационным ресурсам банка [1] (табл. 1.1).

Таблица 1.1

Условная численная шкала для оценки ущерба банку от НСД

Величина ущерба	Описание
0	Раскрытие информации принесет ничтожный моральный и финансовый ущерб банку
1	Ущерб от атаки есть, но он незначителен, основные финансовые операции и положение банка на рынке не затронуты
2	Финансовые операции не ведутся в течение некоторого времени, за это время банк терпит убытки, но его положение на рынке и количество клиентов изменяются минимально
3	Значительные потери на рынке и в прибыли. От банка уходит ощутимая часть клиентов
4	Потери очень значительны, банк на период до года теряет положение на рынке. Для восстановления положения требуются крупные финансовые займы
5	Банк прекращает существование

Можно конкретизировать определение вероятности наступления угрозы ресурсу. Вероятность того, что угроза реализуется, определяется на основе следующих факторов:

- привлекательность ресурса как показатель при рассмотрении угрозы от умышленного воздействия со стороны человека;
- возможность использования ресурса для получения дохода как показатель при рассмотрении угрозы от умышленного воздействия со стороны человека;
- технические возможности угрозы, используемые при умышленном воздействии со стороны человека;
- вероятность того, что угроза реализуется;
- степень легкости, с которой уязвимость может быть использована.

Вопрос о том, как провести границу между допустимыми и недопустимыми рисками, решается пользователем. Очевидно, что разработка политики безопасности требует учета специфики конкретных организаций.

Пример создания шкалы вероятности того, что угроза будет реализована, приведен в табл. 1.2.

Таблица 1.2

Вероятностно-временная шкала реализации несанкционированного доступа к информационным ресурсам

Вероятность события	Средняя частота события (НСД)
0	Данный вид атаки отсутствует
0,1	Реже, чем раз в год
0,2	Около 1 раза в год
0,3	Около 1 раза в месяц
0,4	Около 1 раза в неделю
0,5	Практически ежедневно

Далее можно создать таблицу рисков (табл. 1.3). На этапе анализа таблицы риски задаются некоторым максимально допустимым уровнем (порогом), например значением 0,5.

Далее проверяется каждая строка таблицы: превышен или не превышен порог для значения риска, связанного с анализируемой атакой? Если такое превышение имеет место, данная атака должна рассматриваться с точки зрения одной из первоочередных целей разработки политики безопасности (табл. 1.3).

Таблица 1.3

Оценка рисков

Описание атаки	Ущерб	Вероятность	Риск (Ущерб × Вероятность)
Спам (переполнение почтового ящика)	1	0,4	0,4
Копирование жесткого диска из центрального офиса	3	0,1	0,3
...	...	...	...
Итого			

Если интегральный риск (итого) превышает допустимый уровень, значит, в системе безопасности набирается множество мелких проблем, которые также нужно решать комплексно. В этом случае из строк таблицы (типов атак) выбираются те, которые «дают» самый весомый вклад в значение интегрального риска. Производится работа по снижению их влияния или полному устранению.

1.1.5. Мероприятия по внедрению политики безопасности

После того как документация по информационной безопасности готова, необходима плановая деятельность по ее внедрению в повседневную работу. Основу таких мероприятий, как было указано в плане выполнения лабораторной работы, составляют *инструкции*, содержащие подробное описание (алгоритмы) действий по организации информационной защиты и обеспечению разработанных стандартов и процедур, и *план мероприятий* по обучению персонала и тестированию знаний сотрудников, имеющих доступ к информационным ресурсам.

Можно выделить следующие общие направления мероприятий:

- управление персоналом;
- физическая защита инфраструктуры ИВС;
- поддержание работоспособности ИВС;
- реагирование на нарушения режима безопасности ИВС;
- планирование восстановительных работ.

Управление персоналом заключается в выполнении следующих условий. Во-первых, для каждой должности должны существовать квалификационные требования по ИБ. Во-вторых, в должностные инструкции должны входить разделы, касающиеся информационной безопасности. В-третьих, каждого работника нужно научить мерам безопасности теоретически и на практике.

Меры физической защиты включают в себя защиту от утечки информации по техническим каналам, инженерные способы защиты и т. д.

Планирование восстановительных работ предполагает:

- слаженность действий персонала во время и после аварии;
- наличие заранее подготовленных резервных производственных площадок;
- официально утвержденную схему переноса на резервные площадки основных информационных ресурсов;
- схему возвращения к нормальному режиму работы.

Поддержание работоспособности включает в себя создание инфраструктуры, включающий в себя как технические, так и процедурные регуляторы и способной обеспечить любой наперед заданный уровень работоспособности на всем протяжении жизненного цикла информационной системы.

Реагирование на нарушение режима безопасности может быть регламентировано в рамках отдельно взятой организации. В настоящее время осуществляется только мониторинг компьютерных преступлений в национальном масштабе и на мировом уровне.

Основой программно-технического уровня являются следующие механизмы безопасности:

- идентификация и аутентификация пользователей;
- управление доступом;
- протоколирование и аудит;
- криптография;
- экранирование;
- обеспечение высокой доступности и т. д.

Таким образом, политика информационной безопасности должна рассматриваться как *система*, как комплекс инструментов по защите информации.

1.2. Практическое задание

Разработать политику информационной безопасности организации согласно варианту, представленному в табл. 1.4, а также план мероприятий по ее реализации.

Отчет по лабораторной работе оформить в соответствии с СТП БГТУ 001-2010.

Придерживайтесь следующей структуры отчета.

Титульный лист.

1. *Обоснование* актуальности, цели и задачи разработки ПИБ в организации (учреждении).

2. *Объекты защиты*. Описание структуры организации (учреждения), периметра и внутренней структуры ИВС. Полный обзор всех возможных объектов, а также субъектов информационных отношений, для защиты которых должны быть приняты меры по обеспечению информационной безопасности.

3. *Основные угрозы и их источники.* Анализ потенциальных угроз: естественных и искусственных, а также преднамеренных и непреднамеренных, внешних и внутренних.

4. *Оценка угроз, рисков и уязвимостей.* Анализ ценности ресурсов, оценка значимости угроз, а также эффективности сущест-

вующих и планируемых средств защиты (воспользуйтесь приведенными в описании таблицами, заполните их).

5. *Меры, методы и средства обеспечения требуемого уровня защищенности информационных ресурсов.* Описание разработанной политики ИБ и программы обеспечения безопасности на всех уровнях работы организации (учреждения).

Выводы и предложения.

Таблица 1.4

Варианты для разработки политики безопасности

Вариант	Тип организации или учреждения
1	Учебная организация. Школа
2	Event-компании
3	Консалтинговая компания
4	Поликлиника
5	Издательство
6	Юридическая компания
7	Страховая компания
8	Банк
9	Туристическая компания
10	Логистическая компания
11	Учебная организация. Университет
12	Интернет-магазин
13	Маркетинговая компания
14	Оператор мобильной связи
15	ИТ-компания
16	Больница
17	Библиотека

ВОПРОСЫ ДЛЯ КОНТРОЛЯ И САМОКОНТРОЛЯ

1. Охарактеризовать актуальность и основные причины проблемы информационной безопасности организации, страны.
2. Сформулировать цели и задачи политики информационной безопасности.
3. Охарактеризовать основные угрозы информационной безопасности каждой организации (учреждения) из табл. 1.4.
4. Как правильно проводить оценку рисков?
5. Что должна включать в себя программа внедрения политики информационной безопасности?

Лабораторная работа № 2

ЭЛЕМЕНТЫ ТЕОРИИ ИНФОРМАЦИИ. ПАРАМЕТРЫ И ХАРАКТЕРИСТИКИ ДИСКРЕТНЫХ ИНФОРМАЦИОННЫХ СИСТЕМ

Цель: приобретение практических навыков расчета и анализа параметров и информативных характеристик дискретных ИС.

Задачи:

1. Закрепить теоретические знания по основам теории информации.

2. Разработать приложение для расчета и анализа параметров и информативных характеристик дискретных ИС.

3. Результаты выполнения лабораторной работы оформить в виде описания разработанного приложения, методики выполнения экспериментов с использованием приложения и результатов эксперимента.

2.1. Теоретические сведения

2.1.1. Основные понятия из предметной области

Передача информации (данных) осуществляется между двумя абонентами, называемыми *источником сообщения* (ИсС) и *получателем сообщения* (ПС). Третьим элементом информационной системы является *канал* (среда) *передачи*, связывающий ИсС и ПС.

Отметим также, что и в системах с хранением информации всегда можно выделить ИсС и ПС. В данном случае каналом передачи здесь выступает устройство хранения информации (память). Например, при записи данных в ОЗУ (оперативное запоминающее устройство) компьютера в качестве ИсС и ПС может выступать процессор (соответственно при записи и чтении данных).

Таким образом, простейшая информационная система состоит из трех элементов: источника сообщения, канала передачи сообщения и получателя сообщения.

Отображение сообщения обеспечивается изменением какой-либо физической величины, характеризующей процесс (например, амплитуда, частота, фаза). Эта величина является *информационным параметром сигнала* (в общем случае – информационной системы).

Сигналы, как и сообщения, могут быть *непрерывными* и *дискретными*. Информационный параметр непрерывного сигнала с течением времени может принимать любые мгновенные значения в определенных пределах. Непрерывный сигнал часто называют *аналоговым,* а каналы и устройства, функционирующие на основе такого типа сигналов, – аналоговыми.

Дискретный сигнал (устройство или канал передачи) характеризуется конечным числом значений информационного параметра.

Дискретные сообщения состоят из последовательности *дискретных знаков*. Часто этот параметр принимает всего два значения (0 или 1). Сообщение или канал его передачи на основе этих двух значений сигнала называют *двоичным* или *бинарным*.

Построение сигнала по определенным правилам, обеспечивающим соответствие между сообщением и сигналом, называют ***кодированием***.

Кодирование в широком смысле – *преобразование сообщения в сигнал*.

Кодирование в узком смысле – *представление исходных знаков*, называемых символами, в другом алфавите с меньшим числом знаков. Оно осуществляется с целью повышения надежности и преобразования сигналов к виду, удобному для передачи по каналам связи. Последний тип кодирования относится к так называемой *прикладной теории кодирования информации*, занимающейся поиском и реализацией методов и средств обнаружения несоответствий (*ошибок*) между переданным X_k и принятым Y_k сообщениями.

Рассмотрим основные характеристики и параметры двоичных систем.

Важнейшая характеристика источника, получателя или канала – алфавит.

Алфавит, A – это общее число знаков или символов (N), используемых для генерации или передачи сообщений. Символы алфавита будем обозначать через $\{a_i\}$, где $1 \le i \le N$; N – *мощность алфавита*.

Минимальное число элементов алфавита $N_{\min} = 2$, $\boldsymbol{A} = \{0, 1\}$ – двоичный код. Один дискретный знак представляет собой *элементарное сообщение*, последовательность знаков – сообщение.

Набор элементов алфавита, создаваемых дискретным источником сообщений, заранее, априори (до опыта) известен получа-

телю. ИсС в каждый дискретный момент времени выдает один элемент алфавита. Этот элемент сообщения является одним из символов алфавита. Понятно, что ПС заранее не известно, какой это элемент. Если обозначить вероятность выбора каждого элемента алфавита $p(a_i)$, то

$$\sum_{i=1}^{N} p(a_i) = 1.$$

Вероятности $p(a_i)$ могут быть получены в результате анализа частотных свойств символов алфавита, если на входе такого анализатора принять документ на основе соответствующего алфавита. Причем объем документа должен быть таким, чтобы от частости (частоты) появления каждого символа в анализируемом документе можно было перейти к вероятности соответствующего события. Можно предположить, что указанному требованию будет соответствовать объем электронного документа *не менее нескольких десятков килобайт*.

2.1.2. Двоичный канал передачи информации

Двоичный канал передачи информации строится на основе двоичного алфавита: $\boldsymbol{A} = \{0, 1\}$. При этом канал, в котором вероятности искажения переданного 0 (принята соответственно 1; этому событию соответствует *условная вероятность* $p(1|0)$) и переданной 1 (принят соответственно 0; этому событию соответствует условная вероятность $p(0|1)$) равны, как и равны вероятности передачи 0 ($p(0)$) и 1 ($p(1)$), называют *двоичным симметричным каналом* (*ДСК*).

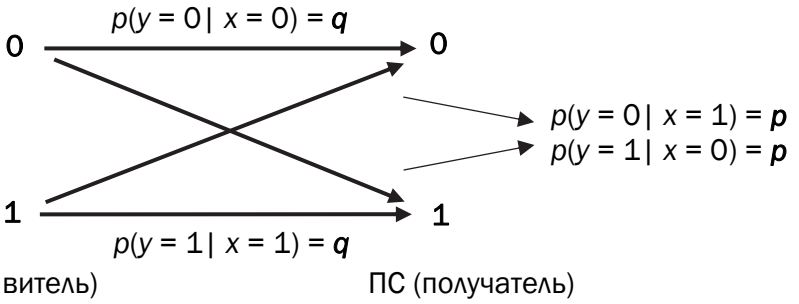
В общем случае, если передается сообщение $X_k = x_1, x_2, \ldots, x_k$, а принимается сообщение $Y_k = y_1, y_2, \ldots, y_k$, то данные условные вероятности можно рассматривать с двух точек зрения: $p(x_i|y_j)$ и $p(y_j|x_i)$.

На рисунке далее представлен ДСК.

На рисунке обозначены: q – вероятность правильной (безошибочной) передачи бита сообщения, p – вероятность передачи бита с ошибкой. Понятно, что $p + q = 1$.

Информационной характеристикой алфавита (источника сообщений на основе этого алфавита) является ***энтропия***.

Этот термин применительно к техническим системам был введен К. Шенноном и Р. Хартли.



Схематичное представление двоичного симметричного канала

Энтропию алфавита $\boldsymbol{A} = \{a_i\}$ по К. Шеннону рассчитывают по следующей формуле:

$$H_S(A) = -\sum_{i=1}^{N} P(a_i) \cdot \log_2 P(a_i). \qquad (2.1)$$

С физической точки зрения энтропия алфавита показывает, какое количество информации приходится в среднем на один символ алфавита.

Частным случаем энтропии Шеннона считается энтропия Хартли. Дополнительным условием при этом является то, что все вероятности одинаковы и постоянны для всех символов алфавита. С учетом этого формулу (2.1) можно преобразовать к виду:

$$H_{Ch}(A) = \log_2 N. \qquad (2.2)$$

Сообщение X_k, которое состоит из k символов, должно характеризоваться определенным *количеством информации* $I(X_k)$:

$$I(X_k) = H(A) \cdot k. \qquad (2.3)$$

Здесь $H(A)$ – энтропия алфавита с соответствующим распределением вероятностей $p(a_i)$.

Если принять, что $p(a_i = 1) = p(1)$ и $p(a_i = 0) = p(0)$, используя выражение (2.1), вычислим энтропию бинарного алфавита:

$$H(A_2) = -p(0) \cdot \log_2(p(0)) - p(1) \cdot \log_2(p(1)). \qquad (2.4)$$

К примеру, если сообщение X_k состоит только из единиц ($X_k = 11\ldots1$) и имеет длину k, то вероятность того, что произвольный символ равен единице, составляет единицу ($p(a_i = 1) = 1$), и другая вероятность $p(a_i = 0) = 0$ для $i = \overline{1, N}$. Фактически здесь имеет место использование *моноалфавита*: алфавита, состоящего из одного символа.

Учитывая, что сумма $p(1) + p(0) = 1$, и выражая одну вероятность через другую (например, $p(1) = 1 - p(0)$), можно теоретически доказать информативность бинарного алфавита, решив дифференциальное уравнение $[dH(A)/dp(1)] = 0$ (вспомним из курса математики, как найти экстремум функции; можно для этого воспользоваться пособием [5]).

Если вероятность ошибки в ДСК отлична от 0 ($p > 0$), переданное сообщение может содержать ошибки: $X_k \neq Y_k$. Количество информации в таком сообщении при его передаче по ДСК будет определяться не энтропией двоичного алфавита (в соответствии с выражением (2.3)), а эффективной энтропией $H_e(A)$ алфавита или пропускной способностью канала:

$$H_e(A) = 1 - H(Y \mid X), \tag{2.5}$$

где $H(Y \mid X)$ – условная энтропия:

$$H(Y \mid X) = -p \cdot \log_2 p - q \cdot \log_2 q. \tag{2.6}$$

2.2. Практическое задание

Создать приложение для расчета и анализа параметров и информативных характеристик дискретных ИС, с помощью которого:

а) рассчитать энтропию указанных преподавателем алфавитов: один – на латинице, другой – на кириллице (по формуле (2.1) перейти от частоты появления каждого символа алфавита к соответствующей вероятности); в качестве входного может быть принят произвольный электронный текстовый документ на основе соответствующего алфавита; частоты появления символов алфавитов оформить в виде гистограмм (можно воспользоваться приложением MS Excel);

б) для входных документов, представленных в бинарных кодах, определить энтропию бинарного алфавита;

в) используя значения энтропии алфавитов, полученных в пунктах (а) и (б), подсчитать количество информации в сообщении, состоящем из собственных фамилии, имени и отчества (на основе исходного алфавита – (а) и в кодах ASCII – (б)); объяснить полученный результат;

г) выполнить задание пункта (в) при условии, что вероятность ошибочной передачи единичного бита сообщения составляет: 0,1; 0,5; 1,0.

ВОПРОСЫ ДЛЯ КОНТРОЛЯ И САМОКОНТРОЛЯ

1. Что такое алфавит источника сообщения?
2. Что такое мощность алфавита источника сообщения?
3. Какова мощность алфавита белорусского языка?
4. Какова мощность алфавита русского языка?
5. Какова мощность алфавита «компьютерного» языка?
6. Что такое энтропия алфавита?
7. Что такое энтропия сообщения?
8. От чего зависит энтропия алфавита?
9. Записать формулу для вычисления энтропии.
10. Что нужно знать для вычисления энтропии алфавита?
11. Как рассчитываются энтропия Шеннона и энтропия Хартли? В чем принципиальное различие между этими характеристиками? Дайте толкование физического смысла энтропии.
12. Пояснить назначение знака «минус» в формулах (2.1) и (2.4).
13. Что такое избыточность алфавита и избыточность сообщений, сформированных в компьютерных системах? Принцип действия каких систем основан на существовании данной избыточности?
14. Расположить в порядке возрастания энтропии известные Вам алфавиты.
15. Вычислить энтропию алфавита белорусского (русского) языка.
16. Вычислить энтропию Шеннона бинарного алфавита, если вероятность появления в произвольном документе на основе этого алфавита одного из символов составляет 0,25, другого – 0,75; либо 0 и 1,0; либо 0,5 и 0,5.
17. Чему равна энтропия алфавита по Хартли, если мощность этого алфавита равна: а) 1 символ; б) 2 символа; в) 8 символов?

Лабораторная работа № 3

ЭЛЕМЕНТЫ ТЕОРИИ ИНФОРМАЦИИ. ИНФОРМАТИВНОСТЬ ДАННЫХ В РАЗЛИЧНЫХ КОДИРОВКАХ

Цель: приобретение практических навыков трансформации данных и сопоставление энтропийных свойств используемых при этом алфавитов.

Задачи:

1. Закрепить теоретические знания по взаимной конвертации данных, представленных в кодах ASCII и base64.
2. Разработать приложение для конвертации произвольного документа в формат base64 и обратно.
3. Исследовать энтропийные характеристики используемых в конвертерах алфавитов.
4. Изучить особенности практической реализации операции XOR над данными, представленными в разных форматах.
5. Результаты выполнения лабораторной работы оформить в виде описания разработанного приложения, методики выполнения экспериментов с использованием приложения и результатов эксперимента.

3.1. Теоретические сведения

Из энтропийных оценок (алфавитов и сообщений), полученных в ходе выполнения лабораторной работы № 2, мы выяснили, что энтропия зависит от статических характеристик самих алфавитов и сообщений (вспомним энтропию по Шеннону и по Хартли).

Энтропия максимальна при равномерном появлении букв на любом месте сообщения. Для характеристики источника сообщений с различным алфавитом представляет интерес сравнение фактической энтропии источника с максимально возможной. В этом смысле введено понятие *избыточности источника сообщений*, или *избыточности алфавита*.

Избыточностью алфавита называют уменьшение информационной нагрузки на один символ вследствие разной вероятности и взаимозависимости появления его символов в сообщениях.

В наиболее общем виде избыточность алфавита R можно оценить отношением энтропии по Хартли и по Шеннону; при этом первая рассчитывается по выражению (2.2), вторая – по формуле (2.1):

$$R = \left[(H_{Ch}(A) - H_S(A) / H_{Ch}(A) \right] \cdot 100\%.$$

При выполнении предыдущей работы мы убедились, что формально одно и то же сообщение, но представленное на основе алфавита русского (белорусского, английского или иного) языка – с одной стороны, и представленное в кодах ASCII – с другой, будут характеризоваться различным количеством содержащейся в них информации. Эта дополнительная избыточность обусловлена переносом сообщения из одной среды в другую или, иначе говоря, кодированием символов исходного алфавита.

Утверждение восьмибитных кодировок (ASCII) как стандарта принесло некоторые проблемы. К этому моменту уже существовала определенная инфраструктура, использующая семибитные кодировки. Известны проблемы с «обрезанием восьмого бита» в системе электронной почты. Утверждение восьмибитного символа дало 256 различных значений, что позволило уместить в одной кодовой таблице и общепринятые символы (цифры, знаки препинания, латиницу), и символы кириллицы.

Уже созданное к тому времени и работающее программное обеспечение зачастую было приспособлено для семибитных кодировок, что приводило, например, к тому, что почтовый сервер при передаче письма обнулял старшие биты в каждом байте сообщения. Одним из решений проблемы стала кодировка (а точнее – алгоритм) *base64*. В PGP алгоритм base64 используется для кодирования бинарных данных.

Кодирование base64 разработано для представления произвольных последовательностей октетов в форме, позволяющей использовать строчные и прописные буквы. Используется 65-символьное подмножество набора символов US-ASCII, обеспечивающее представление одним печатным символом 6 битов данных (дополнительный 65-й символ используется для обозначения функции специальной обработки).

Процесс кодирования представляет группу из 24 последовательных битов в форме строки из 4 символов. Обработка выполняется слева направо, а 24-битная исходная группа образуется конкатенацией трех 8-битных групп (байтов). Данные 24 бита после

этого трактуются как 4 сцепленных группы по 6 битов, каждая из которых транслируется в один символ алфавита base64.

Каждая 6-битная группа используется в качестве индекса массива из 64 печатных символов. Символы алфавита, соответствующие индексу, помещаются в выходную строку [6]. Кодирование base64 с безопасным алфавитом используется для представления URL и имен файлов.

В табл. 3.1 перечислен алфавит, используемый для base64-кодировки. Значения представлены в различных системах счисления: десятичной (10), двоичной (2), восьмеричной (8) и шестнадцатеричной (16 или hex).

Еще раз обратимся к процессу кодировки. Как было выше установлено, каждые 6 битов буфера, начиная с самых старших, используются как индексы строки «ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/», и ее символы, на которые указывают индексы, помещаются в выходную строку.

Если кодируются только один или два байта, в результате получаются только первые два или три символа строки, а выходная строка дополняется двумя или одним символами «=». Это предотвращает добавление дополнительных битов к восстановленным данным.

Процесс повторяется над оставшимися входными данными. Такая обработка выполняется в тех случаях, когда последняя группа входных данных содержит меньше 24 битов. Кодируемое значение всегда завершается полным квантом кодирования.

Если на входе доступно менее 24 битов, входная группа дополняется (справа) нулями до формирования целого числа 6-битных групп. Заполнение в конце данных осуществляется как раз с использованием символа «=». Поскольку входная информация base64 всегда включает целое число октетов, возможны лишь перечисленные ниже случаи:

- размер финального блока кодирования на входе кратен 24 битам, кодированный результат будет содержать целое число 4-символьных групп без заполнения символами «=»;
- размер финального блока кодирования на входе составляет 8 битов, выходной блок будет представлять 2 символа, дополненные последовательностью из двух символов заполнения «==»;
- размер финального блока кодирования на входе составляет 16 битов, выходной блок будет представлять 3 символа, дополненные символом заполнения «=».

Таблица 3.1

Схема соответствия трансформации «символ – значение» в base64

Символ	Значение				Символ	Значение				Символ	Значение				Символ	Значение			
	10	2	8	16		10	2	8	16		10	2	8	16		10	2	8	16
A	0	000000	00	00	Q	16	010000	20	10	g	32	100000	40	20	w	48	110000	60	30
B	1	000001	01	01	R	17	010001	21	11	h	33	100001	41	21	x	49	110001	61	31
C	2	000010	02	02	S	18	010010	22	12	i	34	100010	42	22	y	50	110010	62	32
D	3	000011	03	03	T	19	010011	23	13	j	35	100011	43	23	z	51	110011	63	33
E	4	000100	04	04	U	20	010100	24	14	k	36	100100	44	24	0	52	110100	64	34
F	5	000101	05	05	V	21	010101	25	15	l	37	100101	45	25	1	53	110101	65	35
G	6	000110	06	06	W	22	010110	26	16	m	38	100110	46	26	2	54	110110	66	36
H	7	000111	07	07	X	23	010111	27	17	n	39	100111	47	27	3	55	110111	67	37
I	8	001000	10	08	Y	24	011000	30	18	o	40	101000	50	28	4	56	111000	70	38
J	9	001001	11	09	Z	25	011001	31	19	p	41	101001	51	29	5	57	111001	71	39
K	10	001010	12	0A	a	26	011010	32	1A	q	42	101010	52	2A	6	58	111010	72	3A
L	11	001011	13	0B	b	27	011011	33	1B	r	43	101011	53	2B	7	59	111011	73	3B
M	12	001100	14	0C	c	28	011100	34	1C	s	44	101100	54	2C	8	60	111100	74	3C
N	13	001101	15	0D	d	29	011101	35	1D	t	45	101101	55	2D	9	61	111101	75	3D
O	14	001110	16	0E	e	30	011110	36	1E	u	46	101110	56	2E	+	62	111110	76	3E
P	15	001111	17	0F	f	31	011111	37	1F	v	47	101111	57	2F	/	63	111111	77	3F

Рассмотрим пример. Пусть необходимо закодировать сообщение «Map».

Все необходимые преобразования наглядно отображены в табл. 3.2.

Таблица 3.2

Пояснение к кодировке сообщения «Map» в алфавит base64

Исходный текст	M	a	p
Коды ASCII	77 (hex**4d**)	97 (hex**61**)	112 (hex**70**)
Двоичный вид	0 1 0 0 1 1 0 1 0 1 1 0 0 0 0 1 0 1 1 1 0 0 0 0		
Полученный индекс в base64	19	22	5	48
Конечный результат в base64	T	W	F	w

В приведенной таблице подчеркивание используется для разделения двоичных кодов символов в ASCII.

Если, к примеру, входная строка имеет вид «qwe», то выходная – «cXdl», при расширении входной строки на один символ («qwer») на выходе получим «cXdlcg==»

Листинг 3.1 ниже содержит программный код для выполнения рассмотренного преобразования [7]. Онлайн-кодировщик доступен на сайте [8].

```
function base64_encode (s)
{
  // the result/encoded string, the padding string, and the pad count
  var base64chars =
="ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz012
3456789+/";
  var r = "";
  var p = "";
  var c = s.length % 3;
  // add a right zero pad to make this string a multiple of 3 characters
  if (c > 0) {
    for (; c < 3; c++) {
      p += '=';
      s += "\0";
    }
  }
```

```
  // increment over the length of the string, three characters at a time
  for (c = 0; c <s.length; c += 3) {
   // we add newlines after every 76 output characters, according to
the MIME specs
   if (c > 0 && (c / 3 * 4) % 76 == 0) {
    r += "\r\n";
   }
   // these three 8-bit (ASCII) characters become one 24-bit number
   var n = (s.charCodeAt(c) << 16) + (s.charCodeAt(c+1) << 8) +
s.charCodeAt(c+2);
   // this 24-bit number gets separated into four 6-bit numbers
   n = [(n >>> 18) & 63, (n >>> 12) & 63, (n >>> 6) & 63, n & 63];
   // those four 6-bit numbers are used as indices into the base64 char-
acter list
   r += base64chars[n[0]] + base64chars[n[1]] + base64chars[n[2]] +
base64chars[n[3]];
  }
  // add the actual padding string, after removing the zero pad
  return r.substring(0, r.length - p.length) + p;
}
```

Пример реализации кодировщика в формат base64

При изучении раздела курса, касающегося криптографического преобразования данных, мы вернемся к вопросу о расширения области применения base64-формата. Сейчас же ограничимся рассмотрением особенностей дальнейшего преобразования данных этого формата с использованием операции XOR (вспомним, что эта операция называется также сложением по модулю 2, логическим сложением, исключающим «ИЛИ», строгой дизъюнкций, поразрядным дополнением).

Вспомним таблицу истинности над двумя переменными: *a* и *b* (табл. 3.3).

Таблица 3.3

Таблица истинности операции XOR

a	*b*	*a*XOR*b*
0	0	0
0	1	1
1	0	1
1	1	0

Если *a* и *b* имеют длину более 1 бита, к примеру 1 байт, то рассматриваемая операция над ними выполняется побитово. Указанным байтам могут соответствовать символы в определенной кодировке. Положим, символу «M» (hex**4d**) соответствует 8-битный код 01001101 (см. табл. 3.2), а символу «a» (hex**61**) соответствует код 01100001, тогда операция сложения по модулю 2 этих двух бинарных кодов дает 00101100 ((hex**2c**), или символ «,».

3.2. Практическое задание

1. Создать собственное приложение (приветствуется!) или воспользоваться *Base64-онлайн-кодировщиком*, с помощью которого конвертировать произвольный документ (а) на латинице (можно использовать документ из лабораторной работы № 1) в документ (б) формата base64. В качестве входных данных можно использовать указанный преподавателем вариант из списка:

- входные параметры;
- текстовый файл (*.txt);
- документ Word (*.doc);
- документ Word (*.docx);
- документ PowerPoint (*.ppt, *.pptx);
- архив (*.zip);
- текстовая строка;
- случайное число (от 999999);
- PDF-файл;
- архив (*.rar);
- архив (*.7z).

2. С помощью приложения, созданного в лабораторной работе № 1, получить распределение частотных свойств алфавитов по документам (а) и (б). Вычислить энтропию Хартли и Шеннона, а также избыточность алфавитов. Объяснить полученный результат.

3. Написать функцию, которая принимает в качестве аргументов два буфера (*a* и *b*) одинакового размера и возвращает XOR (собственная фамилия (*a*) и имя (*b*); при разной длине меньшую дополнить нулями). Входные аргументы представлять: 1) в кодах ASCII; 2) в кодах base64. Что будет результатом операции *a*XOR*b*XOR*b*?

При написании не использовать стандартные функции языка программирования. Итоговые данные сравнить с результатами использования стандартных функций языка программирования (если они есть).

4. Результаты оформить в виде отчета по установленным правилам.

ВОПРОСЫ ДЛЯ КОНТРОЛЯ И САМОКОНТРОЛЯ

1. Что такое base64?

2. Как проверить, была ли определенная строка символов закодирована в base64?

3. Как с помощью base64 проверить подлинность вводимых данных в форму пароля и логина?

4. Охарактеризовать энтропийные свойства алфавитов в проанализированных форматах данных.

5. Объяснить результат операции *a*XOR*b*XOR*b*. Где может найти применение такая операция?

6. Как будут выглядеть строки:

efd8b295a633908a3c0828b2

faea8766

4d72cde3aaa0

после их конвертации в base64?

7. Результатом операции *a*XOR*b* (*a* – каждый байт строки, *b* – некоторая неизменная величина) будет строка:

1f180d1e1f04051c404c0f19

1f180308050d024c030a4c18

04094c1f18030009024c1c00

Найти значение *b*.

Лабораторная работа № 4

ИЗБЫТОЧНОЕ КОДИРОВАНИЕ ДАННЫХ В ИНФОРМАЦИОННЫХ СИСТЕМАХ. КОД ХЕММИНГА

Цель: приобретение практических навыков кодирования/декодирования двоичных данных при использовании кода Хемминга.

Задачи:

1. Закрепить теоретические знания по использованию методов помехоустойчивого кодирования для повышения надежности передачи и хранения в памяти компьютера двоичных данных.

2. Разработать приложение для кодирования/декодирования двоичной информации кодом Хемминга с минимальным кодовым расстоянием 3 или 4.

3. Результаты выполнения лабораторной работы оформить в виде описания разработанного приложения, методики выполнения экспериментов с использованием приложения и результатов эксперимента.

4.1. Теоретические сведения

4.1.1. Основные понятия и определения

Надежность системы – характеристика способности программного, аппаратного, аппаратно-программного средства выполнить при определенных условиях требуемые функции в течение конкретного периода времени.

Достоверность работы системы (устройства) – свойство, характеризующее истинность конечного (выходного) результата работы (выполнения программы), определяемое способностью средств контроля фиксировать правильность или ошибочность работы.

Ошибка устройства – неправильное значение сигнала (бита – в цифровом устройстве) на внешних выходах устройства или отдельного его узла, вызванное технической неисправностью, или воздействующими на него помехами (преднамеренными либо непреднамеренными), или иным способом.

Ошибка программы – проявляется в не соответствующем реальному (требуемому) промежуточном или конечном значении

(результате) вследствие неправильно запрограммированного алгоритма или неправильно составленной программы.

Как следует из вышеприведенного определения, надежность есть внутреннее свойство объекта, заложенное в него при изготовлении и проявляющееся во время эксплуатации. Вторая особенность надежности состоит в том, что она проявляется во времени. И третья особенность надежности выражается по-разному при различных условиях эксплуатации и различных режимах применения объекта (информационной системы в целом, отдельного ее блока, канала передачи сообщения, оперативной или внешней памяти компьютера).

Надежность является комплексным свойством, включающим в себя единичные свойства: безотказность, ремонтопригодность, сохраняемость, долговечность.

Безотказность – это свойство технического объекта непрерывно сохранять работоспособное состояние в течение некоторого времени (или *наработки*). Наработка, как правило, измеряется в единицах времени.

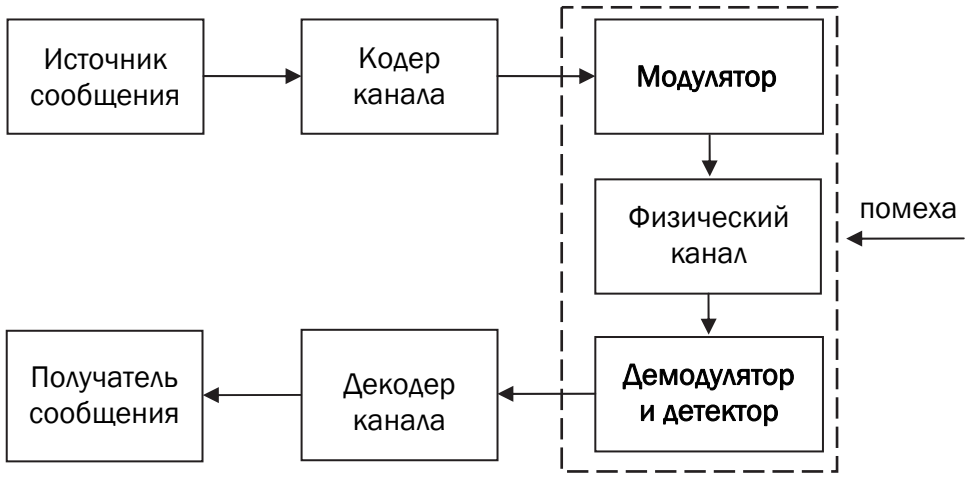
Ремонтопригодность – это свойство технического объекта, заключающееся в приспособленности к поддержанию и восстановлению работоспособного состояния путем технического обслуживания, ремонта (или с помощью дополнительных, избыточных технических средств, функционирующих параллельно с объектом). Большинство современных цифровых систем и устройств (в том числе компьютеры и компьютерные системы, отдельные блоки и модули компьютеров – полупроводниковая, магнитная или оптическая память) содержат специальные средства, призванные автоматически восстанавливать работоспособность этих объектов при нарушении нормального функционирования.

Такие специальные средства контроля называются *избыточными*. На рисунке приведена упрощенная структурная схема системы передачи данных с избыточными средствами аппаратного контроля.

Не вдаваясь в детали функционирования блоков, названия которых на схеме выделены полужирным, рассмотрим основные принципы работы представленной схемы в соответствии с задачами данной лабораторной работы. Как видим, сначала осуществляется формирование данных в виде двоичных символов. Затем *кодер* канала вносит в принятую информационную последовательность некоторую *избыточность* (данный процесс называется

кодированием или *помехоустойчивым кодированием*), которую *декодер* может использовать для исправления возникающих при передаче данных по каналу связи ошибок.

Таким образом, простейшая структурная схема дополнена двумя интересующими нас блоками: кодером (канала), осуществляющим преобразование исходного сообщения (*информационного слова*) X_k (длина сообщения – k символов) в избыточное сообщение (*кодовое слово*) X_n длиной n символов ($n > k$), и декодером (канала).



Структурная схема системы передачи данных
с избыточными средствами аппаратного контроля

Изначальной причиной нарушения нормальной работы цифрового устройства являются технические *дефекты* (неисправности), возникающие внутри узлов или блоков устройства либо в каналах связи между ними.

Дефекты или неисправности могут приводить либо к кратковременному нарушению достоверности работы устройства (*сбой*), либо к полной и окончательной потере достоверности (*отказ*).

В каждом из этих случаев следствием неисправности являются ошибки в информации (*информационные ошибки*). Чаще всего причиной ошибок бывают внешние помехи, как это показано на рисунке. Количество таких ошибок (количество ошибочных двоичных символов) принято называть *кратностью ошибки*.

Пример. Исходная (правильная) информационная последовательность $X_k = 1001$. Длина этой последовательности равна 4 битам ($k = 4$). Некоторая из перечисленных причин привела к тому, что в этой последовательности появились две ошибки (*кратность*

ошибки равна двум): Y_k = 1<u>11</u>1. Здесь Y_k – сообщение на выходе канала (без учета его кодирования/декодирования).

Обнаружение и/или исправление подобных ошибок как раз и призваны обеспечить кодер и декодер. Дополнительную информацию по рассматриваемым аспектам можно найти в пособиях [5, 9].

При использовании избыточных кодов исходные данные делятся на блоки из k битов (называются информационными битами). В процессе кодирования каждый k-битный блок данных преобразуется, как было отмечено выше, в блок из n битов (кодовое слово). Число k часто называется *размерностью кода*. Таким образом, к каждому блоку данных в процессе кодирования присоединяются $r = n - k$ битов, которые называют *избыточными битами* (redundant bits), *битами четности* (parity bits) или *контрольными битами* (check bits); новой информации они не несут.

Для обозначения описанного кода обычно пользуются записью (n, k) и говорят, что данный код использует n символов для передачи (хранения) k символов сообщения. Отношение числа битов данных к общему числу битов k/n именуется степенью кодирования (code rate) – доля кода, которая приходится на полезную информацию. Еще одним важным параметром кода является *расстояние Хемминга* (d), которое показывает, что два кодовых слова различаются по крайней мере в d позициях.

В общем случае код позволяет обнаруживать $t_о$ ошибок:

$$t_о = \begin{cases} \dfrac{d}{2}, & d - \text{четное}; \\ \dfrac{d-1}{2}, & d - \text{нечетное}. \end{cases}$$

Количество исправляемых кодом ошибок $t_и$ определяется следующим образом:

$$t_и = \begin{cases} \dfrac{d-1}{2}, & d - \text{нечетное}; \\ \dfrac{d-2}{2}, & d - \text{четное}. \end{cases}$$

Во всех этих простых математических выражениях d соответствует минимальному кодовому расстоянию Хемминга анализируемого кода.

4.1.2. Теоретические основы линейных блочных кодов

К. Шеннон сформулировал теорему для случая передачи дискретной информации по каналу связи с помехами, утверждающую, что вероятность ошибочного декодирования принимаемых сигналов может быть обеспечена сколь угодно малой путем выбора соответствующего способа кодирования сигналов. В теореме Шеннона не говорится о том, как нужно строить необходимые помехоустойчивые коды. Однако в ней указывается на принципиальную возможность кодирования, при котором может быть обеспечена сколь угодно высокая надежность передачи.

Код Хемминга относится к классу линейных блочных кодов.

Линейные блочные коды – это класс кодов с контролем четности, которые можно описать парой чисел (n, k).

Для формирования r проверочных символов (кодирования), т. е. вычисления проверочного слова X_r, используется *порождающая матрица* G: совокупность базисных векторов будем далее записывать в виде матрицы G размерностью $k \times n$ с единичной подматрицей (I) в первых k строках и столбцах:

$$G = [P \mid I]. \tag{4.1}$$

Более точно матрица G называется *порождающей матрицей линейного корректирующего кода в приведенно-ступенчатой форме*. Кодовые слова являются линейными комбинациями строк матрицы G (кроме слова, состоящего из нулевых символов).

Кодирование заключается в умножении вектора сообщения X_k длиной k на порождающую матрицу по правилам матричного умножения (все операции выполняются по модулю 2). Очевидно, что при этом первые k символов кодового слова равны соответствующим символам сообщения, а последние r символов образуются как линейные комбинации первых [5, 9].

Для всякой порождающей матрицы G существует матрица H размерности $r \times n$, задающая базис нулевого пространства кода и удовлетворяющая равенству

$$G \cdot H^T = 0. \tag{4.2}$$

Справедливо также

$$X_n \cdot H^T = H \cdot (X_n)^T = 0. \tag{4.3}$$

В последнем выражении символ «*T*» означает *транспонирование*, а $X_n = x_1, x_2, \ldots, x_n$.

Матрица *H*, называемая *проверочной*, равна

$$H = \left[-P^T \mid I \right] = P' \mid I. \tag{4.4}$$

В коде Хемминга с минимальным кодовым расстоянием $d_{\min} = 3$ проверочная матрица *H* имеет классический вид и состоит из двух подматриц: *P*' размером $k \times r$ и *I* размером $r \times r$ соответственно.

В последнем выражении *I* – единичная матрица порядка *r* ($r \times r$).

Количество *r* избыточных (проверочных) символов кодового слова определяется из следующей простой логической цепи рассуждений.

Общее число всех возможных комбинаций 2^r должно удовлетворять неравенству

$$2^r \geq n + 1$$

в силу того, что

$$2^r \geq n+1 \Rightarrow 2^r \geq k + r + 1 \Rightarrow r \geq \log_2(k + r + 1) \Rightarrow$$
$$r \geq \log_2(k+1).$$

Присутствие цифры «1» в приведенных выражениях соотносит ее с нулевым вектор-столбцом, который в матрице не используется.

Например, для (7, 4)-кода Хемминга проверочная матрица в упорядоченном виде выглядит так:

$$H_{(7,4)} = \begin{vmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{vmatrix}.$$

Результат умножения сообщения на выходе канала передачи (Y_n) или (что равнозначно) сообщения, считываемого из памяти, на проверочную матрицу (*H*) называется ***синдромом*** (***вектором ошибки***) ***S***:

$$S = H \cdot (Y_n)^T = Y_n \cdot H^T, \tag{4.6}$$

где $Y_n = y_1, y_2, \ldots, y_n$ – принятый вектор (сообщение на выходе канала), полученный после передачи либо считывания из памяти. Вектор Y_n обычно представляют в следующем виде:

$$Y_n = X_n + E_n, \tag{4.7}$$

где $E_n = e_1, e_2, \ldots, e_n$ – вектор ошибки.

Синдром – это результат проверки четности, выполняемой над сообщением Y_n для определения его принадлежности заданному набору кодовых слов. При положительном результате проверки синдром S равен 0, т. е. $Y_n = X_n$. Если Y_n содержит ошибки, которые можно исправить, то синдром имеет определенное ненулевое значение, что позволяет обнаружить и исправить конкретную ошибочную комбинацию.

Важно запомнить, что в силу выражений (4.3)–(4.7) ненулевой синдром всегда равен сумме по модулю 2 тех векторстолбцов матрицы *H*, номера которых соответствуют номерам ошибочных битов в слове Y_n.

Корректирующая способность кода Хемминга с $d_{\min} = 3$ может быть увеличена введением дополнительной проверки на четность. В этом случае проверочная матрица будет иметь вид:

$$H' = \left|\begin{array}{c:c} & 0 \\ & 0 \\ H & \vdots \\ & 0 \\ \hdashline 1\ 1\ \cdots\ 1 & 1 \end{array}\right| \qquad (4.8)$$

Так, минимальное кодовое расстояние такого кода будет равно 4: $d_{\min} = 4$. Такой код может исправлять все единичные ошибки с одновременным обнаружением всех двойных в анализируемом кодовом слове.

При этом нужно помнить, что вид матрицы (4.8) не соответствует ее каноническому представлению, поскольку во всех столбцах единичной матрицы, кроме последнего, будет по 2 единицы.

Для придания матрице канонического вида необходимо сложить посимвольно все строки между собой и результат сложения записать в последнюю строку (под горизонтальной линией в (4.8)).

4.2. Практическое задание

1. На основе информационного сообщения, представленного символами русского/английского алфавитов, служебными символами и цифрами, содержащегося в некотором текстовом файле (согла-

совать с преподавателем), сформировать информационное сообщение в двоичном виде; длина сообщения в бинарном виде должна быть не менее 16 символов. Для выполнения этого задания можно использовать коды ASCII символов алфавита либо результаты лабораторной работы № 3.

2. Для полученного информационного слова построить проверочную матрицу Хемминга (значение минимального кодового расстояния согласовать с преподавателем).

3. Используя построенную матрицу, вычислить избыточные символы (слово X_r).

4. Принять исходное слово со следующим числом ошибок: 0, 1, 2. Позиция ошибки определяется (генерируется) случайным образом.

5. Для полученного слова $Y_n = Y_k, Y_r$, используя уже известную проверочную матрицу Хемминга, вновь вычислить избыточные символы (обозначим их Y_r'), используя выражение (4.6).

6. Вычислить и проанализировать синдром. В случае, если анализ синдрома показал, что информационное сообщение было передано с ошибкой (или 2 ошибками), сгенерировать унарный вектор ошибки $E_n = e_1, e_2, \ldots, e_n$ и исправить одиночную ошибку, используя формулу (4.7); проанализировать ситуацию при возникновении ошибки в 2 битах.

7. Результаты оформить в виде отчета по установленным правилам.

Файл анализируемой информации (а соответственно и интерфейс приложения) должен содержать исходное информационное сообщение, значения величин k, r, n, проверочную матрицу Хемминга $H_{n,k}$, слово X_n, X_r, Y_n,Y_r, Y_r', синдром S, вектор ошибки E_n. Программа не должна быть чувствительна к длине информационного сообщения.

Некоторой сложностью, с которой можно столкнуться при выполнении данной лабораторной работы, является выбор и реализация алгоритма построения проверочной матрицы Хемминга.

Существует несколько способов, позволяющих заполнить матрицу Хемминга. Одним из простых является следующий.

Поскольку известно, что минимальный вес столбцов в подматрице P' (см. выражение (4.5)) равен 2, вычисляем значение бинома Ньютона по следующей формуле:

$$C_{wt}^{r} = \frac{r!}{wt(r-wt)!},$$

где $wt = 2$, а $r = \log_2 k + 1$; wt – это вес (количество единиц).

Бином Ньютона позволяет вычислить количество различных комбинаций из нулей и единиц, которые мы можем получить при заданных значениях *r* и *wt*. Ниже приведен синтаксис функций, позволяющих вычислить бином Ньютона (листинг 4.1).

```
int fact (int n) //функция вычисления
                  //факториала натурального числа n
{
int answer;   //переменная, которая будет хранить
               //значение факториала натурального числа n
if (n==1)
return 1;     //1 (1!=1)
              //рекурсивное обращение функции fact() к самой себе
answer=fact(n - 1)*n;
return (answer);//возвращение результата
                //работы функции в место
                //вызова ее в теле основной программы
}
                //функция вычисления бинома Ньютона
int binom(int wt,int r) {
int C;          //переменная, которая будет хранить значение
                //бинома Ньютона
C=fact(r)/(fact(wt)*fact(r - wt));
return (C); //возвращение результата работы
            //функции в место вызова ее в теле основной программы
}
```

Листинг 4.1. Пример реализации алгоритма для вычисления бинома Ньютона (числа сочетаний)

Далее переводя натуральные числа, начиная с 3 (первое натуральное число, которое при переводе в двоичную систему счисления имеет вес, равный 2) в двоичную систему счисления и дополняя полученную комбинацию нулями до *r* позиций (если это необходимо), по очереди заполняем столбцы подматрицы *P*'.

Например, положим, что $r = 5$, $wt = 2$.

Тогда

$$C_2^5 = \frac{5!}{2(5-2)!} = 10 .$$

$3_2 = 11$, т. е. первым столбцом матрицы P' будет вектор [$\underbrace{00\ 0\ 11}$]

дополнено в соответствии со значением r

$4_2 = 100 - wt\ (4_2) = 1$ – не подходит,
$5_2 = 101 - wt\ (5_2) = 2$.
Следовательно, второй столбец матрицы $A_{k,r}$: 00101, и т. д.
Программно это может выглядеть в соответствии с листингом 4.2.

```
itoa(num,buf,2); //представление натурального
                 //числа в двоичной системе счисления
int len=strlen(buf);
            //переменная len хранит длину
            //полученной двоичной последовательности
sum=0;    //переменная, используемая для вычисления
            //веса полученной двоичной последовательности
            //(двоичного представления натурального числа n)
for (int i=0;i<len;i++)
 {
  if (buf[i]=='1')
sum++;    //при встрече в двоичном
            //представлении числа символа '1' счетчик sum
            //увеличивается на 1
 }
if (sum==wt) //если вес числа в двоичном
               //представлении равен текущему
               //весу (первоначально wt=2)
 {
 count++; //тогда число комбинаций,
            //подходящих для заполнения
            //столбцов матрицы Ak,r,
            //увеличивается на 1
if (len<r)  //если длина полученной
            //двоичной последовательности
            //меньше r, происходит
            //увеличение ее на (r - длина
            //последовательности) нулей
   {
    for (int j=0;j<(r - len);j++)
     {
      char buf1[80]={'0'};
```

```
strcat(buf1,buf);
strcpy(buf,buf1);
    }
  }
 for (int i=0;i<r;i++){
                //заполнение столбца матрицы Ak,r
  A[i][count - 1]= buf[i];}
 }
num++;       //переход к  следующему натуральному числу
```

Листинг 4.2. Программная реализация алгоритма создания проверочной матрицы кода Хемминга с $d_{min} = 3$

В случае если все комбинации при заданных r и wt уже исчерпаны, увеличиваем вес wt столбцов матрицы на единицу и опять вычисляем бином Ньютона.

В заполнении матрицы I нет никаких сложностей. Для начала элементы матрицы инициализируются нулями. Если общее количество столбцов матрицы $H_{n,r}$ $n = k + r$, а первый столбец матрицы I имеет индекс k (нумерация элементов матрицы $H_{n,r}$ начинается с 0), тогда это можно реализовать с помощью кода, представленного на листинге 4.3.

```
for (int i=k; i<n; i++)
 {
  for (int j=0;j<r;j++)
   {
    if (j=i - k) I[i,j]=1;
    else I[i,j]=0;
   }
 }
```

Листинг 4.3. Фрагмент кода для заполнения столбцами единичной матрицы

Далее при вычислении i-го символа слова X_r достаточно подсчитать количество пар (1; 1) в соответствующих позициях слова X_k и i-й строки подматрицы P'. Если количество таких пар четное, т. е. по модулю 2 равно 0, то i-й элемент слова $X_r[i]$ равен 0, в обратном случае – 1.

Вычисление синдрома – это сложение по модулю 2 векторов (слов) Y_r и Y_r'. Пусть размерность этих векторов равна r (нумера-

цию элементов векторов примем с 0). Тогда можно использовать код, показанный на листинге 4.4.

```
for (int i=0;i<r;i++)
 {
  if (Yr[i]<>Yr'[i])
   s[i]=1;
  else s[i]=0;
 }
```

Листинг 4.4. Фрагмент кода для вычисления синдрома

Из последнего следует, что *i*-й элемент синдрома *S* будет равен 0, если *i*-е элементы векторов Y_r и Y_r' равны между собой, и 1 – в противном случае.

Если анализ синдрома показал, что переданное слово содержит ошибку, ищем номер позиции в слове Y_n, в которой эта ошибка возникла. Зная позицию ошибки, можем сформировать унарный вектор ошибки E_n.

Пусть позицию ошибки хранит некоторая переменная *mist*. Первоначально необходимо проинициализировать все элементы вектора E_n нулями, далее выполнить анализ и вычисления в соответствии с листингом 4.5.

```
for (int i=0;i<n;i++)
 {
  if (i<>mist) E[i]=0;
  else E[i]=1;
 }
```

Листинг 4.5. Фрагмент кода программы для определения вектора ошибки

Чтобы исправить ошибку, необходимо сложить по модулю 2 векторы Y_n и E_n в соответствии с выражением (4.7).

ВОПРОСЫ ДЛЯ КОНТРОЛЯ И САМОКОНТРОЛЯ

1. В чем заключается цель и функциональная сущность преобразования информации на основе избыточного кодирования?

2. Пояснить зависимость r от длины информационного слова k. Охарактеризовать относительную избыточность сообщения и время его передачи по сети.

3. Записать проверочную матрицу кода Хемминга с $d_{\min} = 3$ и $d_{\min} = 4$ для $k = 4; 6; 8; 9; 10; 15; 16$.

4. Записать проверочную матрицу кода простой четности для k из вопроса 3. Пояснить на примере определение минимального кодового расстояния Хемминга для данного кода.

5. Предположим, есть выбор (при построении матрицы кода) между вектор-столбцами большего и меньшего веса. Какой вариант Вы предпочтете и почему?

Лабораторная работа № 5

ИЗБЫТОЧНОЕ КОДИРОВАНИЕ ДАННЫХ В ИНФОРМАЦИОННЫХ СИСТЕМАХ. ИТЕРАТИВНЫЕ КОДЫ

Цель: приобретение практических навыков кодирования/декодирования двоичных данных при использовании итеративных кодов.

Задачи:

1. Закрепить теоретические знания по использованию итеративных кодов для повышения надежности передачи и хранения в памяти компьютера двоичных данных.

2. Разработать приложение для кодирования/декодирования двоичной информации итеративным кодом с различной относительной избыточностью кодовых слов.

3. Результаты выполнения лабораторной работы оформить в виде описания разработанного приложения, методики выполнения экспериментов с использованием приложения и результатов эксперимента.

5.1. Теоретические сведения

5.1.1. Итеративные коды: определение, принципы построения и использования

Итеративные коды относятся к классу *кодов произведения*.

Кодом произведения двух исходных (базовых) помехоустойчивых кодов называется такой *многомерный помехоустойчивый код*, кодовыми последовательностями которого являются все двумерные таблицы со строками кода (k_1) и столбцами кода (k_2).

Итеративные коды могут строиться на основе использования дву-, трехмерных матриц (таблиц) и более высоких размерностей. Каждая из отдельных последовательностей информационных символов кодируется определенным линейным кодом (групповым или циклическим). Получаемый таким образом итеративный код также является *линейным*.

Простейшим из итеративных кодов является *двумерный* код с проверкой на четность по строкам и столбцам. Итеративные

коды, иногда называемые *прямоугольными* кодами (англ. *rectangular code*) либо *композиционными* (англ. *product code*), являются одними из самых простых (с точки зрения аппаратной реализации) избыточных кодов, позволяющих исправлять ошибки в информационных словах.

Основное достоинство рассматриваемых кодов – простота как аппаратной, так и программной реализации. Основной недостаток – сравнительно высокая избыточность.

В упомянутой двумерной матрице кодовые слова записываются в виде таблицы. Проверочные символы вычисляются исходя из того, что строки и столбцы должны содержать четное (нечетное) число единиц. Например, при кодировании информационного слова X_k = **011101111** с помощью таблицы с четностью по строкам и столбцам получим избыточные символы $X_r = X_h, X_v, X_{hv} = 0010011$, как показано на рис. 5.1 (информационные символы выделены жирным шрифтом, а проверочные – курсивом).

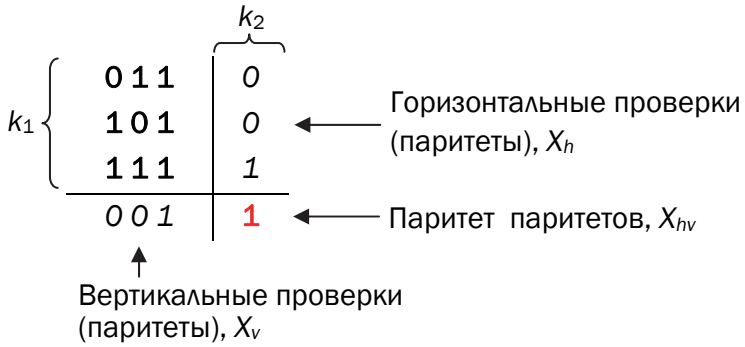


Рис. 5.1. Пояснение к принципу формирования избыточных символов итеративного кода

В соответствии с рис. 5.1 кодовое слово будет иметь следующий вид: X_n = **011101111***0010011*. Как видно, избыточные символы (называемые также паритетами) в приведенном кодовом слове в принятом порядке (X_h, X_v, X_{hv}) записываются сверху вниз, справа налево. Возможен обратный или иной порядок. Важно только, чтобы при декодировании сообщения использовался аналогичный порядок следования паритетов. Символ X_{hv} (паритет паритетов) равен сумме по модулю 2 символов информационного слова X_k, а также проверочных символов X_v и X_h.

Поскольку двумерная матрица формируется как комбинация двух кодов простой четности (по каждому измерению), каждый из

которых характеризуется минимальным кодовым расстоянием $d_{\min} = 2$, то полученный итеративный код ($r = k_1 + k_2$) будет характеризоваться минимальным кодовым расстоянием, равным произведению $d_{\min}$ по строкам и по столбцам, т. е. 4.

Использование символа X_{hv} обеспечивает минимальное кодовое расстояние такого итеративного кода $d_{\min}$ ($r = k_1 + k_2 + 1$) на единицу больше. В этом легко обнаруживается сходство кода с кодом Хемминга при $d_{\min} = 4$.

Нетрудно также представить процесс вычисления проверочных символов кодового слова для примера на рис. 5.1 с помощью проверочной матрицы Хемминга и соотношения (4.4). Для указанного примера проверочная матрица кода с $d_{\min} = 3$ выглядит так:

$$H_{15,9} = \begin{matrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & & & & & \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & & 1 & & & & \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & & & 1 & & & \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & & & & 1 & & \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & & & & & 1 & \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & & & & & & 1 \end{matrix}$$

Передачу символов кода обычно осуществляют последовательно символ за символом, от одной строки к другой, либо параллельно целыми строками.

Как показано на рис. 5.1, проверочный символ есть сверка по модулю 2 информационных символов, записанных в соответствующие строку или столбец матрицы.

Декодирование начинают сразу, не ожидая поступления всего блока информации. Проверка соответствия избыточных символов полученного слова ($Y_r = Y_h, Y_v, Y_{hv}$ либо $Y_r = Y_h, Y_v$) при декодировании позволяет обнаружить любое нечетное число искаженных символов, расположенных в одной строке или в одном столбце. Формально такое декодирование осуществляется сравнением принятых (Y_h, Y_v, Y_{hv}) и вновь вычисленных (Y'_h, Y'_v, Y'_{hv}) для полученного слова паритетов. В упрощенной форме это показано на рис. 5.2. Определение местоположения одиночной ошибки по строке указывает на наличие ошибки в этой строке матрицы, а проверка по столбцу – конкретный символ (рис. 5.2, *а*).

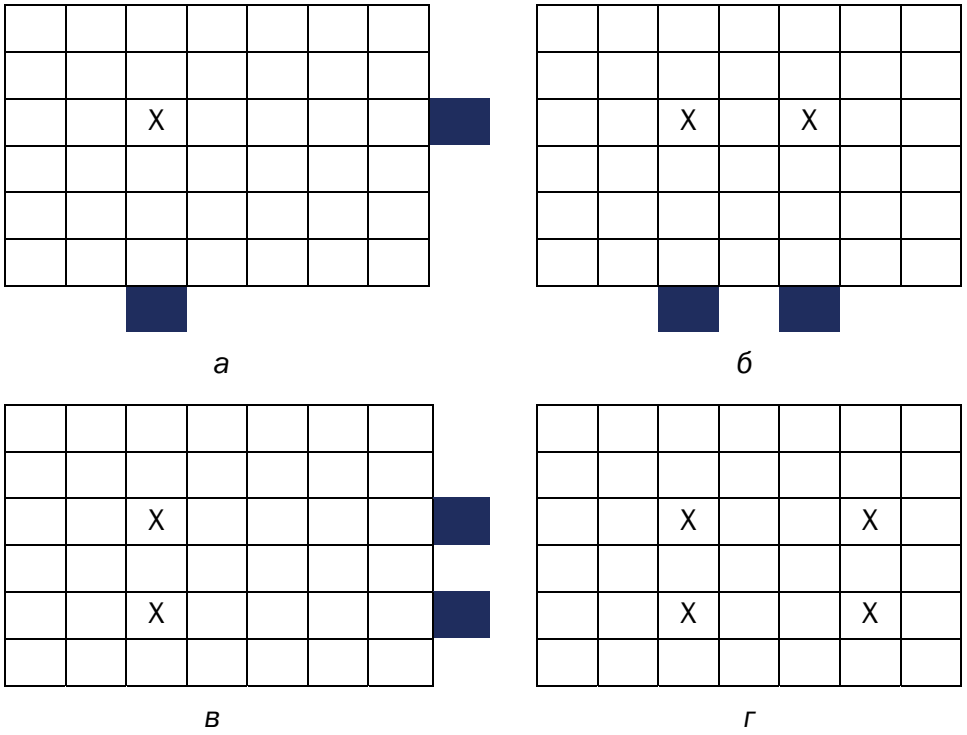


Рис. 5.2. Пояснение к принципу обнаружения местоположения ошибочных битов в принятом сообщении:
а–г – варианты местоположения ошибок

Однако этим кодом не могут быть установлены местоположения многократных ошибок, имеющих четное число искаженных символов как по строкам, так и по столбцам (рис. 5.2, *б*, *в*). Простейшая необнаруживаемая ошибка содержит четыре искаженных символа, расположенных в вершинах прямоугольника или квадрата (рис. 5.2, *г*). Это происходит из-за того, что четность (паритет) по строкам и по столбцам матрицы не нарушается. Полезную информацию о кодировании и декодировании информации итеративным кодом можно найти в источниках [10, 11].

5.1.2. Многомерные линейные итеративные коды

Принято считать рассматриваемый код *многомерным*, если количество измерений, по которым вычисляются и анализируются паритеты, не менее 3. Таким образом, простейшим многомерным линейным итеративным кодом является код *трехмерный*. Его достаточно подробное описание и особенности использования можно найти в статьях [12–14].

Пример реализации трехмерного случая иллюстрирует рис. 5.3. Дополнительно к двум кодам на основе кодов простой четности (по вертикали и горизонтали) избыточные символы вычисляются по диагонали: X_d.

Приведенную на рис. 5.3 трехмерную структуру итеративных кодов можно дополнить достаточно большим числом разнообразных проверок на четность по диагоналям, тем самым получив набор кодов с высокими корректирующими возможностями.

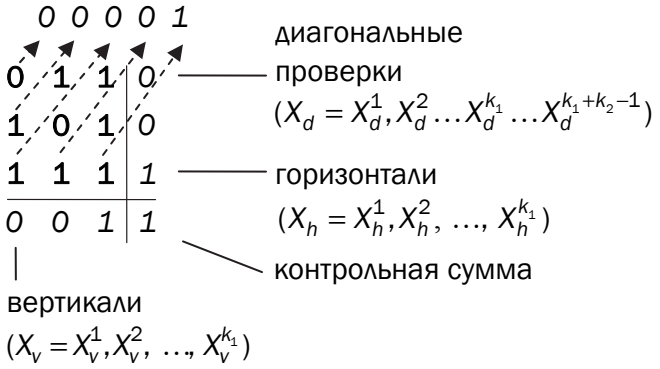


Рис. 5.3. Принцип формирования избыточных символов для линейного итеративного кода с диагональными проверками

Принцип формирования проверочных символов для кодов, построенных на основе вычисления различных диагональных паритетов, при $k = 64$ бита приведен на рис. 5.4 (пять групп линейно независимых паритетов): в первой плоскости, или матрице, записаны первые 16 битов сообщения, во второй плоскости-матрице – символы с 17 по 32 и т. д. Указанные плоскости располагаются по условной оси Z.

Как показано на рис. 5.4, для информационного сообщения из 64 битов ($X_k = x_1, x_2, \ldots, x_{64}$) сформировано 80 битов общего избыточного слова, $X_{rr} = x_{r1}, x_{r2}, \ldots, x_{r80}$:

$$x_{r1} = x_1 + x_2 + x_3 + x_4,$$

$$x_{r2} = x_5 + x_6 + x_7 + x_8, \ldots,$$

$$x_{r5} = x_1 + x_5 + x_9 + x_{13}, \ldots,$$

$$x_{r14} = x_3 + x_8 + x_9 + x_{14}, \ldots,$$

$$x_{r16} = x_1 + x_6 + x_{11} + x_{16},$$

$$x_{r17} = x_{17} + x_{18} + x_{19} + x_{20}, \ldots,$$
$$x_{r37} = x_{33} + x_{37} + x_{41} + x_{45}, \ldots,$$
$$x_{r60} = x_{52} + x_{55} + x_{58} + x_{61}, \ldots,$$
$$x_{r80} = x_{16} + x_{32} + x_{48} + x_{64}.$$

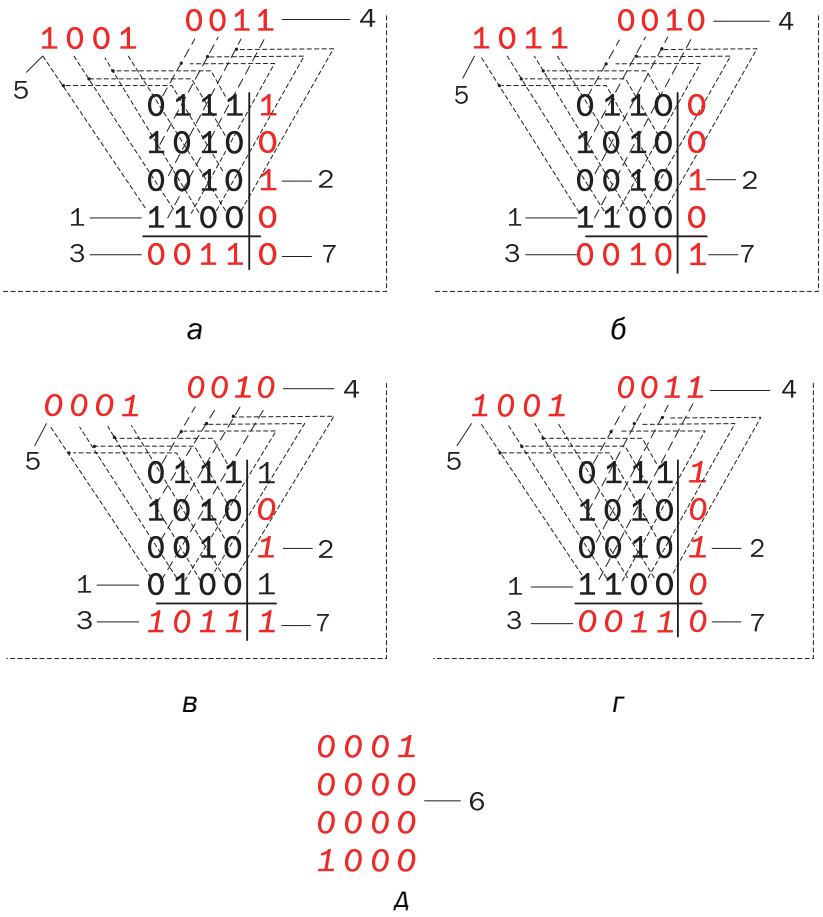


Рис. 5.4. Пояснение к принципу построения кодового слова на основе многомерного линейного итеративного кода с 5 независимыми группами паритетов:
а–г – плоскости с первой по четвертую соответственно;
д – паритеты в *z*-плоскости;
1 – информационные биты; 2 – горизонтальные паритеты;
3 – вертикальные паритеты; 4, 5 – соответственно первые и вторые объединенные диагональные паритеты;
6 – *z*-паритеты; 7 – контрольная сумма (X_{hv})

В табл. 5.1 приведены параметры рассмотренного многомерного итеративного кода, которые можно также отнести к классу *низкоплотностных* [15].

Таблица 5.1

Параметры некоторых многомерных итеративных кодов

Длина информационной последовательности (бит), k	Пять линейно независимых паритетов		Семь линейно независимых паритетов		Девять линейно независимых паритетов	
	Избыточность (бит), r	Скорость передачи, $k/(k + r)$	Избыточность (бит), r	Скорость передачи, $k/(k + r)$	Избыточность (бит), r	Скорость передачи, $k /(k + r)$
64	80	0,44	112	0,36	144	0,31
512	320	0,62	448	0,53	576	0,47
4096	1280	0,76	1792	0,70	2304	0,64

5.2. Практическое задание

Разработать собственное приложение, которое позволяет выполнять следующие операции:

1) вписывать произвольное двоичное представление информационного слова X_k (кодируемой информации) длиной k битов в двумерную матрицу размерностью в соответствии с вариантом либо в трехмерную матрицу в соответствии с вариантом (указаны в табл. 5.2);

2) вычислять проверочные биты (биты паритетов): а) по двум; б) по трем; в) по четырем направлениям (группам паритетов);

3) формировать кодовое слово X_n присоединением избыточных символов к информационному слову;

4) генерировать ошибку произвольной кратности (i, $i > 0$), распределенную случайным образом среди символов слова X_n, в результате чего формируется кодовое слово Y_n;

5) определять местоположение ошибочных символов итеративным кодом в слове Y_n в соответствии с используемыми группами паритетов по пункту (2) и исправлять ошибочные символы (результат исправления – слово Y_n');

6) выполнять анализ корректирующей способности используемого кода (количественная оценка) путем сравнения соответствующих слов X_n и Y_n'; результат анализа может быть представлен в виде отношения общего числа сгенерированных кодовых слов с ошибками определенной одинаковой кратности (с одной ошибкой, с двумя ошибками и т. д.) к числу кодовых слов, содержащих

ошибки этой кратности, которые правильно обнаружены и которые правильно скорректированы.

Таблица 5.2

Варианты заданий

Вариант	Длина информационного слова (бит), k	k_1	k_2	z	Количество групп паритетов
1	16	4 8 4 2	4 2 2 4	– – 2 2	2; 3 2; 3 2; 3; 4; 5 2; 3; 4; 5
2	20	4 2 2 2	5 10 5 2	– – 2 5	2; 3 2; 3 2; 3; 4; 5 2; 3; 4; 5
3	24	4 3 3 6	6 8 3 2	– – 4 2	2; 3 2; 3 2; 3; 4; 5 2; 3; 4; 5
4	32	4 2 8 4	8 16 2 4	– – 2 2	2; 3 2; 3 2; 3; 4; 5 2; 3; 4; 5
5	40	5 4 5 2	8 10 4 10	– – 2 2	2; 3 2; 3 2; 3; 4; 5 2; 3; 4; 5

Пример. Для определенного кодового слова X_n с выбранными группами вычисленных избыточных символов сгенерировано N_1 вариантов этого кодового слова (теперь это слово мы обозначаем Y_n) с 3 ошибками ($i = 3$). Среди N_1 ошибочных кодовых слов в N_2 случаях кратность ошибки идентифицирована правильно ($N_2 \le N_1$) и в N_3 случаях все ошибки скорректированы (слова X_n и Y_n' совпадают). Нужно вычислить соотношения N_2/N_1 и N_3/N_1.

ВОПРОСЫ ДЛЯ КОНТРОЛЯ И САМОКОНТРОЛЯ

1. Охарактеризовать основные параметры итеративного кода.
2. Сравнить основные параметры кодов Хемминга и итеративных кодов.

3. Составить проверочные матрицы кодов Хемминга для кодирования 9-битных сообщений по схеме, представленной на рис. 5.1 (с учетом и без учета символа X_{hv}).

4. Составить проверочные матрицы кодов Хемминга для кодирования 9-битных сообщений по схеме, представленной на рис. 5.3 (с учетом и без учета символа X_{hv}).

5. Составить проверочные матрицы кодов Хемминга для кодирования 9-битных сообщений по схеме, представленной на рис. 5.4 (с учетом и без учета символа X_{hv}).

6. Какое максимальное число ошибок может быть обнаружено итеративным кодом? При каком условии?

7. Определить, какая геометрическая фигура, являющаяся формой для записи символов информационного слова, обеспечивает наименьшую относительную избыточность кодового слова при фиксированном (каком?) k.

8. Результаты оформить в виде отчета по установленным правилам.

Лабораторная работа № 6

ИЗБЫТОЧНОЕ КОДИРОВАНИЕ ДАННЫХ В ИНФОРМАЦИОННЫХ СИСТЕМАХ. ЦИКЛИЧЕСКИЕ КОДЫ

Цель: приобретение практических навыков кодирования/декодирования двоичных данных при использовании циклических кодов (ЦК).

Задачи:

1. Закрепить теоретические знания по алгебраическому описанию и использованию ЦК для повышения надежности передачи и хранения в памяти компьютера двоичных данных, для контроля интегральности файлов информации.

2. Разработать приложение для кодирования/декодирования двоичной информации циклическим кодом.

3. Результаты выполнения лабораторной работы оформить в виде описания разработанного приложения, методики выполнения экспериментов с использованием приложения и результатов эксперимента.

6.1. Теоретические сведения

6.1.1. Основные свойства циклических кодов

Циклические коды – это семейство помехоустойчивых кодов, одной из разновидностей которых являются коды Хемминга.

Основные свойства ЦК:

- относятся к классу линейных, систематических;
- сумма по модулю 2 двух разрешенных кодовых комбинаций дает также разрешенную кодовую комбинацию;
- каждый вектор (кодовое слово), получаемый из исходного кодового вектора путем циклической перестановки его символов, также является разрешенным кодовым вектором; к примеру, если кодовое слово имеет следующий вид: 1101100, то разрешенной кодовой комбинацией будет и такая: 0110110;
- при простейшей циклической перестановке символы кодового слова перемещаются слева направо на одну позицию, как в приведенном примере;

• поскольку к числу разрешенных кодовых комбинаций ЦК относится нулевая комбинация 000...00, то минимальное кодовое расстояние $d_{\min}$ для ЦК определяется минимальным весом разрешенной кодовой комбинации;

• циклический код не обнаруживает только такие искаженные помехами кодовые комбинации, которые приводят к появлению на стороне приема других разрешенных комбинаций этого кода;

• в основе описания и использования ЦК лежит полином или многочлен некоторой переменной (обычно X).

Для более глубокого изучения параметров и свойств ЦК, равно как и других корректирующих кодов, полезно ознакомиться с классическими книгами, например [16–18].

6.1.2. Операции кодирования и декодирования циклических кодов

Рассматриваемые операции сводятся к известным процедурам умножения и деления двоичных чисел либо соответствующих этим числам полиномов. Действия с кодовыми словами производятся по правилам арифметики по модулю 2. Следует помнить, что ***вычитание равносильно сложению***. Это следует из простых рассуждений: из равенства $x^z - 1 = 0$ получаем $x^z = 1$. Прибавив к левой и правой частям по единице, имеем $x^z + 1 = 1 + 1 = 0$. Таким образом, вместо двучлена $x^z - 1$ можно ввести $x^z + 1$ или $1 + x^z$, из чего следует также, что $x^z + x^z = x^z(1 + 1) = 0$.

Понятные примеры перевода двоичных последовательностей в полиномы и наоборот даны в пособии [5]. Здесь приведем лишь один простой пример. Переведем кодовое слово X_n = 101100 (здесь используются символьные обозначения, как в предыдущих лабораторных работах) в полиномиальный вид и получим *полином 5-й степени*:

$$B(X) = 1 \cdot x^5 + 0 \cdot x^4 + 1 \cdot x^3 + 1 \cdot x^2 + 0 \cdot x^1 + 0 \cdot x^0 = \\ = x^5 + x^3 + x^2. \qquad (6.1)$$

В данном выражении коэффициентами порождающего полинома являются: $g_5 = 1$, $g_4 = 0$, $g_3 = 1$, $g_2 = 1$, $g_1 = 0$, $g_0 = 0$. Как видим, полином есть сумма произведений $g_i \cdot x^i$, где $0 \le i \le r$.

Порождающие полиномы циклических кодов. Характеризуя ЦК в общем случае, обычно отмечают следующее: ЦК составляют

множество многочленов $\{B_j(X)\}$ степени r (r – число проверочных символов в кодовом слове), кратных порождающему (образующему) полиному $G(X)$ степени r, который должен быть делителем бинома $X^n + 1$, т. е. остаток после деления бинома на $G(X)$ должен быть нулевым.

Формирование разрешенных кодовых комбинаций ЦК $B_j(X)$ основано на предварительном выборе ***порождающего (генераторного или образующего) полинома G(X),*** который обладает важным отличительным признаком: ***все комбинации $B_j(X)$ делятся на порождающий полином G(X) без остатка***:

$$B_j(X) / G(X) = A_j(X), \qquad (6.2)$$

здесь $B_j(X) = X_n$ – кодовое слово; $A_j(X) = X_k$ – информационное слово.

Степень порождающего полинома определяет число проверочных символов: $r = n - k$. Из этого свойства следует простой способ формирования разрешенных кодовых слов ЦК – умножение информационного слова $A(X)$ на порождающий полином $G(X)$:

$$B_j(X) = A_j(X) \cdot G(X). \qquad (6.3)$$

Порождающими могут быть только такие полиномы, которые являются делителями двучлена (бинома) $X^z + 1$:

$$(X^z + 1) / G(X) = H(X) \qquad (6.4)$$

при нулевом остатке: $R(X) = 0$.

Возможные порождающие полиномы для различных длин k информационного слова, найденные с помощью ЭВМ, сведены в обширные таблицы. В табл. 6.1 приведены полиномы для некоторых значений n, k, r и соответствующие им минимальные кодовые расстояния $d_{\min}$.

Можно добавить, что для $k = 4$ существуют только два порождающих полинома: $x^3 + x + 1$ (1011), который указан в табл. 6.1, и $x^3 + x^2 + 1$ (1101). Напомним, что эти полиномы относятся к *двойственным*: обратная запись (справа налево) одного в бинарной форме дает другой.

Таким образом, в основе построения ЦК лежит операция деления передаваемой кодовой комбинации на порождающий неприводимый полином степени r в соответствии с выражением (6.2). Остаток $R(X)$ от деления используется при формировании проверочных разрядов. При декодировании принятой n-разрядной

кодовой комбинации (Y_n) опять производится ее деление на порождающий (производящий, образующий) полином.

Таблица 6.1

Параметры некоторых циклических кодов

n	k	r	Полином	d_{min}
7	4	3	$x^3 + x + 1$	3
15	11	4	$x^4 + x + 1$	3
15	7	8	$x^8 + x^7 + x^6 + 1$	5
15	5	10	$x^{10} + x^8 + x^5 + x^4 + x^2 + x + 1$	7
31	26	5	$x^5 + x^2 + 1$	3
31	21	10	$x^{10} + x^9 + x^8 + x^6 + x^5 + x^3 + 1$	5
31	16	15	$x^{15} + x^{11} + x^{10} + x^9 + x^8 + x^7 + x^5 + x^3 + x^2 + x + 1$	7
31	11	20	$x^{20} + x^{18} + x^{17} + x^{13} + x^{10} + x^9 + x^7 + x^6 + x^4 + x^2 + 1$	11
31	6	25	$x^{25} + x^{24} + x^{21} + x^{19} + x^{18} + x^{16} + x^{15} + x^{14} + x^{13} + x^{11} + x^9 + x^5 + x^2 + x + 1$	15
63	57	6	$x^6 + x + 1$	3
63	51	12	$x^{12} + x^{10} + x^8 + x^5 + x^4 + x^3 + 1$	5
63	45	18	$x^{18} + x^{17} + x^{16} + x^{15} + x^9 + x^7 + x^6 + x^3 + x^2 + x + 1$	7
63	39	24	$x^{24} + x^{23} + x^{22} + x^{20} + x^{19} + x^{17} + x^{16} + x^{13} + x^{10} + x^9 + x^8 + x^5 + x^4 + x^2 + x + 1$	9

Синдромом ошибки в этих кодах является наличие остатка от деления принятой кодовой комбинации на порождающий полином. Если синдром равен нулю, то считается, что ошибок нет. В противном случае с помощью полученного синдрома можно определить номер разряда принятой кодовой комбинации, в котором произошла ошибка, и исправить ее примерно по той же схеме, которую мы использовали для кода Хемминга.

При этом следует обратить внимание на важную деталь: умножение полинома на x приводит к сдвигу членов полинома на один разряд влево, а при умножении на x^r – на r разрядов влево с заменой r младших разрядов полинома на нули. Деление же полинома на x приводит к соответствующему сдвигу членов полинома вправо с уменьшением показателей членов на 1. Такой сдвиг требует дописать справа r проверочных символов к исходной кодовой комбинации $A_i(X)$ после умножения ее на x^r.

Например, если $A_i(X) = x^3 + x^2 + 1$, т. е. $X_k = 1101$, $r = 2$, то $A_i(X) \cdot x^2 = x^5 + x^4 + x^2$, т. е. 110100, что равнозначно присоединению к бинарному представлению исходного полинома двух нулей справа, т. е. в младших разрядах преобразованного полинома. Если же этот преобразованный полином ($x^5 + x^4 + x^2 = 110100$) разделить на x^2, то мы получим 001101, или просто 1101, т. е. $x^3 + x^2 + 1$.

Кодирование информационного слова. Деление полиномов позволяет представить кодовые слова в виде блочного кода, т. е. информационных X_k ($A_i(X)$) и проверочных X_r ($R_i(X)$) символов. Поскольку число последних равно r, то для компактной их записи в младшие разряды кодового слова надо предварительно к кодируемому (информационному) слову $A_i(X)$ справа дописать r нулевых символов.

Пример. Рассмотрим процедуру кодирования для $X_k = 1001$, т. е. сформируем кодовое слово циклического кода (7, 4).

В данном ЦК $n = 7$, $k = 4$, $r = 3$. Выберем в качестве порождающего полином $G(X) = x^3 + x + 1$ (как видим, это соответствует коду Хемминга).

Запишем информационное слово в виде полинома: $X_k = 1001 = x^3 + 1$.

Далее выполняем вычисление проверочных символов и их присоединение (конкатенацию) к информационному слову для создания кодового слова, которое может передаваться по каналу или записываться в память для хранения:

1. $X_k \cdot X^r = (x^3 + 1) \cdot x^3 = x^6 + x^3 \sim 1001000$ ($n = 7$).
2. $X_k \cdot X^r / G(X) = x^6 + x^3 \mid x^3 + x + 1$

$$\begin{array}{r} x^6 + x^4 + x^3 \\ \hline x^4 \\ x^4 + x^2 + x \\ \hline x^2 + x \end{array} \qquad \begin{array}{l} \\ x^3 + x \end{array}$$

$x^2 + x$ – остаток; таким образом, $X_r = x^2 + x \sim 110$.

3. $X_n = X_k || X_r = 1001110$ – итоговая комбинация ЦК (кодовое слово X_n).

Декодирование принятого сообщения по синдрому. *Основная операция*: принятое кодовое слово (Y_n) нужно поделить на порождающий полином, *который использовался при кодировании*.

Если Y_n принадлежит коду, т. е. слово не искажено помехами, то остаток от деления (синдром) будет нулевым. Ненулевой оста-

ток свидетельствует о наличии ошибок в принятой кодовой комбинации: $Y_n \neq X_n$.

Для исправления ошибки нужно определить *вектор (полином) ошибки* E_n (по аналогии к кодом Хемминга, см. выражение (4.7)).

Принимаем, что после передачи по каналу с помехами кодовое слово можно записать в виде:

$$Y_n = X_n + E_n \quad \text{или } X_n = Y_n + E_n. \tag{6.5}$$

Деление принятого кодового слова на $G(X)$ формально запишем в следующем виде:

$$Y_n / G(X) = U,\ S_r, \tag{6.6}$$

где S_r – остаток от деления $(Y_n) / (G(X))$, или *синдром*.

Здесь, как и в предыдущих лабораторных работах, всякому ненулевому синдрому соответствует определенное расположение (конфигурация) ошибок: синдром для ЦК имеет те же свойства, что и для кода Хемминга.

Для примера предположим, что в сообщении, сформированном выше, произошла ошибка в третьем слева бите, т. е. $Y_n = 10\underline{\mathbf{1}}1110$ ($\sim x^6 + x^4 + x^3 + x^2 + x$).

Выполняем операцию деления в соответствии с формулой (6.6):

$$Y_n / G(X) = x^6 + x^4 + x^3 + x^2 + x \ \Big|\ \frac{x^3 + x + 1}{x^3}$$
$$\underline{x^6 + x^4 + x^3}$$
$$x^2 + x.$$

Мы получили $U = x^3$ и r-разрядный ($r = 3$) синдром $S_r = x^2 + x$, или $S_r = 110$.

Декодирование синдрома и исправление ошибки в принятом сообщении. Декодирование ненулевого синдрома имеет целью определение ошибочного бита в принятом сообщении или, иначе говоря, определение вектора E_n.

Как и в лабораторной работе № 4, поиск ошибочного бита будем производить через поиск соответствия между синдромом и проверочной матрицей кода.

Наряду с полиномиальным способом задания кода, структуру построения кода можно определить с помощью матричного представления. При этом в ряде случаев проще реализуется построение кодирующих и декодирующих устройств ЦК.

Здесь нам следует вернуться к упомянутым матрицам и представлению ЦК с их помощью.

Порождающая матрица G циклического кода имеет в качестве строк векторы:

$$G = \begin{vmatrix} G(X) \\ G(X)/x \\ \dots \\ G(X)/x^{k-1} \end{vmatrix} = \begin{vmatrix} g_r & g_{r-1} & \dots & g_0 & 0 & & \dots & & 0 \\ 0 & g_r & g_{r-1} & \dots & g_0 & 0 & \dots & & 0 \\ & & \dots & \dots & \dots & & & & \\ 0 & \dots & & & 0 & g_r & g_{r-1} & \dots & g_r \end{vmatrix}, \quad (6.7)$$

где $G(X)$ – порождающий полином ЦК; g_0, ..., g_r – коэффициенты полинома (см. формулу (6.1)).

Справедливо:

$$G \cdot H^T = 0 \text{ или } H \cdot G^T = 0. \quad (6.8)$$

С учетом этого *проверочная матрица* H кода строится на основе полинома, полученного в результате следующей операции:

$$H(X) = (X^n + 1) / G(X). \quad (6.9)$$

$$H = \begin{vmatrix} H(x) \\ xH(x) \\ \\ x^{r-1}H(x) \end{vmatrix} = \begin{vmatrix} 0 & \dots & 0 & h_k & \dots & & h_2 & h_1 & h_0 \\ 0 & \dots & h_k & h_{k-1} & \dots & & h_1 & h_0 & 0 \\ & & & & \dots & \dots & & & \\ h_k & \dots & h_1 & h_0 & 0 & \dots & & & 0 \end{vmatrix}, \quad (6.10)$$

где h_j – коэффициенты полинома $H(X)$ из выражения (6.8).

Пример. Задан ЦК (7, 4) порождающим полиномом $G(7, 4) = x^3 + x + 1$. Составить порождающую матрицу кода.

Решение. Первой строкой в матрице записывается порождающий полином (в двоичном представлении) с домножением его на оператор сдвига X^r для резервирования места под запись $r = 3$ проверочных символов.

Следующие $k - 1$ строк матрицы получаются путем последовательного циклического сдвига базового кодового слова (первой строки) матрицы на одну позицию вправо. Для заданного полинома и при $r = 3$ первая строка матрицы будет иметь вид $x^6 + x^4 + x^3$, или 1011000.

Следуя формуле (6.7), построим порождающую матрицу нашего кода:

$$G(7,4)=\left|\begin{array}{ccccccc|l} 1 & 0 & 1 & 1 & 0 & 0 & 0 & \text{1-я строка} \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & \text{2-я строка} \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & \text{3-я строка} \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & \text{4-я } (k=4) \text{ строка} \end{array}\right|$$

Для построения порождающей матрицы, формирующей *разделимый блочный код*, необходимо матрицу преобразовать к *каноническому* виду (см. формулу 4.2) путем линейных операций над строками. Канонический вид подразумевает, что матрица состоит из единичной подматрицы размерностью k и подматрицы P (в соответствии с выражением (4.2)) размерностью $k \times r$.

После выполнения упомянутых линейных преобразований мы получим следующую матрицу:

$$G_k(7,4)=\left|\begin{array}{cccccccl} 1 & 0 & 0 & 0 & 1 & 0 & 1 & \text{строка 1 + строка 3 + строка 4} \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & \text{2 + 4} \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & \text{3 = 3} \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & \text{4 = 4} \end{array}\right|$$

Как видно, первая строка матрицы $G_k(7, 4)$ получена поразрядным сложением по модулю 2 1, 3 и 4-й строк матрицы $G(7, 4)$. Принцип получения остальных строк понятен из пояснений справа к каждой строке матрицы $G_k(7, 4)$.

Проверочная матрица $H_{7,4}$ размерностью $r \times n$ (для рассматриваемого примера: 3×7) может быть получена из порождающей матрицы канонического вида путем дополнения проверочной подматрицы единичной матрицей размерности $r \times r$:

$$H_{7,4}=\left|\begin{array}{ccc} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{array}\right|$$

или в каноническом виде (см. выражение (6.8)):

$$(H_{7,4})^{\mathrm{T}} = \left| \begin{array}{cccccccc} 1 & 1 & 1 & 0 & & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & & 0 & 0 & 1 \end{array} \right|.$$

Вернемся к последней части рассмотренного примера. Мы получили значение синдрома ошибки: $S_r = 110$.

Вспомним, что в силу выражений (4.3)–(4.7) ненулевой синдром всегда равен сумме по модулю 2 тех векторстолбцов матрицы *H*, номера которых соответствуют номерам ошибочных битов в слове Y_n.

Понятно, что при одиночной ошибке декодирование синдрома должно указывать на соответствующий столбец проверочной матрицы. Для нашего примера – третий (такое же значение имеет третья строка матрицы $H_{7,4}$). Отсюда следует, что вектор ошибки $E_7 = 0010000$.

Воспользовавшись формулой (6.5), исправим ошибку:

$$\begin{array}{r} X_n = Y_n + E_n = 10\underline{\mathbf{1}}1110 \\ + \\ \underline{0010000} \\ 1001110. \end{array}$$

6.2. Практическое задание

1. Задание выполняется по указанию преподавателя в соответствии с вариантом из табл. 6.2, из которого выбирается порождающий полином ЦК, а по значению соответствующего ему значения r – длина k информационного слова X_k. Полагаем, что каждый полином соответствует коду, обнаруживающему и исправляющему одиночные ошибки в кодовых словах. Определить параметры (n, k)-кода для своего варианта. Основой задания является разработка приложения.

2. Составить порождающую матрицу (n, k)-кода в соответствии с формулой (6.7), трансформировать ее в каноническую форму и далее – в проверочную матрицу канонической формы.

3. Используя порождающую матрицу ЦК, вычислить избыточные символы (слово X_r) кодового слова X_n и сформировать это кодовое слово.

Таблица 6.2

Варианты задания

Вариант	Количество избыточных символов кода, r	Полином
1	4	$x^4 + x + 1$
2	4	$x^4 + x^2 + 1$
3	5	$x^5 + x^2 + 1$
4	5	$x^5 + x^3 + 1$
5	5	$x^5 + x^3 + x^2 + x + 1$
6	5	$x^5 + x^4 + x^2 + x + 1$
7	5	$x^5 + x^4 + x^3 + x + 1$
8	5	$x^5 + x^4 + x^3 + x^2 + 1$
9	6	$x^6 + x + 1$
10	6	$x^6 + x^3 + 1$
11	6	$x^6 + x^5 + 1$
12	6	$x^6 + x^4 + x^2 + x + 1$
13	6	$x^6 + x^4 + x^3 + x + 1$
14	6	$x^6 + x^5 + x^2 + x + 1$
15	6	$x^6 + x^5 + x^3 + x^2 + 1$
16	6	$x^6 + x^5 + x^4 + x + 1$

4. Принять кодовое слово Y_n со следующим числом ошибок: 0; 1; 2. Позиция ошибки определяется (генерируется) случайным образом.

5. Для полученного слова Y_n вычислить и проанализировать синдром. В случае, если анализ синдрома показал, что информационное сообщение было передано с ошибкой (или 2 ошибками), сгенерировать унарный вектор ошибки $E_n = e_1, e_2, \ldots, e_n$ и исправить одиночную ошибку, используя выражение (6.5); проанализировать ситуацию при возникновении ошибки в 2 битах.

6. Результаты оформить в виде отчета по установленным правилам.

ВОПРОСЫ ДЛЯ КОНТРОЛЯ И САМОКОНТРОЛЯ

1. Охарактеризовать основные параметры циклических кодов.

2. Сравнить основные параметры кодов Хемминга, итеративных и циклических кодов.

3. Пары десятичных чисел: 14, 11; 19, 15; 29, 13; 35, 45 преобразовать в двоичные числа и представить их в виде полиномов.

Выполнить арифметические операции над двоичными числами (по модулю 2) и полиномами.

4. Если кодовое слово ЦК имеет следующий вид: 11011010, то будут ли разрешенными кодовыми комбинациями следующие: 01101101, 10110101, 11011011?

5. Запишите в полиномиальном виде кодовое слово ЦК: 11011011, 1010101010, 0000000, 0000000001.

6. Чему равен результат операции по модулю два над полиномами $x^6 + x^3 + x^2$ и $x^4 + x^2 + x$, если такой операцией будет: сложение, вычитание, деление, умножение? Записать результат операции в двоичной форме.

7. Определить число и вычислить значения проверочных символов ЦК, если код задается полиномом $x^3 + x^2 + 1$, а информационное слово имеет вид: 1010; 1100.

8. Составить проверочную матрицу канонического вида, если порождающий полином имеет вид $x^3 + x^2 + 1$.

9. Определить минимальное кодовое расстояние ЦК, если он задается полиномом: $x^5 + x^4 + x^2 + x + 1$; $x^6 + x^4 + x^3 + x + 1$.

Лабораторная работа № 7

ПЕРЕМЕЖЕНИЕ/ДЕПЕРЕМЕЖЕНИЕ ДАННЫХ В ИНФОРМАЦИОННО-ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМАХ

Цель: приобретение практических навыков использования методов перемежения/деперемежения двоичных данных в информационных системах.

Задачи:

1. Закрепить теоретические знания по алгебраическому описанию и использованию методов перемежения/деперемежения двоичных данных в информационных системах.

2. Разработать приложение для реализации метода перемежения/деперемежения символов в сообщениях на основе двоичного алфавита.

3. Результаты выполнения лабораторной работы оформить в виде описания разработанного приложения, методики выполнения экспериментов с использованием приложения и результатов эксперимента.

7.1. Теоретические сведения

7.1.1. Описание и основные свойства перемежителей/деперемежителей

Проанализированные и исследованные нами коды, как и большинство других кодов, которые были разработаны для увеличения надежности каналов передачи и хранения информации, наиболее эффективны, когда возникающие ошибки статистически независимы, т. е. вероятность передачи (хранения в памяти) любого символа одинакова. Однако довольно часто распределение ошибок носит взаимозависимый характер. В таких случаях говорят о *группах* (или *пакетах*) ошибок. Такие ошибки характерны и для каналов передачи, и для устройств памяти (главным образом магнитной и полупроводниковой; см., например, [9–10, 19–22]).

Существуют специальные коды, корректирующие пакетные ошибки, однако на практике чаще используют перемежение/деперемежение совместно с традиционными кодами.

Идея перемежения/деперемежения состоит в следующем. Если биты каждого кодового слова X_n передаются не в обычной последовательности, а через интервалы, превышающие ожидаемую *длину пакета ошибок* (в промежутки между битами одного слова вставляются биты других кодовых слов), то при возникновении такого типа ошибки обратная перемежению операция – деперемежение – разнесет («размажет») группу ошибок по всей совокупности кодовых слов, составляющих данное сообщение.

Длина пакета в нашем случае – это число рядом расположенных ошибочных битов.

Например, если X_n = 1011111, а Y_n = 1000011, то длина пакета ошибок составляет 3 бита (ошибочные биты подчеркнуты).

Предложено много алгоритмов перемежения/деперемежения. Наиболее простыми являются *блочные* [5]. При блочном перемежении входные биты делятся на блоки, которые последовательно записываются в строки некоторой таблицы, приведенной для наглядности на рис. 7.1.

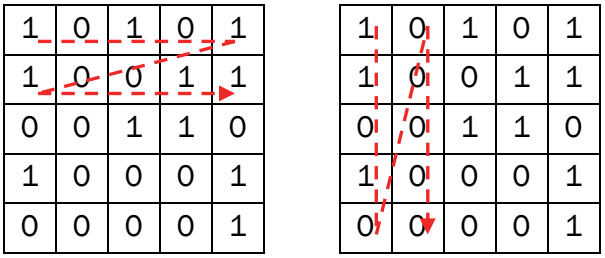


Рис. 7.1. Пояснение принципа блочного перемежения

Передаваемая последовательность (1010110011…) делится на блоки по 5 битов. Каждый блок записывается в отдельную строку таблицы по порядку. Сообщение для передачи или хранения формируется при считывании символов из таблицы по столбцам: 11010000001… . Деперемежение производится в обратной последовательности. Для данного примера *глубина перемежения* (разница между позициями одного и того же символа до и после перемежения) равна 4: например, второй символ после перемежения станет шестым. Особенностью является неизменная позиция первого символа.

В общем случае выбор глубины перемежения зависит от двух факторов. С одной стороны, чем больше расстояние между сосед-

ними символами, тем большей длины пакет ошибок может быть исправлен. С другой стороны, чем больше глубина перемежения, тем сложнее аппаратно-программная реализация оборудования и больше задержка сигнала.

Для борьбы с длинными пакетами ошибок желательно увеличивать размеры таблицы. Однако это приводит к увеличению задержки в отправке и декодировании сообщения.

7.1.2. Использование избыточного кодирования и перемежителей

Пример. Рассмотрим процесс передачи информации с использованием кода Хемминга и блокового перемежителя. Информационный поток (в виде отдельных информационных слов длиной 4 бита) на входе кодера Хемминга (используется код (7, 4)) имеет вид, как показано на рис. 7.2.

Инф. слово 1				Инф. слово 2				Инф. слово 3				Инф. слово 4				Инф. слово 5				Инф. слово 6				Инф. слово 7				...	Инф. слово *N*			
1	0	0	1	1	1	0	0	0	0	1	0	0	1	0	1	0	1	1	1	1	0	1	0	1	1	1	0	...	0	0	0	0

Рис. 7.2. Структура информационного потока на входе кодера

На выходе кодера сообщение будет иметь вид, представленный на рис. 7.3.

Кодовое слово 1							Кодовое слово 2							Кодовое слово 3						
1	0	0	1	1	1	0	1	1	0	0	0	1	0	0	0	1	0	1	1	0

Кодовое слово 4							Кодовое слово 5							Кодовое слово 6						
0	1	0	1	1	0	0	0	1	1	1	0	1	0	1	0	1	0	0	1	1

Кодовое слово 7							...	Кодовое слово *N*						
1	1	1	0	1	0	0	...	0	0	0	0	0	0	0

Рис. 7.3. Последовательность символов сообщения на выходе кодера

Используется таблица перемежения размером 7×7, которая после заполнения закодированными сообщениями будет иметь вид, показанный на рис. 7.4.

1	0	0	1	1	1	0
1	1	0	0	0	1	0
0	0	1	0	1	1	0
0	1	0	1	1	0	0
0	1	1	1	0	1	0
1	0	1	0	0	1	1
1	1	1	0	1	0	0

Рис. 7.4. Вид и содержание матрицы перемежения

После перемежения сообщение соответствует последовательности, изображенной на рис. 7.5.

Кодовая комбинация 1–7 после перемежения																			
1	1	0	0	0	1	1	0	1	0	1	1	0	1	0	0	1	0	1	1	1

Кодовая комбинация (продолжение) 1–7 после перемежения																			
1	0	0	1	1	0	0	1	0	1	1	0	0	1	1	1	1	0	1	1	0

Кодовая комбинация (окончание) 1–7 после перемежения							Кодовая комбинация 8–14 после перемежения								
0	0	0	0	0	1	0	...	...	...	...	...	...	...	...	...

Рис. 7.5. Закодированное сообщение после перемежения

Предположим, что в процессе передачи информации по каналу возник пакет ошибок (выделено черным) длиной 7 битов (рис. 7.6).

Кодовая комбинация (продолжение) 1–7 после перемежения																			
1	1	0	0	0	1	1	0	1	0	1	1	0	1	0	1	0	1	0	0	0

Кодовая комбинация (продолжение) 1–7 после перемежения																			
0	0	0	1	1	0	0	1	0	1	1	0	0	1	1	1	1	0	1	1	0

Кодовая комбинация (окончание) 1–7 после перемежения							Кодовая комбинация 8–14 после перемежения								
0	0	0	0	0	1	0	...	...	...	...	...	...	...	...	...

Рис. 7.6. Передаваемое сообщение, содержащее группу ошибок

Сообщение на выходе канала записывается по столбцам в матрицу (тех же размеров) деперемежения (рис. 7.7).

1	0	0	0	1	1	0
1	1	1	0	0	1	0
0	0	0	0	1	1	0
0	1	1	1	1	0	0
0	1	0	1	0	1	0
1	0	0	0	0	1	1
1	1	0	0	1	0	0

Рис. 7.7. Вид и содержание матрицы деперемежения

Из матрицы деперемежения двоичные символы сообщения считываются по строкам и поступают на декодер кода Хемминга (рис. 7.8).

Кодовая комб. 1							Кодовая комб. 2							Кодовая комб. 3						
1	0	0	0	1	1	0	1	1	1	0	0	1	0	0	0	0	0	1	1	0

Кодовая комб. 4							Кодовая комб. 5							Кодовая комб. 6						
0	1	1	1	1	0	0	0	1	0	1	0	1	0	1	0	0	0	0	1	1

Кодовая комб. 7							...	Кодовая комб. *N*						
1	1	0	0	1	0	0	...	0	0	0	0	0	0	0

Рис. 7.8. Ошибки разнесены по всему сообщению

После деперемежения пакет ошибок преобразован в одиночные (формально – независимые) ошибки кратности 1 для каждой из кодовых комбинаций кода Хемминга. Как помним, такие ошибки код в состоянии исправить. Информационный поток на выходе декодера кода Хемминга имеет вид в соответствии с рис. 7.9.

Инф. комб. 1				Инф. комб. 2				Инф. комб. 3				Инф. комб. 4				Инф. комб. 5				Инф. комб. 6				Инф. комб. 7				...	Инф. комб. *N*			
1	0	0	1	1	1	0	0	0	0	1	0	0	1	0	1	0	1	1	1	1	0	1	0	1	1	1	0	...	0	0	0	0

Рис. 7.9. Информационный поток на выходе декодера кода Хемминга

Рассмотренный метод блочного перемежения применяется в GSM. К числу других используемых на практике относятся следующие методы перемежения/деперемежения: псевдослучайный, *S*-типа (применяется в турбо-кодировании: CDMA (*Code Division Multiple Access*) – стандарт беспроводной связи *множественного доступа с кодовым разделением* каналов и др.); циклически-сдвиговый; сверточный; случайный; диагональный; многошаговый [5, 23, 24].

Перемежение (перестановка) символов также является основой некоторых классов криптографических методов [1], которые мы будем анализировать с практической точки зрения в другой части курса.

7.2. Практическое задание

1. Необходимо разработать авторское приложение в соответствии с целью лабораторной работы. По умолчанию используется блочный перемежитель/деперемежитель. По желанию студент может использовать иной. Задание выполняется по указанию преподавателя в соответствии с вариантом из таблицы.

Варианты задания

Вариант	Используемый корректирующий код	Длина пакета ошибок	Число столбцов в матрице	Длина сообщения, байт	Длина информационного слова, *k*, бит
1	Хемминга	3, 4, 5	5	15	6
2	Циклический	4, 5, 6	5	15	7
3	Итеративный	3, 5, 6	6	15	6
4	Хемминга	3, 4, 5	6	15	5
5	Циклический	4, 5, 7	6	15	6
6	Итеративный	3, 5, 7	7	12	5
7	Хемминга	3, 5, 7	7	13	6
8	Циклический	3, 6, 7	7	13	7
9	Итеративный	4, 5, 8	8	13	5
10	Хемминга	4, 6, 8	7	14	5
11	Циклический	3, 5, 7	7	14	6
12	Итеративный	3, 4, 8	8	14	7
13	Хемминга	4, 5, 6	8	15	5
14	Циклический	3, 5, 6	7	15	6
15	Итеративный	3, 5, 7	7	15	7
16	Хемминга	3, 6, 7	8	15	6

За основу разрабатываемого приложения может быть взято приложение из выполненной лабораторной работы, соответствующей заданному корректирующему коду.

2. Местоположение заданной группы ошибок выбирается (генерируется) случайным образом. Необходимо для группы ошибок каждой длины сгенерировать 30–40 случайных ситуаций. После деперемежения и исправления ошибок в сообщении сравнить

передаваемую последовательность и полученную после исправления ошибок. Проанализировать эффективность перемежения/деперемежения.

3. Результаты оформить в виде отчета по установленным правилам.

ВОПРОСЫ ДЛЯ КОНТРОЛЯ И САМОКОНТРОЛЯ

1. Пояснить назначение и особенности использования технологии перемежения/деперемежения данных в ИС.

2. Что такое группирующиеся ошибки и с чем, по Вашему мнению, связано их появление в каналах передачи данных, в полупроводниковой памяти, на магнитных носителях?

3. Что такое глубина перемежения и как она влияет на эффективность перемежения/деперемежения данных?

4. Пояснить принцип работы блокового метода перемежения.

5. Какая последовательность будет сформирована на выходе блокового перемежителя, если входная последовательность имеет вид 10111010011001100100, а таблица перемежения имеет размер 5×4, 5×5, 4×4, 4×5?

6. В чем состоит основное отличие блочного перемежителя от других Вам известных?

Лабораторная работа № 8
СЖАТИЕ/РАСПАКОВКА ДАННЫХ МЕТОДОМ БАРРОУЗА – УИЛЕРА

Цель: приобретение практических навыков использования метода Барроуза – Уилера для сжатия/распаковки данных.

Задачи:

1. Закрепить теоретические знания по алгебраическому описанию и использованию методов сжатия/распаковки (архивации/ разархивации) данных на основе метода Барроуза – Уилера (Burrows-Wheeler transform, BWT).

2. Разработать приложение для реализации метода Барроуза – Уилера.

3. Результаты выполнения лабораторной работы оформить в виде описания разработанного приложения, методики выполнения экспериментов с использованием приложения и результатов эксперимента.

8.1. Теоретические сведения

8.1.1. Описание и основные характеристики методов сжатия

Данная лабораторная работа открывает второй раздел практикума, относящийся к методам сжатия данных (англ. *data compression*), которые играют важную роль в решении проблемы защиты информации, ведь архивация данных необходима не только для экономии места на локальном дисковом носителе, но и для переноса информации, резервирования, резервного копирования и т. п. Исходя из этого, далее дадим краткую характеристику данному классу методов преобразования информации.

Сжатие информации является одним из способов ее кодирования.

Развитие методов сжатия данных имеют длинную историю, которая началась задолго до появления компьютеров и компьютерных сетей (см., например, [25]).

В основе сжатия данных, как одна из первопричин, лежит *избыточность*, что являлось предметом исследования в лабораторной работе № 3.

Основная *цель сжатия* – обеспечить более компактное представление данных, вырабатываемых источником, т. е. уменьшить физический объем сообщений, генерируемых источником, и сократить время его передачи (читай – стоимость) по каналам связи. *Фундаментальная теорема К. Шеннона о кодировании информации* утверждает, что «стоимость кодирования всегда не меньше энтропии источника, хотя может быть сколь угодно близка к ней» [26–27]. Поэтому для любого алгоритма сжатия всегда имеется некоторый предел *степени* (или *эффективности*) *сжатия*, определяемый *энтропией входного потока* (или *сжимаемого сообщения*).

Все алгоритмы сжатия преобразуют входной поток данных, минимальной единицей которых является бит, а максимальной – байт или несколько байт. Основными техническими характеристиками процессов сжатия и результатов их работы являются:

• *степень сжатия* (англ. *compress rating*), или отношение R (англ. *ratio*) объемов исходного (до сжатия, $V_{дс}$) и результирующего (после сжатия, $V_{пс}$) потоков данных (сообщений);

• *скорость сжатия* – время, затрачиваемое на сжатие некоторого объема информации входного потока до получения из него эквивалентного выходного потока;

• *качество сжатия* – величина, показывающая, насколько сильно сжат выходной поток при помощи применения к нему повторного сжатия по этому же или иному алгоритму.

Степень сжатия R обычно оценивается следующим образом:

$$R_1 = (V_{пс} / V_{дс}) \cdot 100\%,$$

$$R_2 = (V_{дс} - V_{пс}) / V_{дс} = (1 - R_1) \cdot 100\%.$$

Первое отношение показывает, какую часть объема сообщения (файла) до сжатия занимает сообщение (файл) после сжатия; второе отношение выражает основной физический смысл сжатия и показывает степень сжатия.

Что касается третьей из приведенных технических характеристик (качества сжатия), то она показывает, по существу, совместимость данного метода с другими. Это важно, если принять во внимание то обстоятельство, что практически все современные архиваторы основаны на использовании нескольких разных методов сжатия (кодирования).

Существуют различные подходы к реализации сжатия информации. Они отличаются математической базой, уровнем сложности (простоты) практической реализации, *форматом* кодируемого потока данных, *степенью соответствия сжимаемых и распакованных данных*.

По критерию, связанному с характером или форматом данных или степенью соответствия сжимаемых данных распакованным, все методы сжатия разделяют на два класса: *обратимое* и *необратимое сжатие*, или иначе: *сжатие без потерь* и *сжатие с частичной потерей информации* (англ. *lossy compression*).

Понятно, что недопустимы никакие потери при упаковке текстовых документов, кодов компьютерных программ, файлов баз данных.

Сжатие с потерей информации реализуется на основе таких известные форматов данных и алгоритмов сжатия, как JPEG и MPEG. Алгоритм JPEG используется при сжатии фотоизображений. Алгоритмы MPEG используют при сжатии видео и музыки.

Методы и алгоритмы сжатия с потерей информации применяют обычно для решения так называемых потребительских задач. Это значит, например, что если фотография передается для просмотра, а музыка для воспроизведения, то подобные алгоритмы применять можно. Если же они передаются для дальнейшей обработки, например для редактирования, то никакая потеря информации в исходном файле недопустима. Считают, что на фотографических иллюстрациях, предназначенных для воспроизведения на экране, потеря 5% информации не критична, а в некоторых случаях можно допустить и 20–25% уровень потерь.

В настоящее время можно встретить достаточно большое число архивирующих и сжимающих утилит, большинство из которых доступны для некоммерческого использования. Поддержка популярных форматов файловых архивов начинает включаться в разные утилиты и программы. Часто используемые форматы становятся стандартными форматами архивов (zip, arj, rar, ha, pak, cab и др.).

Из стандартных свойств архиваторов специалисты отмечают следующие:

- создание многотомных архивов с возможностью задания произвольного размера тома;
- создание самораспаковывающихся SFX-архивов;

- создание многотомных SFX-архивов;
- автоматическое удаление файлов после архивации;
- архивирование каталогов и дисков полностью с сохранением атрибутов файлов;
- помещение в архив авторских комментариев;
- паролирование доступа к архиву;
- поддержка защищенного режима (DPMI, VCPI), расширенной и расширяемой памяти;
- внедрение в архив циклических кодов контроля ошибок, позволяющих восстанавливать поврежденные архивы; заметим, что такой возможностью давно наделяются доступные архиваторы;
- выдача подробной информации по окончанию процесса архивации и по требованию (коэффициент сжатия, приблизительное время сжатия/распаковки, размеры файлов и т. п.);
- наличие справочной системы;
- относительно малый размер модуля архиватора.

8.1.2. Метод Барроуза – Уилера

BWT-преобразование (англ. *Burrows-Wheeler Transform*) – техника сжатия информации (в особенности текстов), основанная на преобразовании, открытом в 1983 г. BWT не сжимает данные в классическом понимании процесса, но преобразует блок данных в формат, исключительно подходящий для сжатия.

BWT оперирует сразу целым блоком данных, который выделяется из входного потока (сообщения).

Прямое преобразование (формально – сжатие) выполняется в 4 этапа:

1) выделяется блок данных (строка длиной k символов некоторого алфавита мощностью N), который обозначим символом M;

2) составляется таблица W_1 размером $k \times k$ всех циклических сдвигов входной строки M: $W_1 = (M)$;

3) производится лексикографическая (в алфавитном порядке) сортировка строк таблицы W_1, в результате чего получается таблица W_2 того же размера;

4) в качестве выходной строки (обозначим ее $BWT(M)$, z) выбирается последний столбец (M_k) таблицы W_2 преобразования и номер строки z, совпадающей с исходной строкой M. Как видим,

выходная строка (сжатое сообщение) всегда по объему превышает входную.

Более подробное и строгое математическое описание алгоритма прямого (как и обратного) преобразования можно найти, например, в статье [28]. Здесь для примера приведем фрагмент кода на языке C# для получения отсортированной матрицы (листинг 8.1; здесь матрица W_2 обозначена *SkT*).

```
for (int i=0; i<size; i++)
 {
  SkT[i] = String.Copy(CurStr);
  tempLetter = CurStr[0];
  Curstr = curstr.Substring(1, size-1) + tempLetter;
  initialPermutation.Text += SkT[i]+ '\n';
}
```

Листинг 8.1. Код программы
для получения отсортированной матрицы

Обратимость преобразования, т. е. возможность извлечения M из $BWT(M)$, i основана на рекурсивности преобразования. Вспомним, что общим для рекурсивных функций, рекурсивных алгоритмов и рекуррентных последовательностей является то, что для вычисления очередного значения функции, получения очередной реализации алгоритма, определения очередного члена последовательности необходимо обращаться к предшествующим их значениям, вычисленным раньше.

Итак, входной для обратного преобразования является информация вида $BWT(M)$, i. Это преобразование заключается в выполнении k одинаковых шагов, каждый из которых состоит из 2 операций, с целью воссоздания матрицы W_2:

1) в крайний справа пустой столбец матрицы записывается последовательность символов M_k;

2) производится лексикографическая сортировка столбцов заполненной части воссоздаваемой матрицы.

После k шагов матрица W_2 будет получена. Зная значение числа z, получаем входной блок M сообщения. Как видим, последующий вид матрицы определяется предыдущим ее состоянием. Листинг 8.2 содержит код программы для выполнения таких операций (обратим внимание, что воссоздаваемая матрица обозначена M).

```
for (int i=0; i<size; i++)
 {
  Sorting.Text += “Добавление столбца:” + ‘\n’;
     for (int j=0; j<size; j++)
      {
       M[j] = CurStr[j] + M[j];
       Sorting.Text += M[j] + ‘\n’;
      }
      Array.Sort(M);
 }
```

Листинг 8.2. Фрагмент кода для реализации обратного преобразования в алгоритме BWT

Пример. Пусть M = «столб»; $k = 5$.

Прямое преобразование:

$$W_1 = \begin{vmatrix} с & т & о & л & б \\ т & о & л & б & с \\ о & л & б & с & т \\ л & б & с & т & о \\ б & с & т & о & л \end{vmatrix} \Rightarrow W_2 = \begin{vmatrix} б & с & т & о & л \\ л & б & с & т & о \\ о & л & б & с & т \\ с & т & о & л & б \\ т & о & л & б & с \end{vmatrix}$$

Таким образом, имеем: M_k = «лотбс» ($k = 5$), $z = 4$ (исходное сообщение – это 4-я строка матрицы W_2).

Обратное преобразование:

$$W = \begin{vmatrix} л \\ о \\ т \\ б \\ с \end{vmatrix} \Rightarrow \begin{vmatrix} л & б \\ о & л \\ т & о \\ б & с \\ с & т \end{vmatrix} \Rightarrow \begin{vmatrix} л & б & с \\ о & л & б \\ т & о & л \\ б & с & т \\ с & т & о \end{vmatrix} \Rightarrow$$

$$\Rightarrow \begin{vmatrix} л & б & с & т \\ о & л & б & с \\ т & о & л & б \\ б & с & т & о \\ с & т & о & л \end{vmatrix} \Rightarrow \begin{vmatrix} л & б & с & т & о \\ о & л & б & с & т \\ т & о & л & б & с \\ б & с & т & о & л \\ с & т & о & л & б \end{vmatrix} \Rightarrow W = \begin{vmatrix} б & с & т & о & л \\ л & б & с & т & о \\ о & л & б & с & т \\ с & т & о & л & б \\ т & о & л & б & с \end{vmatrix}$$

Как видно, на первом шаге в крайний справа незаполненный столбец (пятый) воссоздаваемой матрицы W_2 (на приведенной условной схеме для математической корректности матрица обозначена символом W) записывается значение слова M_k = «лотбс», на втором шаге матрица сортируется (реально сортируются строки, содержащие только символы последнего столбца): получим в последнем столбце «блост». Первая операция закончена. Вторая операция: в крайний справа (теперь четвертый) столбец опять записывается входное сообщение «лотбс»; производится сортировка матрицы по значениям 2 последних столбцов. После выполнения указанных операций фактически будет воссоздана (если не возникли ошибки) матрица W_2. Зная $z = 4$, получим исходное сообщение: M = «столб».

Пример. $M = 101010$, $k = 6$.

Матрица W_1:

1	0	1	0	1	0
0	1	0	1	0	1
1	0	1	0	1	0
0	1	0	1	0	1
1	0	1	0	1	0
0	1	0	1	0	1

Матрица W_2:

0	1	0	1	0	1
0	1	0	1	0	1
0	1	0	1	0	1
1	0	1	0	1	0
1	0	1	0	1	0
1	0	1	0	1	0

Результат преобразования: $M_k = 111000$, $z = 4$ (либо 5, либо 6). Как видно, двоичное сообщение может на выходе создавать бинарные последовательности с блоками повторяющихся символов. Такие последовательности легко преобразуются в другие гораздо меньшей длины (с помощью, например, метода кодирования длин серий; см. пособие [5]).

8.2. Практическое задание

1. Разработать авторское приложение в соответствии с целью лабораторной работы. Входной блок данных может иметь произвольную длину.

2. С помощью приложения выполнить прямое и обратное преобразования 3 отдельных блоков данных, состоящих:

а) из собственного имени (можно краткий вариант записи);

б) собственной фамилии;

в) варианта в соответствии с таблицей ниже.

Можно использовать любой из известных методов сортировки символов массива.

Выполнить качественный сравнительный анализ длительности процессов прямого и обратного преобразований в зависимости от длины блока данных.

Варианты задания

Вариант	Входной блок данных, *M*
1	летоисчисление
2	времяпрепровождение
3	мультимиллионер
4	семенохранилище
5	песнетворчество
6	электрифицированный
7	водоворотоподобный
8	самообороноспособность
9	малосимпатичный
10	достопримечательность
11	сорокадневный
12	телегаммааппарат
13	полуторапроцентный
14	гидроаэроионизация
15	экстраординарный

3. Перевести первые 3 символа из блока данных, указанного в варианте таблицы, в бинарную последовательность в соответствии с кодами ASCII. Выполнить прямое и обратное преобразование. Оценить время прямого и обратного преобразований.

4. Результаты оформить в виде отчета по установленным правилам.

ВОПРОСЫ ДЛЯ КОНТРОЛЯ И САМОКОНТРОЛЯ

1. Сформулировать цели применения методов сжатия и архивирования данных.

2. Охарактеризовать основные технические характеристики процессов сжатия/распаковки и результатов.

3. Как можно рассчитать степень сжатия файла?

4. В каких случаях и почему применяется сжатие без потерь, а в каких – с потерей информации?

5. В чем сущность символ-ориентированных методов сжатия? Какие известные Вам методы относятся к этому классу?

6. Составить алгоритмы сжатия/распаковки данных методом Барроуза – Уилера.

7. Что поменяется, если процедуру формирования матрицы W_1 строить на основе циклических сдвигов вправо?

8. В каких известных архиваторах используется метод Барроуза – Уилера?

9. Как вы понимаете рекуррентность (рекурсивность) преобразования по методу Барроуза – Уилера?

10. Привести пример прямого и обратного преобразований сообщений:

а) 101010;

б) 1110011;

в) «колобок»;

г) «молоко».

Лабораторная работа № 9

СЖАТИЕ/РАСПАКОВКА ДАННЫХ НА ОСНОВЕ СТАТИСТИЧЕСКИХ МЕТОДОВ

Цель: приобретение практических навыков использования статистических методов Шеннона – Фано и Хаффмана (Shannon-Fano and Huffman coding) для сжатия/распаковки данных.

Задачи:

1. Закрепить теоретические знания по алгебраическому описанию и использованию методов сжатия/распаковки (архивации/разархивации) данных на основе методов Шеннона – Фано и Хаффмана.

2. Разработать приложение для реализации методов Шеннона – Фано и Хаффмана.

3. Результаты выполнения лабораторной работы оформить в виде описания разработанного приложения, методики выполнения экспериментов с использованием приложения и результатов эксперимента.

9.1. Теоретические сведения

9.1.1. Описание и основные свойства статистических методов сжатия

Мы неоднократно подчеркивали, что каждый из естественных языков обладает *избыточностью*. Среди европейских языков белорусский и русский обладают одним из самых высоких уровней избыточности. Об этом можно судить по размерам русского перевода английского текста. Обычно он примерно на 20–30% больше.

До появления уже упоминавшихся работ К. Шеннона кодирование символов алфавита при передаче сообщения по каналам связи осуществлялось одинаковым количеством битов, получаемым по формуле Хартли (см. формулу (2.2)). Позднее начали появляться способы, кодирующие символы разным числом битов в зависимости от вероятности появления их в тексте, подтверждение чему мы получили при выполнении лабораторной работы № 2. Таким образом, за счет использования для каждого значения байта кодов ASCII (символа алфавита) *кода различной длины* в соответствии с частостью (вероятностью появления этого символа в

сообщении) можно значительно уменьшить общий размер данных. Эта базовая идея лежит в основе алгоритмов статистических (вероятностных) методов сжатия: Шеннона – Фано и Хаффмана.

Статистические алгоритмы позволяют создавать более короткие коды для часто встречающихся и более длинные – для редко встречающихся символов алфавита или конкретного сообщения. В первом случае метод считается статическим статистическим, во втором – динамическим статистическим: вероятностные свойства символов подсчитываются для конкретного сообщения или потока данных.

Частота или вероятность появления того или иного символа алфавита в произвольном сообщении, лежащая в основе алгоритмов, дали название этим алгоритмам и соответствующим методам.

Иногда эти методы называют также *префиксными*.

К примеру, если имеется некоторый код, который записывается как $X_1 = A_1A_2$, и другой код – $X_2 = A_1$, то говорят, что X_2 является *префиксом* X_1. Или если $X_1 = 1010$, а $X_2 = 10101100$, то X_2 также является *префиксом* X_1.

Таким образом, использование описываемых методов предусматривает создание кодовой таблицы (подобно кодам ASCII или base64). Формально процедура сжатия (прямое преобразование) состоит в подстановке соответствующего бинарного кода вместо символа исходного алфавита и наоборот – при обратном преобразовании.

Методы относятся к классу «сжатие без потерь». Различие между двумя рассматриваемыми методами состоит лишь в особенностях формирования таблицы бинарных кодов. При формировании этой таблицы для обоих методов можно воспользоваться статистическими свойствами алфавитов, полученными при выполнении лабораторной работы № 2.

Основополагающая идея методов была предложена К. Шенноном в его известной работе [26] (полезно ознакомиться с переводной версией этой работы, точнее: Ч. 1 «Дискретные системы без шума», п. «Представление операций кодирования и декодирования» [27]).

9.1.2. Метод Шеннона – Фано

Детально метод был описан Р. Фано, который опубликовал его в виде технического отчета [29].

Код Шеннона – Фано не является оптимальным (обеспечивает минимальную избыточность) в общем смысле, хотя и дает оптимальные результаты при некоторых распределениях вероятностей. Для одного и того же распределения вероятностей можно построить, вообще говоря, несколько кодов Шеннона – Фано, и все они могут дать различные результаты.

Итак, необходимо выполнить следующие действия:

1) подсчитать вероятностные параметры символов алфавита $A = \{a_i\}$ (реализуется статическая версия алгоритма);

2) отсортировать – обычно в порядке убывания (невозрастания, т. е. могут иметь место повторяющиеся значения) вероятностей $p(a_i)$; $p(a_i)$ – вероятность появления в сжимаемом сообщении на произвольной позиции символа a_i алфавита, т. е. создать таблицу символов алфавита, на основе которого генерируется сжимаемое сообщение;

3) каждому символу отсортированного множества поставить в соответствие бинарный код, для чего это множество (таблица) символов делится на две группы таким образом, чтобы каждая из групп имела приблизительно одинаковую суммарную частоту (вероятность). Очевидно, на первом шаге такая суммарная вероятность в каждой из групп должна быть максимально близка к 0,5. Первому из полученных подмножеств устанавливается первый символ бинарного кода: 0, второй – 1 (или наоборот). Для вычисления следующих битов кодов данная процедура повторяется рекурсивно для каждого из полученных на текущем шаге подмножеств, в котором содержится больше одного символа. Получим таблицу, в которой длина кодовых комбинаций меняется от минимального (l_{min}) до максимального (l_{max}) значений.

Пример. Имеем алфавит $A\{a_i\}$, $i = [1, \ldots, 10]$, $N(A) = 10$ – мощность алфавита. При этом получены следующие вероятности для символов алфавита:

$p(a_1) = 0{,}05$	$p(a_6) = 0{,}12$
$p(a_2) = 0{,}10$	$p(a_7) = 0{,}05$
$p(a_3) = 0{,}03$	$p(a_8) = 0{,}10$
$p(a_4) = 0{,}20$	$p(a_9) = 0{,}15$
$p(a_5) = 0{,}15$	$p(a_{10}) = 0{,}05$

Далее рассмотрим процесс создания таблицы кодов. Отсортируем таблицу символов в порядке убывания вероятностей. Разделим ее на две части (два подмножества), как показано ниже (отделены гори-

зонтальной линией). Видно, что сумма вероятностей в обоих подмножествах одинакова и равна 0,5. Символам верхней части общей таблицы определим старший символ кода (1), нижней части – 0:

$p(a_4) = 0{,}20$	1
$p(a_5) = 0{,}15$	1
$p(a_9) = 0{,}15$	1
$p(a_6) = 0{,}12$	0
$p(a_2) = 0{,}10$	0
$p(a_8) = 0{,}10$	0
$p(a_1) = 0{,}05$	0
$p(a_7) = 0{,}05$	0
$p(a_{10}) = 0{,}05$	0
$p(a_3) = 0{,}03$	0

Далее в качестве исходного рассматриваем каждое из двух подмножеств (на текущем шаге). Выполняем рекурсивно одну и ту же процедуру. В частности, для первых трех символов таблицы (для первого подмножества) имеем после следующей итерации (ее деления на два меньших подмножества и определения следующих бинарных символов кода):

$p(a_4) = 0{,}20$	11
$p(a_5) = 0{,}15$	10
$p(a_9) = 0{,}15$	10

Последняя таблица в одном из подмножеств (первая строка) содержит только один элемент. Кодирование этого элемента закончено. Далее делим на два нижнюю часть последней таблицы и получим очередные символы соответствующих кодов:

$p(a_5) = 0{,}15$	101
$p(a_9) = 0{,}15$	100

Далее переходим к кодированию символов после первого деления исходной таблицы (начиная с символа $p(a_6)$).

Выполнив стандартные операции, получим таблицу бинарных кодов, как показано ниже:

$p(a_4) = 0{,}20$	11
$p(a_5) = 0{,}15$	101
$p(a_9) = 0{,}15$	100
$p(a_6) = 0{,}12$	011

$p(a_2) = 0{,}10$	010
$p(a_8) = 0{,}10$	0011
$p(a_1) = 0{,}05$	0010
$p(a_7) = 0{,}05$	0001
$p(a_{10}) = 0{,}05$	00000
$p(a_3) = 0{,}03$	00001

Или после расположения символов в порядке возрастания индексов символов алфавита:

$p(a_1) = 0010$	$p(a_6) = 011$
$p(a_2) = 010$	$p(a_7) = 0001$
$p(a_3) = 00001$	$p(a_8) = 0011$
$p(a_4) = 11$	$p(a_9) = 100$
$p(a_5) = 101$	$p(a_{10}) = 00000$

Как видим, для рассмотренного примера получена минимальная длина кода, равная 2 битам ($l_{min} = 2$), и максимальная длина, равная 5 битам ($l_{max} = 5$). Действительно, символам с б*о*льшими вероятностями соответствуют коды меньшей длины ($l_{min} = 2$) и наоборот ($l_{max} = 5$). Замечаем, что выполнено основное требование: все кодовые комбинации разные. И соблюдено «требование префикса»: ни одна из кодовых комбинаций меньшей длины не является началом кодовой комбинации большей длины.

Именно последняя таблица используется в неизменном виде (речь о кодах) в процессах прямого и обратного преобразований.

Алгоритм прямого преобразования: необходимо выполнить одну операцию: заменить символы входного сообщения соответствующими бинарными кодами.

Алгоритм обратного преобразования: на входе – сообщение в виде бинарной последовательности.

Шаг 1. Анализируются l_{min} начальных бинарных символов: осуществляется поиск в таблице соответствующего совпадения. Если такое будет найдено, то на выходе будет символ исходного алфавита с совпадающим кодом. После этого процедура повторяется, т. е. анализируются очередные l_{min} символов. Если не найдено в таблице совпадения, переходим к шагу 2.

Шаг 2. Длина анализируемой последовательности увеличивается на 1 бит: $l_{min} + 1$. Осуществляется поиск совпадающей бинарной комбинации такой же длины в таблице. Если такая комбинация существует, на выходе распаковщика формируется соответст-

вующий символ исходного алфавита, если нет – длина анализируемой последовательности увеличивается еще на один бит, и т. д.

По различным причинам при анализе очередной последовательности длиной l_{max} совпадение в таблице может быть не найдено. Для нейтрализации подобных коллизий архиваторы содержат средства контроля ошибок с помощью корректирующих кодов.

Пример. *Прямое преобразование*. Итак, требуется сжать сообщение X = «$a_1a_1a_9a_9a_9a_5a_1a_5a_5a_1$» (10 символов). Заменяя символы соответствующими им кодами из таблицы, на выходе получим: X_n = 00100010100100100101001010110100 10.

Как видим, общая длина (объем после сжатия $V_{пс}$) сообщения составляет 34 бита. Если бы каждый символ сообщения X заменялся некоторым кодом, подобным ASCII, то длина его составила бы 80 битов (10 · 8).

Обратное преобразование. На входе имеем бинарную последовательность Y_n = 00100010100100100101001010110100 10.

Шаг 1: анализируются начальные 2 (l_{min} = 2) символа этой последовательности: 00. Совпадающих комбинаций в таблице нет.

Шаг 2: длину анализируемой последовательности увеличиваем на один бит: 001. Совпадение не найдено.

Шаг 3: анализируем 4-битовую комбинацию: 0010. Этой комбинации в таблице соответствует символ исходного алфавита «a_1». На выходе распаковщика будет именно этот символ.

Анализируется очередные 2 символа ($l_{мин}$ = 2): 00 (шаг 1), и т. д.

Теперь предположим, что во входном сообщении изменился только один символ (первый):

$$Y_n = \mathbf{1}01000101001001001010010101101001 0.$$

А распаковщик не содержит средств для ее обнаружения.

В таком случае первой совпадающей комбинацией будет 101: «a_5». Следующей – «a_7», за ней – «a_2», и т. д.

Следуя логике рассуждений, нетрудно подсчитать коэффициент компрессии для нашего примера (см. формулы на с. 76):

$$R_1 = 34 / 80 \text{ либо } R_2 = (80 - 34) / 80 = 46 / 80.$$

9.1.3. Метод Хаффмана

Метод основан на алгоритме *оптимального префиксного кодирования* алфавита: исходный алгоритм Хаффмана является

оптимальным для посимвольного кодирования с известным входным распределением вероятностей, т. е. для отдельного кодирования несвязанных символов в таком потоке данных [30]. Отличается от метода Шеннона – Фано лишь в части кодирования символов исходного алфавита.

В данном случае бинарные коды создаются на основе дерева, ветви которого обозначаются бинарными символами.

Бинарным кодом символа исходного алфавита будет последовательность обозначений ветвей дерева от корня до листа, соответствующего этому символу. В основе бинарного кода лежит следующее положение.

Лемма. **Для любого заданного алфавита (источника) с $N > 2$ символами существует оптимальный двоичный код, в котором два наименее вероятных символа (слова) имеют одну и ту же длину и отличаются лишь последним битом [31].**

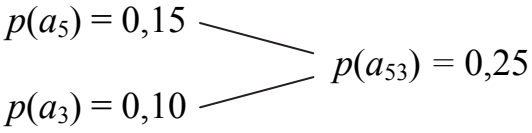
Построение дерева начинается с сортирования символов исходного алфавита в порядке убывания (невозрастания).

Далее выбираются два символа (a_i, a_j) с наименьшими вероятностями ($p(a_i)$, $p(a_j)$) и объединяются в узел. Ветви этого узла обозначаются «1» и «0». Этот узел рассматривается далее как новый, виртуальный символ (a_{ij}), которому будет соответствовать вероятность $p(a_{ij}) = p(a_i) + p(a_j)$. Такой виртуальный символ будет рассматриваться далее наравне с остальными символами исходного алфавита. Два его потомка из дальнейшего рассмотрения исключаются. Создаются новые узлы дерева по тому же принципу. Корень дерева образуют два символа с наибольшими вероятностями.

Пример. Пусть имеется алфавит из пяти символов, который в отсортированном виде выглядит так:

$p(a_4) = 0{,}35$
$p(a_2) = 0{,}20$
$p(a_1) = 0{,}20$
$p(a_5) = 0{,}15$
$p(a_3) = 0{,}10$

Строим дерево. Два нижних символа создают первый узел этого дерева, который обозначаем a_{53}. Ему соответствует вероятность $p(a_{53}) = 0{,}25$.



Верхней ветви этого узла будет соответствовать «1», нижней – «0».

Далее рассматриваем алфавит, состоящий из следующих символов: $p(a_4) = 0{,}35$, $p(a_{53}) = 0{,}25$, $p(a_2) = 0{,}20$, $p(a_1) = 0{,}20$, $p(a_5) = 0{,}15$.

Строго говоря, новая последовательность опять должна быть отсортирована. Далее снова нужно объединить два символа с наименьшими вероятностями, и т. д. С другой стороны, при построении дерева можно придерживаться следующего правила: на одном уровне иерархии дерева в узлы объединяются символы с примерно одинаковыми вероятностями; при обозначении ветвей двоичная единица присваивается символу с большей вероятностью.

Ниже представлена таблица (один из вариантов) соответствий между символами исходного алфавита и их бинарными кодами:

a_1	01
a_2	00
a_3	100
a_4	11
a_5	101

Как видно, для одного и того же алфавита по обоим рассмотренным статистическим методам могут быть созданы разные таблицы, т. е. одним и тем же символам могут соответствовать разные кодовые комбинации. Наилучшей (оптимальной или близкой к оптимальной) является та таблица, которой соответствует минимальное значение интегрального коэффициента C:

$$C = \sum p(a_i) \cdot l_i,$$

где $p(a_i)$ и l_i – соответственно вероятность появления символа и длина бинарного кода для этого символа.

Для данных из последнего примера имеем:

$$C = 0{,}20 \cdot 2 + 0{,}20 \cdot 2 + 0{,}10 \cdot 3 + 0{,}35 \cdot 2 + 0{,}15 \cdot 3 = 2{,}1 \text{ бита}.$$

Это означает, что для выполненной кодировки одному символу исходного алфавита в среднем соответствует 2,1 бита.

Д. Хаффман составил дерево для алфавита английского языка, которое является оптимальным. В таблице ниже представлены эти коды.

Бинарные коды английского алфавита в соответствии с деревом Хаффмана

Символ алфавита	Бинарный код
E	100
T	001
A	1111
O	1110
N	1100
R	1011
I	1010
S	0110
H	0101
D	11011
L	01111
F	01001
C	01000
M	00011
U	00010
G	00001
Y	00000
P	110101
W	011101
B	011100
V	1101001
K	110100011
X	110100001
J	110100000
Q	1101000101
Z	1101000100

Как видно, в текстах на английском языке наиболее часто встречается буква «e» (ей соответствует вероятность 0,127).

Хороший пример программной реализации рассмотренных методов можно найти в Интернет-источнике [32].

9.2. Практическое задание

1. Разработать авторское приложение в соответствии с целью лабораторной работы.

2. С помощью приложения выполнить прямое и обратное преобразования сообщения, состоящего из собственных имени и фамилии.

Можно использовать любой из известных методов сортировки символов массива. Метод кодировки (Шеннона – Фано, Хаффмана) использовать по указанию преподавателя.

При этом таблица отсортированных символов строится:

а) на основе данных, полученных в лабораторной работе № 2;

б) динамически, на основе анализа сжимаемого сообщения.

3. Определить эффективность (в сравнении с кодами ASCII) сжатия сообщения.

4. Результаты оформить в виде отчета по установленным правилам.

ВОПРОСЫ ДЛЯ КОНТРОЛЯ И САМОКОНТРОЛЯ

1. Что такое бинарное дерево, чем характеризуется его структура?

2. Какие коды называются префиксными?

3. Как, на Ваш взгляд, при увеличении мощности алфавита меняется его избыточность (или не меняется)?

4. В чем состоит суть кодирования по методам Шеннона – Фано и Хаффмана?

5. Построить бинарное дерево для метода Шеннона – Фано (сообщение – собственная фамилия).

6. В каких известных Вам архиваторах используются префиксные методы?

7. Сравнить эффективность кодирования по двум анализируемым методам. Показать на конкретном примере.

Лабораторная работа № 10
СЖАТИЕ/РАСПАКОВКА ДАННЫХ МЕТОДОМ ЛЕМПЕЛЯ – ЗИВА

Цель: приобретение практических навыков использования метод Лемпеля – Зива (Lempel-Ziv) для сжатия/распаковки данных.

Задачи:

1. Закрепить теоретические знания по алгебраическому описанию и использованию методов сжатия/распаковки (архивации/разархивации) данных на основе метода Лемпеля – Зива.

2. Разработать приложение для реализации метода Лемпеля – Зива.

3. Результаты выполнения лабораторной работы оформить в виде описания разработанного приложения, методики выполнения экспериментов с использованием приложения и результатов эксперимента.

10.1. Теоретические сведения

10.1.1. Описание и основные свойства и особенности реализации метода Лемпеля – Зива

В 1977 г. Авраам Лемпель и Якоб Зив выдвинули идею формирования «словаря» общих последовательностей анализируемых (сжимаемых) данных. При этом сжатие данных осуществляется за счет замены записей соответствующими кодами из словаря. Классический алгоритм Лемпеля – Зива – LZ77, названный так по году представления метода, формулируется следующим образом: «*если в проанализированном (сжатом) ранее выходном потоке уже встречалась подобная последовательность байт, причем запись о ее длине и смещении от текущей позиции короче, чем сама эта последовательность, то в выходной файл записывается ссылка (смещение, длина), а не сама последовательность*» (оригинальную статью см. по ссылке [33]).

Известный метод сжатия RLE, который заключается в записи вместо последовательности одинаковых символов одного символа и их количества, является подклассом LZ77.

Суть метода LZ77 (как и последующих его модификаций) состоит в следующем: упаковщик постоянно хранит некоторое количество последних обработанных символов в *буфере*. По мере обработки входного потока вновь поступившие символы попадают в конец буфера, сдвигая предшествующие символы и вытесняя самые старые. Размеры этого буфера, называемого также *скользящим словарем* (англ. *sliding dictionary*), варьируются в разных реализациях систем сжатия. Скользящее окно имеет длину n, т. е. в него помещается n символов, и состоит из двух частей:

- последовательности длины $n_1 = n - n_2$ уже закодированных символов (словарь);
- упреждающего буфера (буфера предварительного просмотра, *lookahead*) длиной n_2 – *буфера кодирования.*

Пусть к текущему моменту времени закодировано t символов: S_1, S_2, ..., S_t. Тогда словарем будут являться n_1 предшествующих символов: $S_{t-(n1-1)}$, $S_{t-(n1-1)+1}$, …, S_t.

В буфере находятся ожидающие кодирования (сжатия) символы S_{t+1}, S_{t+2}, …, S_{t+n2}. Если $n_2 \geq t$, то словарем будет являться вся уже обработанная часть входной последовательности.

Нужно найти самое длинное совпадение между строкой буфера кодирования, начинающейся с символа S_{t+1}, и всеми фразами словаря.

Эти фразы могут начинаться с любого символа $S_{t-(n1-1)}$, $S_{t-(n1-1)+1}$, …, S_t, выходить за пределы словаря, вторгаясь в область буфера, но должны лежать в окне. Буфер не может сравниваться сам с собой. Длина совпадения не должна превышать размера буфера. Полученная в результате поиска фраза $S_{t-(p-1)}$, $S_{t-(p-1)+1}$, $S_{t-(p-1)+(q-1)}$ кодируется с помощью двух чисел:

1) смещения (англ. *offset*) от начала буфера p;

2) длины соответствия, или совпадения (англ. *match length*) q.

Ссылки (p и q – указатели) однозначно определяют фразу. Дополнительно в выходной поток записывается символ s, следующий за совпавшей строкой буфера.

Длина кодовой комбинации (триады – p, q, s) на каждом шаге определяется соотношением

$$l(c) = \log_N n_1 + \log_N n_2 + 1, \qquad (10.1)$$

где N – мощность алфавита.

После каждого шага окно смещается на $q + 1$ символов вправо и осуществляется переход к новому циклу кодирования. Величина

сдвига объясняется тем, что мы реально закодировали именно $q + 1$ символов: q – с помощью указателя и 1 – с помощью тривиального копирования.

Передача одного символа в явном виде (s) позволяет разрешить проблему обработки еще ни разу не встречавшихся символов, но существенно увеличивает размер сжатого блока.

10.1.2. Примеры реализации метода

Пример 1. Используется алфавит $A = \{0,1,2,3\}$, $N = 4$, принимаем: длина словаря $n_1 = 15$, длина буфера данных (кодирования) $n_2 = 13$; для обозначения p и q используется четверичная система счисления. Тогда длина кодовой комбинации на каждом шаге составляет

$$l(c) = \log_N n_1 + \log_N n_2 + 1 = 2 + 2 + 1 = 5. \qquad (10.2)$$

Входной поток $S = 2000302013020130313031303130313333333$.

Вспомним соответствие между числами в десятичной и четверичной системах счисления:

$0_{10} = 00_4$, $1_{10} = 01_4$, $2_{10} = 02_4$, $3_{10} = 03_4$, $4_{10} = 10_4$, $5_{10} = 11_4$ и т. д.

Прямое преобразование. Шаг 1: состояние буфера отображает рис. 10.1.

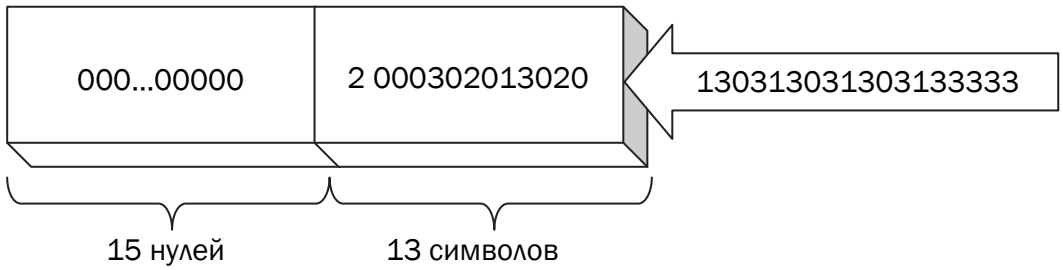


Рис. 10.1. Состояние буфера на первом шаге сжатия сообщения S

Анализируем 1-й символ в буфере кодирования на предмет соответствия (наличия) такого же символа или нескольких символов в словаре. Таких символов нет, следовательно, $p_1 = 0$. Длина повторения $q_1 = 0$. Таким образом, имеем следующую триаду: $(p_1, q_1, s_1) = (00\ 00\ 2)_4$ (нижний индекс справа от знака равенства означает основание системы счисления).

Шаг 2: состояние буфера отображает рис. 10.2.

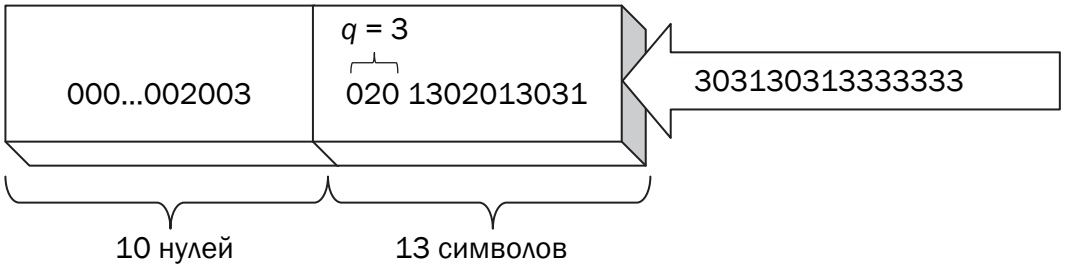


Рис. 10.2. Состояние буфера на втором шаге сжатия сообщения S

Находим повторение (000), длина этого повторения $q_2 = 3$. Поскольку в словаре нулевые символы записываются с 1-й по 14-ю позицию (для упрощения индексация ведется слева направо), то индексом p_2 (началом повторения) может быть выбрано любое число от 1 до 12: $1 \le p_2 \le 12$; выбираем $p_2 = 6$. Итак, получим следующую триаду: $(p_2, q_2, s_2) = (6, 3, 3) = (12\ 03\ 3)_4$.

Передвигаем сообщение в окне на $q_2 + 1 = 3 + 1 = 4$ позиции.

Шаг 3: состояние буфера отображает рис. 10.3.

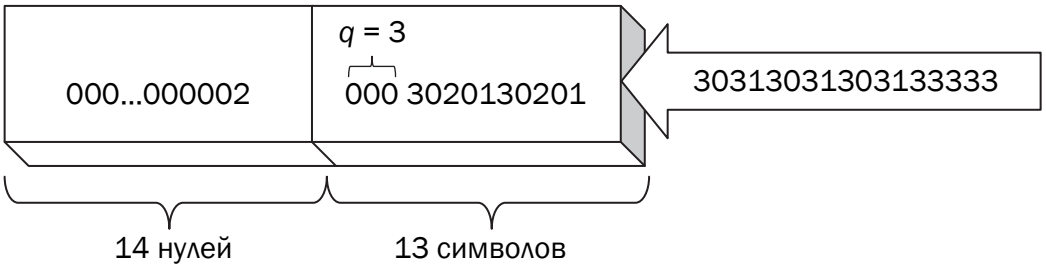


Рис. 10.3. Состояние буфера на третьем шаге сжатия сообщения S

В этой ситуации наиболее длинный повтор – 020, т. е. $q_3 = 3$, $p_3 = 10$. Передвигаем сообщение в окне на $q_3 + 1 = 4$ позиции. И получаем триаду: $(p_3, q_3, s_3) = (10, 3, 1) = (22\ 03\ 1)_4$.

Шаг 4: состояние буфера отображает рис. 10.4.

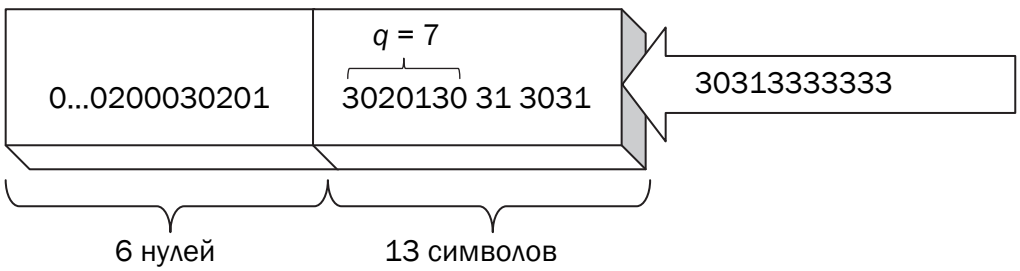


Рис. 10.4. Состояние буфера на четвертом шаге сжатия сообщения S

В этой ситуации наиболее длинный повтор – 3020130. Значит, $q_4 = 7$, $p_4 = 11$. Передвигаемся на $q_4 + 1 = 8$ и получаем триаду: $(p_4, q_4, s_4) = (11, 7, 3) = (23\ 13\ 3)_4$. И т. д.

Обратное преобразование. Имеем на входе «запакованное» сообщение: 00002 12033 22031 23133 30301 02013 32103… .

В исходном состоянии в окне (используется одно скользящее окно) записывают 15 нулей (такой выбрана длина окна). Результат анализа – символы исходного сообщения на основе выбранного алфавита.

Шаг 1: анализируем первые 5 символов (вспомним, $l(c) = 5$): 00002, следовательно, $q_1 = 0$ и $p_1 = 0$. В результате в младший (крайний справа) разряд окна записывается лишь символ 3 ($s_1 = 3$).

Шаг 2: анализируем следующие 5 символов: 12033, т. е. $p_2 = 6$, $q_2 = 3$. Поскольку в окне содержатся только нули, за исключением последнего разряда (15, индексация ведется как и в случае прямого преобразования), то мы можем сделать вывод, что наш повтор – это следующие три символа: 000. К ним дописывается еще последний символ из анализируемой триады ($s_2 = 3$). Таким образом, после двух шагов начальные символы распакованного сообщения будут такими: 20003. В остальных (начальных: с первого по десятый) разрядах окна будут нули.

Шаг 3: анализируем третью триаду сжатого сообщения: 22031. Следовательно, $p_3 = 10$, $q_3 = 3$. Разряды с десятого ($p_3 = 10$) по двенадцатый ($q_3 = 3$) окна содержат символы 020. Эти разряды допишутся справа к существующим символам окна и дополнительно к ним допишется символ 1 ($s_3 = 1$). Таким образом, в окне после этого будет записано: 000000200030201. И т. д.

Пример 2. Рассмотрим преобразование входного потока бинарного типа: S = 100100111011001010110011 0. Размер словаря и буфера данных равны 16.

Прямое преобразование. Буфер словаря изначально заполнен нулями. Буфер данных содержит первые 16 символов исходного сообщения. Так как размер буфера словаря и буфера данных имеет двухзначное число, то при формировании триады необходимо для записи значений p и q отводить 2 знака.

Шаг 1:

Буфер словаря	Буфер данных	Сообщение	Код
0000000000000000	1001001110110010	101100110	00,00,1

Из буфера данных берем первый слева символ «1» и осуществляем поиск данного символа в буфере словаря. Символ в словаре отсутствует. Формируем триаду с началом повторяющей последовательности (p), равным 0, длиной повторяющихся символов (q), равной 0, и следующим за повторяющейся последовательностью символом (c) «1». Сдвигаем окна на $q + 1$ позицию вправо.

Шаг 2:

Буфер словаря	Буфер данных	Сообщение	Код
0000000000000001	0010011101100101	01100110	14,03,0

Из буфера данных берем символ «0» и осуществляем поиск данного символа в буфере словаря. Символ в словаре имеется. Увеличиваем длину анализируемой последовательности на 1. Осуществляем поиск «00» в буфере словаря. Такая последовательность имеется. Увеличиваем длину последовательности на 1. Осуществляем поиск «001» в буфере словаря. Вновь последовательность имеется в буфере словаря. Опять увеличиваем длину последовательности на 1. Осуществляем поиск «0010» в буфере словаря. Такого сочетания символов в словаре не обнаружено. Формируем триаду с началом повторяющей последовательности (p), равным 14, длиной повторяющихся символов (q), равной 3, и следующим за повторяющейся последовательностью символом (c) «0». Сдвигаем окна на $q + 1$ позицию вправо.

Шаг 3:

Буфер словаря	Буфер данных	Сообщение	Код
0000000000010010	0111011001010110	0110	11,02,1

Шаг 4:

Буфер словаря	Буфер данных	Сообщение	Код
0000000010010011	1011001010110011	0	09,02,1

Конечным результатом сжатия будет набор полученных триад (кодов):

(00,00,1)(14,03,0)(11,02,1)(09,02,1)(06,05,1)(09,06,1)(02,01, …).

Обратное преобразование. В процессе восстановления используются только набор триад (кодов) и буфер. Принимаем

размер буфера и его начальное содержимое такое же, как и при сжатии.

Шаг 1:

Сообщение (распакованное)	Буфер словаря	Код
	0000000000000000	00,00,1

Анализируется триада (00,00,1). Начало повторяющейся последовательности (p) и ее длина (q) равны 0 – повторений нет. Добавляем к словарю символ «1» и сдвигаем окно словаря на q + 1 позицию вправо.

Шаг 2:

Сообщение (распакованное)	Буфер словаря	Код
1	0000000000000001	14,03,0

Анализируется триада (14,03,0). Повторяющаяся последовательность (p) начинается с четырнадцатой позиции и длиной (q), равной 3. Добавляем к словарю повторяющуюся последовательность «001» и символ «0». Сдвигаем окно словаря на q + 1 позицию вправо.

Шаг 3:

Сообщение (распакованное)	Буфер словаря	Код
10010	0000000000010010	11,02,1

Анализируется триада (11,02,1). Повторяющаяся последовательность (p) начинается с одиннадцатой позиции и длиной (q), равной 2. Добавляем к словарю повторяющуюся последовательность «01» и символ «1». Сдвигаем окно словаря на q + 1 позицию вправо. И т. д.

Как видно из приведенных примеров, алгоритм LZ77 обладает следующими очевидными недостатками:

1) невозможностью кодирования подстрок, отстоящих одна от другой на расстояние, большее длины словаря;

2) длина кодируемой подстроки ограничена размером буфера.

Однако если увеличивать размер словаря, то это приведет к увеличению времени преобразования и длины кодовых последовательностей.

В 1978 г. создателями алгоритма LZ77 был разработан усовершенствованный алгоритм: LZ78, лишенный названных недостатков. LZ78 не использует «скользящее» окно. Он хранит словарь из уже просмотренных подстрок. При старте алгоритма этот словарь содержит только одну пустую строку (строку длины нуль). Алгоритм считывает символы сообщения до тех пор, пока накапливаемая подстрока входит целиком в одну из фраз словаря. Как только эта подстрока перестанет соответствовать хотя бы одной фразе словаря, алгоритм генерирует код, состоящий из индекса строки в словаре, которая до последнего введенного символа содержала входную строку, и символа, нарушившего совпадение. Затем в словарь добавляется введенная подстрока. Если словарь уже заполнен, то из него предварительно удаляют менее всех используемую в сравнениях фразу.

Алгоритмы LZ-формата используются в тех случаях, когда требуется универсальное сжатие. Например, в известном стандарте модема V.42bis, протоколе передачи данных ZModem, форматах GIF, TIFF, ARC и других прикладных программах. Некоторые алгоритмы данного класса используются в дисковых утилитах сжатия типа DoubleSpace и Stacker, графических форматах типа PNG, а также в универсальных утилитах архивирования и сжатия, включая ZIP, GZIP и LHA.

10.2. Практическое задание

1. Разработать авторское приложение в соответствии с целью лабораторной работы. При этом предусмотреть возможность оперативного изменения размеров окон (n_1, n_2).

2. С помощью приложения выполнить прямое и обратное преобразования произвольного текста длиной несколько килобайт. Формат представления параметров p и q выбрать по указанию преподавателя.

3. Изменяя размеры окон, оценить скорость и эффективность (используя соотношения на с. 76) выполнения операций сжатия/распаковки.

4. Результаты оформить в виде отчета по установленным правилам.

ВОПРОСЫ ДЛЯ КОНТРОЛЯ И САМОКОНТРОЛЯ

1. К какому классу методов сжатия относится метод LZ и почему?

2. В чем заключается, на Ваш взгляд, оптимизация алгоритма LZ?

3. Какое значение имеет последний символ в сжатом текстовом сообщении методом LZ?

4. Какие модификации метода LZ77 Вам известны? В чем заключается особенность модификаций?

5. Пояснить физический смысл соотношения (10.1).

6. Как будет выглядеть сжатое сообщение, состоящее из символа моноалфавита при определенных значениях n_1, n_2?

Лабораторная работа № 11
СЖАТИЕ/РАСПАКОВКА ДАННЫХ АРИФМЕТИЧЕСКИМ МЕТОДОМ

Цель: приобретение практических навыков использования арифметических методов сжатия/распаковки данных.

Задачи:

1. Закрепить теоретические знания по алгебраическому описанию и использованию арифметических методов сжатия/распаковки (архивации/разархивации) данных.
2. Разработать приложение для реализации арифметических методов.
3. Результаты выполнения лабораторной работы оформить в виде описания разработанного приложения, методики выполнения экспериментов с использованием приложения и результатов эксперимента.

11.1. Теоретические сведения

11.1.1. Описание и основные свойства арифметических методов

Как мы установили, вероятностные (префиксные) методы являются достаточно простыми и эффективными: они основаны на использовании кодов переменной длины и для вероятностей появления символов алфавита, кратных степеням числа 2 (1/2, 1/4, 1/8 и т. п.), дают наилучшие результаты. При других значениях вероятностей, как правило, самый короткий код получается большим, чем двоичный логарифм этой вероятности (взятый с отрицательным знаком). Например, при $p(a_i)$ = 0,4 получим $-\log_2 0{,}15 = 1{,}32$. Понятно, что мы не можем закодировать этот символ только 2 битами (либо одним), т. е. решение не всегда является оптимальным. Анализируемого недостатка лишены арифметические методы.

При арифметическом сжатии (кодировании) текст представляется вещественными числами в интервале от 0 до 1. По мере анализа текста отображающий его интервал уменьшается, а количество битов для его представления возрастает. Очередные символы

текста сокращают величину интервала, исходя из значений соответствующих вероятностей.

Основная идея арифметического метода сжатия заключается в том, чтобы присваивать коды не отдельным символам, а их последовательностям.

Таким образом, как и во всех энтропийных алгоритмах, исходной является информация о частоте встречаемости каждого символа алфавита.

Алгоритмы прямого и обратного преобразований базируются на операциях с «*рабочим отрезком*».

Рабочим отрезком называется интервал [*a*; *b*] с расположенными на нем точками. Причем точки расположены таким образом, что длины образованных ими отрезков пропорциональны (или равны) частоте (вероятности) появления соответствующих символов.

11.1.2. Описание примера и особенности методов

Дана последовательность X_k = «*молоко*». Рассчитываем статистику:

$$p(м) = 0{,}1666$$
$$p(л) = 0{,}1666$$
$$p(к) = 0{,}1666$$
$$p(о) = 0{,}5$$

Шаг 0: строим *основной рабочий отрезок*, как показано на рис. 11.1.

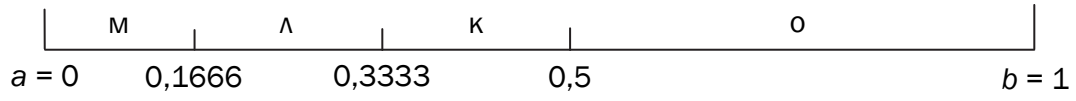


Рис. 11.1. Разметка и точки основного рабочего отрезка

Как показано на рис. 11.1, при инициализации алгоритма $a = 0$, $b = 1$ ($a_0 = 0$, $b_0 = 1$).

Прямое преобразование (сжатие). Один шаг сжатия (кодирования) заключается в простой операции: берется кодируемый символ, для него ищется соответствующий участок на рабочем отрезке. Найденный участок становится новым рабочим отрезком. Его тоже необходимо разбить с помощью точек.

Это и последующие разбиения отрезка (на шаге i) подразумевают определение новых значений верхней (H_i) и нижней (L_i) границ для всего участка и осуществляются по следующим правилам:

$$\left.\begin{aligned} H_i(\alpha_j) &= L_{i-1} + (H_{i-1} - L_{i-1}) \cdot H(\alpha_j)_0; \\ L_i(\alpha_j) &= L_{i-1} + (H_{i-1} - L_{i-1}) \cdot L(\alpha_j)_0, \end{aligned}\right\} \qquad (11.1)$$

где α_j – j-й символ сжимаемой последовательности, L_{i-1} и H_{i-1} – соответственно нижняя и верхняя границы рабочего отрезка на $(i-1)$-м шаге, $L(\alpha_j)_0$ и $H(\alpha_j)_0$ – соответственно исходные нижняя и верхняя границы символа α_j.

Шаг 1: в нашем примере на первом шаге берется первый символ последовательности: «*м*» (α_1 = «*м*») и ищется соответствующий участок на рабочем отрезке. Легко понять по рис. 11.2, что этот участок соответствует интервалу [0; 0,1666]. Он становится новым рабочим отрезком и опять разбивается согласно статистике и соотношениям (11.1):

$$H_1(м) = L_0 + (H_0 - L_0) \cdot H(\alpha_j = м)_0 = \\ = 0 + (0{,}1666 - 0) \cdot 0{,}1666 = 0{,}0277;$$

$$L_1(м) = L_0 + (H_0 - L_0) \cdot L(\alpha_j = м)_0 = \\ = 0 + (0{,}1666 - 0) \cdot 0 = 0;$$

$$H_1(л) = L_0 + (H_0 - L_0) \cdot H(\alpha_j = л)_0 = \\ = 0 + (0{,}1666 - 0) \cdot 0{,}3333 = 0{,}055555;$$

$$L_1(л) = L_0 + (H_0 - L_0) \cdot L(\alpha_j = л)_0 = \\ = 0 + (0{,}1666 - 0) \cdot 0{,}1666 = 0{,}0277.$$

После выполнения всех вычислений на первом шаге получим разметку рабочего отрезка ($a_1 = L_1 = 0$, $b_1 = H_1 = 0{,}1666$) в соответствии с рис. 11.2.

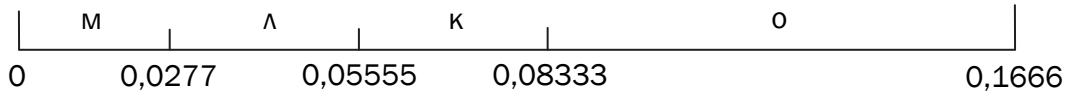


Рис. 11.2. Разметка и точки рабочего отрезка после анализа первого символа (*м*) последовательности «*молоко*»

Шаг 2: анализируем следующий символ входной последовательности (*о*). На новом рабочем отрезке этому символу соответствует интервал [$a_2 = L_2 = 0{,}083333$; $b_2 = H_2 = 0{,}1666$]. Для следующего шага он станет рабочим отрезком (рис. 11.3).

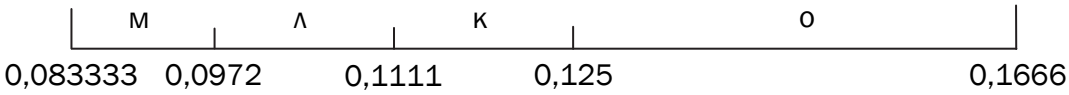


Рис. 11.3. Рабочий отрезок после анализа первых двух символов

Строго говоря, на первом шаге не было необходимости рассчитывать новые границы для каждого символа. Как видим, это только увеличивает время анализа. Достаточно было определить новые верхнюю и нижнюю границы лишь для очередного (на шаге 2) символа (*о*). Мы здесь привели расчеты только для того, чтобы лучше понять процесс вычислений.

Шаг 3: поскольку очередным из анализируемых символов будет буква «*л*», достаточно в соответствии с выражениями (11.1) определить новые границы:

$$H_3(л) = L_2 + (H_2 - L_2) \cdot H(\alpha_j = л)_0 =$$
$$= 0{,}08333 + (0{,}1666 - 0{,}08333) \cdot 0{,}3333 = 0{,}1111;$$
$$L_3(л) = L_2 + (H_2 - L_2) \cdot L(\alpha_j = л)_0 =$$
$$= 0{,}08333 + (0{,}1666 - 0{,}08333) \cdot 0{,}1666 = 0{,}972.$$

Новый рабочий отрезок определим как интервал [$a_3 = L_3 =$ $= 0{,}0972$; $b_3 = H_3 = 0{,}1111$], так как он соответствует символу «*л*». Вид разметки рабочего отрезка представлен на рис. 11.4.

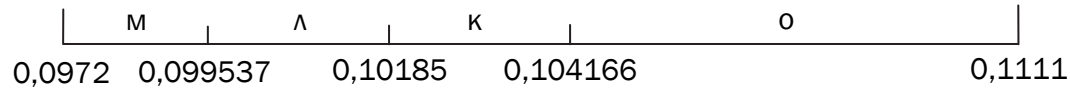


Рис. 11.4. Рабочий отрезок с разметками после анализа последовательности «*мол*»

Продолжаем анализировать символы сообщения. Находим на рабочем отрезке интервал [0,104166; 0,1111], который соответствует очередному символу: «*о*». Производим разметку нового рабочего отрезка, как показано на рис. 11.5.

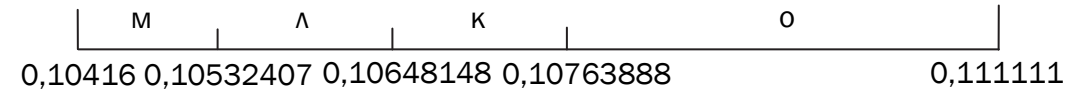


Рис. 11.5. Разметка рабочего отрезка на четвертом шаге

Следующий символ последовательности – «*к*». На рабочем отрезке данному символу принадлежит интервал [0,10648148;

0,10763888]. Новый рабочий отрезок имеет вид, как показано на рис. 11.6.

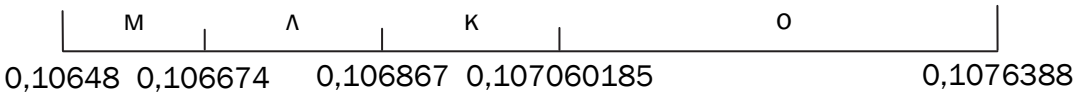


Рис. 11.6. Разметка рабочего отрезка на пятом шаге прямого преобразования

У нас остался последний символ: «*о*». Результатом кодирования цепочки символов является любое число с итогового рабочего отрезка [0,107060185; 0,10763888]. Обычно таким числом является *нижняя граница указанного отрезка*. Поступим и мы в соответствии с данным принципом. Возьмем число с меньшим количеством знаков после запятой.

Таким образом, итогом сжатия входной последовательности «*молоко*» будет число 0,107060185 (для упрощения дальнейших вычислительных операций округлим его до **0,1071**).

Обратное преобразование (декомпрессия). Для восстановления исходного сообщения необходима информация:

- о значении числа, являющегося итогом сжатия сообщения (в нашем случае 0,1071);
- количестве символов в сжатом сообщении;
- вероятностных параметрах всех символов исходного сообщения (таблица вероятностей).

Как и при сжатии, вначале необходимо начальный рабочий отрезок [0; 1) разбить на интервалы, длины которых равны вероятностям появления соответствующих символов, т. е. создать рабочий отрезок, полностью соответствующий рис. 11.1.

Анализ будем проводить с использованием конкретных данных, взятых из последнего примера.

Итак, в качестве исходного участка для обратного преобразования принимается исходный участок для прямого преобразования с одинаковыми точками его разбиения. Ниже он повторяется.

На каждом шаге обратного преобразования выбираем отрезок, в который попадает текущее число (код). Символ, который соответствует данному отрезку, является очередным символом восстановленного (распакованного) сообщения.

В общем случае код символа, восстанавливаемого на шаге i, вычисляется соотношением:

$$\text{код } i = [\text{код } (i-1) - L(\alpha_{i-1})_0] / [H(\alpha_{i-1})_0 - L(\alpha_{i-1})_0], \qquad (11.2)$$

где код $(i-1)$ – число, анализ которого производился на предыдущем шаге – $(i-1)$-м; $H(\alpha_{i-1})_0$ и $L(\alpha_{i-1})_0$ – соответственно верхняя и нижняя исходные границы символа сообщения, восстановленного на предыдущем шаге.

Шаг 1: определяем интервал, в который попадает начальное число: 0,1071 (код 1 = 0,1071). Это интервал [0; 0,166666] и ему соответствует символ «*м*». Данный интервал становится новым рабочим отрезком, а первый символ восстановленного сообщения – «*м*».

Шаг 2: производим вычисление в соответствии с выражением (11.2):

$$\text{код } 2 = [\text{код } 1 - L(\alpha_1 = м)_0] / [H(\alpha_1 = м)_0 - L(\alpha_1 = м)_0] =$$
$$= (0{,}1071 - 0) / (0{,}1666 - 0) = 0{,}643.$$

Вычисленный код соответствует символу «*о*».

Шаг 3: выполняем известное вычисление:

$$\text{код } 3 = [\text{код } 2 - L(\alpha_2 = о)_0] / [H(\alpha_2 = о)_0 - L(\alpha_1 = м)_0] =$$
$$= (0{,}643 - 0{,}5) / (1 - 0{,}5) = 0{,}286.$$

Полученное число соответствует символу «*л*».

Аналогичным образом производятся и последующие вычислительно-аналитические операции.

Другой вариант алгоритма распаковки «сжатого» сообщения основан на *свойстве рекуррентности прямого преобразования*. Производя на каждом шаге вычисление новых границ рабочего участка в соответствии с восстановленным на данном шаге символом (наподобие тех вычислений, которые мы выполняли при сжатии) и анализируя, на какой из отрезков этого участка попадает входное число (в нашем примере это 0,1071), восстанавливаем исходное сообщение.

Таким образом, на первом шаге это число соответствует букве «*м*», которая попадает в интервал [0; 0,1666]. Этот новый диапазон будет рабочим участком на шаге 2.

Так как нам известно, что символов в сообщении было 6, то на последнем шаге необходимо только определить недостающий символ. Таким символом становится символ «*о*». Полученное сообщение «*молоко*» и есть восстановленная («распакованная») последовательность.

Другие примеры и иные графические отображения результатов вычисления рабочих отрезков можно найти в пособии [5].

К числу основных особенностей методов можно отнести следующие:

- проанализированный алгоритм сжатия (кодирования) ничего не передает до полного завершения анализа всего текста;
- обычно результат представляется в формате целочисленной арифметики;
- требуемая для представления интервала $[H_i; L_i]$ точность возрастает вместе с длиной анализируемого текста; постепенное выполнение алгоритма помогает преодолеть эту проблему, при этом необходимо внимательно следить за возможностью переполнения [34]; упомянутые границы текущего интервала могут также представляться в виде целых чисел;
- как мы видим, арифметическое кодирование «работает» при помощи масштабирования или нормализации накопленных вероятностей, поставляемых моделью в интервалах $[H_i; L_i]$ для каждого передаваемого символа; если предположить, что H_i и L_i настолько близки друг к другу, что операция масштабирования «приводит» разные символы сообщения к одному целому числу, входящему в $[H_i; L_i]$, то дальнейшее кодирование продолжать невозможно. Следовательно, программа-кодировщик должна следить за тем, чтобы интервал $[H_i; L_i]$ не «слипался».

Особенности программной реализации метода, его качественные и количественные характеристики достаточно подробно описаны в [34].

11.2. Практическое задание

1. Разработать авторское приложение в соответствии с целью лабораторной работы.

2. С помощью приложения выполнить прямое и обратное преобразования сообщений в соответствии с таблицей.

Каждый студент выполняет задание, состоящее из двух частей. Первая часть предусматривает кодирование/декодирование сообщения, указанного в 2-м столбце, вторая часть – составного сообщения, полученного конкатенацией последовательностей из 2-го столбца, указанных в 3-м столбце. Например, для варианта

№ 1 такой конкатенацией будет последовательность «летоисчислениевремяпрепровождение».

Варианты заданий

Вариант	Сжимаемое сообщение	Строки из столбца 2
1	летоисчисление	1-2
2	времяпрепровождение	2-3
3	мультимиллионер	3-4
4	семенохранилище	4-5
5	песнетворчество	5-6
6	электрифицированный	6-7
7	водоворотоподобный	7-8
8	самообороноспособность	8-9
9	малосимпатичный	9-10
10	достопримечательность	10-11
11	сорокадневный	11-12
12	телегаммааппарат	12-13
13	полуторапроцентный	13-14
14	гидроаэроионизация	14-15
15	экстраординарный	15-1

3. Дать оценку возможности переполнения при выполнении вычислений.

4. Сравнить характеристики арифметического сжатия с вероятностными алгоритмами.

5. Результаты оформить в виде отчета по установленным правилам.

ВОПРОСЫ ДЛЯ КОНТРОЛЯ И САМОКОНТРОЛЯ

1. В чем заключается суть арифметического метода сжатия?

2. Пояснить принцип определения границ отрезков на каждом шаге (на каждом рабочем интервале) прямого и обратного преобразования.

3. В чем заключаются основные отличия метода арифметического сжатия от метода Хаффмана?

4. Какой недостаток, свойственный префиксным методам, отсутствует в арифметическом кодировании?

5. Насколько эффективен арифметический метод для кодирования данных с равномерно распределенными вероятностями появления символов?

6. Какие требования предъявляет арифметический метод к его программной реализации?

7. Какие сложности возникают при реализации алгоритма в программном коде? Как можно их решить?

8. Осуществить сжатие и распаковку сообщений арифметическим методом:

а) «мама мыла раму»;

б) «насос»;

в) «университет»;

г) «шалаш»;

д) «информатика»;

е) «246824328»;

ж) «101101001011010»;

з) собственные фамилия, имя, отчество.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Урбанович, П. П. Защита информации методами криптографии, стеганографии и обфускации: учеб.-метод. пособие для студентов специальности 1-98 01 03 «Программное обеспечение информационной безопасности мобильных систем», направления специальности 1-40 05 01-03 «Информационные системы и технологии (издательско-полиграфический комплекс)», специальности 1-40 01 01 «Программное обеспечение информационных технологий» специализации 1-40 01 01 10 «Программирование Интернет-приложений» / П. П. Урбанович. – Минск: БГТУ, 2016. – 220 с. (https://elib.belstu.by/handle/123456789/23763)

2. Информационная технология. Методы и средства обеспечения безопасности. Системы менеджмента информационной безопасности: ГОСТ ИСО/МЭК 27001–2006 [Электронный ресурс] / Федеральное агентство по техническому регулированию и метрологии. – Режим доступа: http://www.gociss.ru/doc/13.GOST_R_ISO_MEK_27001-2006.pdf. – Дата доступа: 01.12.2018.

3. Рекомендации по обеспечению безопасности информации в локальных сетях, подключенных к сети Интернет [Электронный ресурс] // Оперативно-аналитический центр при Президенте Республики Беларусь: сайт. – Режим доступа: https://oac.gov.by/ safety/ recommendations/state_organization.html. – Дата доступа: 01.12.2008.

4. Британский стандарт BS 7799-3:2006 – Руководство по управлению рисками информационной безопасности [Электронный ресурс] // Управление рисками информационной безопасности: сайт. – Режим доступа: http://mephi.edu/dist/magistracy/urib/ISRisks/ Page14. htm. – Дата доступа: 01.12.2018.

5. Урбанович, П. П. Защита информации и надежность информационных систем: пособие для студентов вузов специальности 1-40 05 01-03 «Информационные системы и технологии (издательско-полиграфический комплекс)» / П. П. Урбанович, Д. В. Шиман. – Минск: БГТУ, 2014. – 91 с. (https://elib.belstu.by/handle/ 123456789/ 23761)

6. The Base16, Base32, and Base64 Data Encodings [Электронный ресурс] // IETF Tools: сайт. – Режим доступа: https://tools.ietf.org/html/rfc4648. – Дата доступа: 01.12.2018.

7. Algorithm Implementation/Miscellaneous/Base64 [Электронный ресурс] // Wikibooks: сайт. – Режим доступа: https://en.wikibooks.

org/wiki/Algorithm_Implementation/Miscellaneous/Base64. – Дата доступа: 01.12.2018

8. Base64 кодер/декодер онлайн [Электронный ресурс] // Онлайн калькуляторы: сайт. – Режим доступа: https://calcus.ru/base64. – Дата доступа: 01.12.2018.

9. Урбанович, П. П. Информационная безопасность и надежность систем: учеб.-метод. пособие по одноименному курсу для студетов специальности 1-40 01 02-03 «Информационные системы и технологии» / П. П. Урбанович, Д. М. Романенко, Е. В. Романцевич. – Минск: БГТУ, 2007. – 87 с. (https://elib.belstu.by/handle/123456789/2937)

10. Урбанович, П. П. Избыточность в полупроводниковых интегральных микросхемах памяти / П. П. Урбанович, В. Ф. Алексеев, Е. А. Верниковский. – Минск: Навука і тэхніка, 1995. – 262 с. (https://elib.belstu.by/handle/123456789/24777)

11. Урбанович, П. П. Коррекция многократных ошибок в информационных словах итеративным кодом / П. П. Урбанович, Д. М. Романенко // Труды БГТУ. Сер. IV, Физико-математические науки и информатика. – 1998. – Вып. VI. – С. 82–86. (https://elib.belstu.by/handle/123456789/26713)

12. Multilevel turbocoding schemes on the basis of twodimensional linear iterative codes with diagonal checks / P. P. Urbanovich [et al.] // Przegląd Elektrotechniczny. – 2008. – R. 84, № 3. – S. 152–154. (https://www.researchgate.net/profile/Natallia_Patsei/publication/296559784_Multilevel_turbocoding_schemes_on_the_basis_of_twodimensional_linear_iterative_codes_with_diagonal_checks.pdf)

13. Urbanovich, P. P. The algorithm for determining of the errors multiplicity by multithreshold decoding of iterative / P. P. Urbanovich, M. F. Vitkova, D. M. Romanenko // Przegląd Elektrotechniczny. – 2014. – R. 90, № 3. – S. 235–238. (https://elib.belstu.by/handle/123456789/24782)

14. Виткова, М. Ф. Адаптивная система кодирования (декодирования) на основе многомерных итерактивных кодов многопороговых декодеров / М. Ф. Виткова, Д. М. Романенко // Труды БГТУ. – 2014. – № 6 (170): Физ.-мат. науки и информатика. – С. 116–120. (https://elib.belstu.by/handle/123456789/12198)

15. Multithreshold majority decoding of LDPC-codes / P. Urbanovich [et al.] // Informatyka, Automatyka, Pomiary w Gospodarce i

Ochronie Środowiska. – 2012. – № 4a. – S. 22–24. (https://elib.belstu.by/handle/123456789/3637)

16. Питерсон, У. Коды, исправляющие ошибки / У. Питерсон, Э. Уэлдон. − М.: Мир, 1976. – 593 с.

17. Мак-Вильямс, Ф. Дж. Теория кодов, исправляющих ошибки / Ф. Дж. Мак-Вильямс, Н. Слоэн. − М.: Связь, 1979. – 744 с.

18. Колесник, В. Д. Декодирование циклических кодов / В. Д. Колесник, Е. Т. Мирончиков. – М.: Связь, 1968. – 250 с.

19. Урбанович, П. П. Распределение ошибок в телефонных каналах передачи дискретной информации / П. П. Урбанович, Н. В. Пацей, В. В. Спиридонов // Известия белорусской инженерной академии. – 1997. – № 1 (3). – С. 24–26. (https://elib.belstu.by/handle/123456789/26760)

20. Урбанович, П. П. Модель распределения дефектных запоминающих элементов на кристаллах БИС ЗУ / П. П. Урбанович // Радиоэлектроника. Изв. вузов. – 1986. – Т. 29, № 9. – С. 92–95. (https://elib.belstu.by/handle/123456789/24780)

21. Верниковский, Е. А. Статистические характеристики отказов запоминающих элементов в микросхемах памяти / Е. А. Верниковский, П. П. Урбанович // Электронная техника. Сер. 3, Микроэлектроника. – 1989. – Вып. 1 (130). – С. 61–63. (https://elib.belstu.by/handle/123456789/25697)

22. Майоров, С. А. Определение характеристик распределения дефектов в микросхемах полупроводниковых ОЗУ / С. А. Майоров, П. П. Урбанович // Электронная техника. Сер. 3: Микроэлектроника. – 1992. – Вып. 1 (146). – С. 42–45. (https://elib.belstu.by/handle/123456789/26765)

23. Цифровые многопрограммные телевизионные системы, предназначенные для использования спутниками, работающими в диапазоне частот 11/12 ГГц: рекомендация МСЭ-R ВО.1516-1 [Электронный ресурс] // База данных ru.convdocs.org. – Режим доступа: http://ru.convdocs.org/docs/index-365397.html. – Дата доступа: 01.02.2019.

24. CDMA: кодирование и перемежение. [Электронный ресурс] // Сети / Network world: сайт. – Режим доступа: https://www.osp.ru/nets/2000/12/141602/. – Дата доступа: 01.02.2019.

25. Кричевский, Р. Е. Сжатие и поиск информации / Р. Е. Кричевский. – М.: Радио и связь, 1989. – 168 с.

26. Shannon, C. E. A Mathematical Theory of Communication / C. E. Shannon // Bell System Technical Journal. – 1948, July. – Vol. 27. – P. 379–423.

27. Шеннон, К. Математическая теория связи / К. Шеннон [Электронный ресурс] // Электронная библиотека УрГУ. – Режим доступа: http://shannon.usu.edu.ru/Shannon/shannon1948/node2. html#sec8. – Дата доступа: 23.12.2018.

28. Хмелёв, Д. В. Преобразование Барроуза – Уилера, массив суффиксов и сжатие словарей / Д. В. Хмелёв [Электронный ресурс] // Все о сжатии данных, изображений и видео: сайт. – Режим доступа: http://compression.ru/download/articles/bwt/khmelev_2003_ bwt. pdf. – Дата доступа: 23.12.2018.

29. Fano, R. M. The Transmission of information / R. M. Fano // Technical report #65. – MTI, 1949. – 34 p. (https://hcs64.com/files/ fano-tr65-ocr.pdf)

30. Huffman, D. A Method for the Construction of Minimum-redundancy Codes / D. Huffman // Proc. of the IRE. – 1952, Sept. – P. 1098–1101. (http://www.compression.ru/download/articles/huff/huffman_ 1952_minimum-redundancy-codes.pdf)

31. Галлагер, Р. Теория информации и надежная связь / Р. Галлагер. – М.: Советское радио, 1974. – 720 с. (http://padaread.com/ ?book=1696&pg=1)

32. Поляков, К. Сжатие данных. Тренажеры для изучения алгоритмов сжатия / К. Поляков [Электронный ресурс] // kpolyakov. spb.ru: сайт. – Режим доступа: http://kpolyakov.spb.ru/prog/compress. htm. – Дата доступа: 10.01.2019.

33. Lempel, A. Compression of individual sequences via variable-rate coding / A. Lempel, J. Ziv // IEEE Transactions on Information Theory. – 1978. – Vol. 24, no. 5. – P. 530–536.

34. Ochrona informacji w sieciach komputerowych / pod red. prof. P. Urbanowicza. – Lublin: Wydawnictwo KUL, 2004. – 150 p. (https:// elib.belstu.by/handle/123456789/27516)

ОГЛАВЛЕНИЕ

Учебное издание

Урбанович Павел Павлович
Шиман Дмитрий Васильевич
Шутько Надежда Павловна

ЛАБОРАТОРНЫЙ ПРАКТИКУМ ПО ДИСЦИПЛИНАМ «ЗАЩИТА ИНФОРМАЦИИ И НАДЕЖНОСТЬ ИНФОРМАЦИОННЫХ СИСТЕМ» И «КРИПТОГРАФИЧЕСКИЕ МЕТОДЫ ЗАЩИТЫ ИНФОРМАЦИИ»

В 2-х частях
Часть 1. Кодирование информации

Учебно-методическое пособие

Редактор *О. П. Приходько*
Компьютерная верстка *О. Ю. Шантарович*
Дизайн обложки *П. П. Падалец*
Корректор *О. П. Приходько*

Подписано в печать 22.05.2019. Формат $60\times84^{1}/_{16}$.
Бумага офсетная. Гарнитура Таймс. Печать ризографическая.
Усл. печ. л. 6,8. Уч.-изд. л. 7,0.
Тираж 150 экз. Заказ .

Издатель и полиграфическое исполнение
УО «Белорусский государственный технологический университет».
Свидетельство о государственной регистрации издателя,
Изготовителя, распространителя печатных изданий
№ 1/227 от 20.03.2014.
Ул. Свердлова, 13а, 220006, г. Минск.